Imperial College of Science, Technology and Medicine
University of London
Department of Computing

Domain-Based Security for Distributed Object Systems

Nikolaos Yialelis

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the Faculty of Engineering of the University of London, and for the Diploma of Imperial College of Science, Technology and Medicine.

August 1996

Corrected October 1996

Abstract

Advances in telecommunications technology have resulted in the proliferation of large distributed systems in commercial environments. Distributed systems, however, are vulnerable to unauthorised access to resources and compromise of information, either in terms of integrity or confidentiality. Furthermore, a distributed system may contain a large number of objects that are mutually suspicious making it hard to specify security policy. In addition, such a system may cross organisational boundaries necessitating decentralised security management.

This thesis proposes a security architecture for distributed object systems that supports access control services based on the concept of a *domain*. Domains can be used to group objects in a hierarchical structure, to apply a common security policy, to reflect organisational or geographical structure, or to partition the security management in order to cope with the complexity of large distributed systems.

An access control policy specifies, in terms of domains, what operations a set of subjects is permitted to perform on a set of targets. In a distributed system, however, a client often delegates access rights to a proxy server to perform operations on behalf of the client. As delegation of access rights should be controlled, the notion of the access control policy has been extended to deal with cascaded delegation.

The security architecture provides a high degree of access control and authentication transparency to the application level by utilising security agents on each host. A policy dissemination mechanism has been developed to propagate policies through hierarchical domain structures to the agents of the concerned objects and deal with changes in the domain structure.

The access control mechanism, which is based on the Access Control List (ACL) paradigm, enforces access control policies specified in terms of domains and deals with cascaded delegation of access rights.

As the access control decisions are based on domain membership, there is a need to efficiently authenticate domain membership as well as object and user identity. The proposed intra-realm authentication system is based on symmetric cryptography to minimise the encryption/decryption overhead. Verification of domain membership is based on statements issued by the domain service and translated by the authentication system into the keys of the verifiers. Similarly, verification of delegation is based on delegation tokens issued by the grantors and translated into the keys of the end-points.

ΣΤΟΥΣ ΓΟΝΕΙΣ ΜΟΥ
ΓΕΩΡΓΙΟ ΚΑΙ ΚΩΝΣΤΑΝΤΙΝΑ
(To my parents George and Constantina)

Acknowledgements

I am indebted to my supervisor Professor Morris Sloman. This work would not have been carried out without his guidance, suggestions and constructive criticism. My thanks are also due to Dr Jonathan Moffett, Dr Kevin Twidle, Damian A. Marriott and Emil Lupu who took time to read reports describing earlier stages of this work and provided useful feedback.

I also wish to acknowledge the help given me by many research students, research assistants and others members of the Distributed Software Engineering Section. Thank you Paris Bayias, S.C. Cheung, Paul Dias, Douglas Donaldson, Naranker Dulay, Hal Fossa, Dimitra Giannakopoulou, Celso Hirata, Christos Karamanolis, Masoud Mansouri-Samani, Nabor das Chagas Mendonca, Kaveh Moazami-Goudarzi, Keng Ng, Wai Leung Poon, Thanwadee Thanitsukkarn and Andrea Zisman.

My thanks are also due to Tracy Banton, Aspassia Daskalopoulou, Kostis Dryllerakis, Monica Leutner, Fotini G. Markopoulou-Kalamara, Stavros Menegos, Anne O’Neill, Yongyuth Permpoontanalarp and Nikos Scarmeas, who all have contributed considerable help during my studies at Imperial College.

I gratefully acknowledge financial support from the British Council, Swiss Bank Corporation (London), Esprit SysMan (7026) Project and Imperial College.

Table of Contents

Appendix

List of Figures

List of Tables

List of Abbreviations

AA	Authentication Agent
ACA	Access Control Agent
ACPL	Access Control Policy List
AR	Access Rule
AS	Authentication Service
CA	Certification Authority
CCPL	Candidate Channel Policy List
CCS	Capability Certificate Set
CF	Cryptographic Facility
CHID	Secure Channel Identifier
CID	Certificate Identifier
CRL	Certificate Revocation List
DAR	Delegated Access Rights
DID	Delegation token Identifier
DMCL	Delegation Membership Certificate Set
DPCL	Delegation Pseudo-Capability List
DRL	Delegation token Revocation List
DSE	Domain Scope Expression
DSSA	Distributed Systems Security Architecture
DT	Delegation Token
GCCS	Grantee Capability Certificate Set
GPCL	Grantee Pseudo-Capability List
MAC	Message Authentication Code
OID	Object Identifier
OMG	Object Management Group
ORB	Object Request Broker
PCF	Propagation Control Flag
PCL	Pseudo-Capability List
PDT	Preceding Delegation Token
PET	Policy scope Evaluation Token
PFS	Perfect Forward Security
PKC	Private-Key Certificate
PSCE	Pseudo-Capability Select Expression
RCPL	Resolved Channel Policy List
RMF	Reference Monitor Facility
RPD	Role Position Domain
SMCL	Subject Membership Certificate List
SN	Serial Number
TCB	Trusted Computing Base
UID	Unique Identifier
URD	User Representation Domain

Chapter 1

Introduction

Recent advances in telecommunications technology provide sufficient ground for the proliferation of large distributed systems in commercial environments. This, however, gives rise to information security issues. This thesis is mainly concerned with the enforcement of access control in distributed object systems and uses the notion of a domain as a means of dealing with the size and management distribution of large distributed systems. This chapter gives the motivation of this thesis and an outline of the main goals achieved.

1.1 Motivation and Aims

Large distributed systems typically involve large numbers of objects and human users and cross the boundaries of multiple organisations. These characteristics pose considerable difficulties in terms of management as it difficult for human managers to understand the operation of these systems and follow their evolution. Further, management should be decentralised to reflect the distributed nature of these systems.

The concept of the *management domain* has been developed and used in the Esprit-funded SysMan project (Becker et al. 1993). This concept facilitates the management of large, inter-organisational distributed systems and provides a means of partitioning management responsibility by grouping objects in hierarchical structures in order to specify *policies* (Sloman 1994) for object-based distributed systems.

In general, policies influence the interactions amongst objects in a system. Objects interact by invoking operations. In this thesis, an object that invokes an operation is referred to as the *subject* while the invoked object is referred to as the *target*. In the above-mentioned research project two types of policies have been developed:

- *obligation policies* that specify what operations human or automated managers should (or should not) perform upon certain events
- *authorisation or access control policies* which specify which operations subjects are authorised or forbidden to invoke on targets.

A large part of this thesis is concerned with *access control policies*. By controlling the operations a subject can invoke on a target, we actually control the ability of certain human users or system components to influence the state of resources that are provided by a system (for instance the *write* operation on a file). Further, we can control the information a human or a system component can acquire by receiving the reply of an operation (for instance the *read* operation on a file). There are applications where the ability to control the principals that can modify the state of some resources or get specific information is of critical importance.

Considerable work has been carried out in the framework of the SysMan project in terms of specification of access control policies (Becker et al. 1993; Sloman 1994) but there is no implementation of mechanisms to enforce these policies. The main objective of this thesis is to investigate the problems arising in a distributed environment with regard to the enforcement of access control policies and propose mechanisms that provide access control efficiently and transparently to the application level. Transparency is important as it facilitates the development of the application code without considering security issues.

In a centralised system, the enforcement of access control policies does not typically require the use of cryptographic techniques. This is because all data traffic takes place inside the local system software and hardware which has to be trusted to provide the necessary communication security amongst the colocated entities. Many applications require *data integrity* which means that the receiver of a message knows the identity of the sender. Other applications also require *data secrecy* or *confidentiality*. Further, in many cases it is important to enable the receiver to verify the freshness of a message, that is, the message is not a replay of an old message.

In a centralised system the underlying system can be trusted to provide data secrecy, integrity and freshness. This is not the case in a distributed environment where hosts are interconnected via a physically unprotected network which typically cannot guarantee communication security. Data transmitted over a network are subject to interference and wiretapping, so there is a need to employ cryptographic and authentication techniques that can provide the necessary level of message protection.

Concerning the enforcement of the access control policies, communication security is required when:

- The access control components receive a new policy or a request to withdraw or modify an active policy. They should ensure that a new policy or a request has been issued by the principal that has the right to specify policy for the affected objects.
- A subject invokes an operation on a target. The components controlling access to the target should verify the source, the integrity and freshness of the invocation request. Similarly, when the subject receives a reply, it has to ensure that it is fresh and has been sent by the target from which it is expected. Further, the invocation and reply data may carry sensitive information and thus message secrecy is sometimes required.
- Access control decisions may be based on attributes of the subject (e.g. current location). There is a need to ensure that statements about a subject's attributes are issued by a trusted principal.

The mechanisms providing communication security should be integrated in the system in such a way that their impact on the functionality and performance of the system is as small as possible. Another objective for the thesis is to provide mechanisms that can be implemented efficiently.

The access control policies are specified in terms of domains rather than in terms of individual entities, so there is a need to verify the domain membership of the subject in order to determine the policies enabling it to invoke operations. Further, as the domain structure is not static, a mechanism is required to dynamically evaluate the scopes of the policies. Policy scopes are specified by Domain Scope Expressions that make use of set operators such as union, intersection and difference. The policy scope evaluation is related to the policy dissemination issue in that the policies should be given to the components of the system that have to enforce them.

In distributed systems resources are distributed over multiple hosts and various tasks often result in cascaded invocations. This requires delegation of access rights which is related to the access control and authentication issues. The thesis analyses this problem and discusses mechanisms to enable the delegation of access rights.

We adopt the object-based paradigm. Most of the work in the area of distributed systems security deals with a simpler model where the main entities in terms of authentication and access control are users and hosts or at most application servers. A system providing security for distributed object systems has to consider a finer level of granularity since the basic entity is the object rather than the host or the server. This imposes additional difficulties as the population of the objects is much higher, which may result in additional overhead with regard to authentication and access control. The thesis analyses the trust relationships appearing in a distributed object system in order to avoid unnecessary cryptographic or authentication overheads.

1.2 Our Approach

We adopt an object-based view where each object is identified by a globally unique name and provides an interface which supports operations on its state. The CORBA-compliant Orbix^{TM} (IONA 1993) environment has been used as a distributed platform to implement a number of the components described in this thesis but the description of the architecture does not depend on that platform.

This thesis is concerned with access control policies suited for commercial distributed systems and it does not consider military-oriented access control. The access control enforcement mechanism is based on the Access Control List (ACL) paradigm. That is, access control decisions are made by components that reside on the target host and know the policies controlling access to the targets.

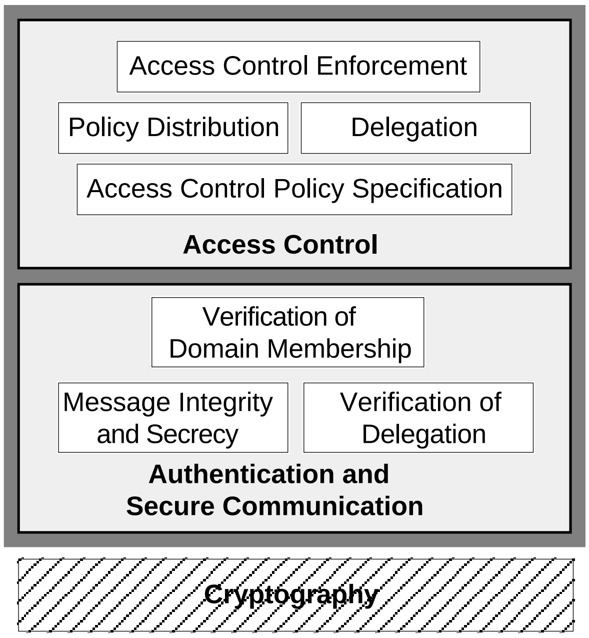

Access Control Enforcement
Policy Distribution
Delegation
Access Control Policy Specification
Access Control
Verification of Domain Membership
Message Integrity and Secrecy
Verification of Delegation
Authentication and Secure Communication
Cryptography

Figure 1.1 Main Issues Discussed in This Thesis

The fact that management is distributed means that different parts of a system may be managed by different authorities. This implies that we cannot base the access control decisions on membership statements issued by a single authority. In general, we avoid any centralisation of services as this does not facilitate the scalability, performance, reliability and security of the systems.

The main issues discussed in this thesis are depicted in Figure 1.1. Though the provision of communication security requires the use of cryptographic techniques, this thesis does not propose new solutions in this area. Instead, it uses well known techniques to provide the necessary communication security.

1.3 What is New in this Thesis

This thesis describes a security architecture that supports the enforcement of access control policies specified in terms of domains. A description of this architecture is also given in (Yialelis and Sloman 1996) and (Yialelis et al. 1996). The novel aspects of our work can be summarised as follows:

- The thesis shows how to specify delegation policy as an extension of the basic access control policy. The delegation mechanism supports cascaded delegation of access rights. The verification of delegation is based on symmetric cryptography and the use of relays.
- There is a need to distribute policies to the access control components. The thesis discusses a policy dissemination mechanism that copes with nested domain structures that change dynamically as well as Domain Scope Expressions. The effect is that the access control enforcement components are continuously aware of the policies applying to the objects for which they are responsible.
- As access control policies are specified in terms of domains, the access control decision making becomes more complex. The thesis shows how to use pseudo-capabilities as hints in a domain framework. Pseudo-capabilities reduce the access rule space that has to be searched by the target in order to make access control decisions. This mechanism also supports access control when delegation of access rights takes place.
- The thesis presents a domain membership verification mechanism that is based on the use of symmetric cryptography and relays in order to reduce the cryptographic overhead. Use of asymmetric cryptography would seriously affect the performance of the system, especially when target hosts accept a large number of invocations from multiple subjects.
- The provision of security requires an architecture that integrates the relevant services, namely domain, policy, access control and authentication services. The authentication system is based on symmetric cryptography and uses relays. Security agents are employed on each host in order to make the security services transparent to the application level and decrease the security components that have to be replicated over

multiple application servers. The authentication system supports the establishment of secure channels, delegation of access rights and domain membership verification. A *secure channel* between a subject and a target represents access control, authentication and cryptographic information. The access control policies enabling the subject to invoke operations on the target are determined when the channel is being established. Once a secure channel has been established, the involved objects can securely exchange messages and access control decisions are made on a per-invocation basis.

1.4 Thesis Outline

This chapter has discussed the motivation and the main objectives of this thesis. It has also stressed the importance of communication security in distributed systems and outlined the approach as well as the achievements of the thesis.

Chapter two gives the basic security aspects with regard to distributed systems that are relevant to this work. Specifically, it discusses access control, authentication and cryptography. It aims at introducing the main principles and terminology used in the following chapters of this thesis.

Chapter three discusses the work that has been done in the area of management using domains and policies. The emphasis is on the notions of domains and access control policies as this thesis builds on these two concepts. It also gives a brief description of the domain and policy services.

Chapter four discusses the issue of delegation of access rights in a distributed environment. It presents related work and emphasises the specification of policy to control delegation. It also introduces the notion of an extended access control policy which is used as a means of specifying access control and delegation policy in terms of domains.

Chapter five provides an overview of the security architecture that is described in detail in Chapters 6 and 7. It also discusses the trust model on which the architecture is based as well as related work.

Chapter six is concerned with the access control enforcement mechanism. It analyses the policy dissemination issue and proposes a technique to distribute access control policies to the relevant access control components. It discusses the access control decision mechanism and issues relevant to delegation, restricted access rights and Application Programming Interfaces (APIs).

Chapter seven describes how the authentication system supports secure channel establishment, verification of domain membership and delegation of access rights.

Chapter eight provides a critical assessment of the system and makes suggestions for further work.

Finally, the Appendix gives a detailed description of the policy propagation technique introduced in Chapter 6.

Chapter 2

Background

This chapter discusses security aspects with regard to distributed systems. It also introduces basic terminology that is used in the following chapters of this thesis. In particular, this chapter examines the issues of access control and authentication, and provides an overview of the basic cryptographic tools.

2.1 Security in Distributed Systems

Distributed computer systems have come to play an important role in many commercial activities. Many enterprises are heavily dependent on distributed systems which maintain and process sensitive information. Failure to protect this information, in terms of disclosure to competitors, malicious modification and loss can have undesirable effects on the operation of these enterprises. Consequently, information security is gradually becoming a crucial aspect of commercial distributed systems design.

Security is a well examined aspect of centralised systems but distributed systems exhibit a number of characteristics that make security a much harder issue to analyse. First, as it is difficult to provide physically secure communication, some sort of cryptographic tools have to be employed. Further, distributed systems consist of multiple nodes that are trusted to different extents and thus the notion of a *single* Trusted Computing Base (TCB) that is capable of preserving the secrecy and integrity of data does not make sense in distributed systems. In addition, these systems are typically managed by multiple managers representing different, potentially conflicting interests and may consist of a large number of entities. These factors make the management of security in distributed systems more complex than in centralised systems.

2.1.1 Information Security

Information security is concerned with:

- *Secrecy* or *confidentiality*. Information that is stored on a system or transmitted over communication links should not be disclosed to persons that are not authorised to access it.
- *Integrity*. Information should be protected from unauthorised alteration.
- *Availability*. Information systems have to be protected from denial-of-service attacks.
- *Accountability*. Human users and their agents should be accountable for their actions.
- *Non-repudiation*. A particular case of accountability where responsibility for actions cannot be denied.

2.1.2 Threats in a Distributed System

A distributed system is susceptible to a number of threats both from legitimate users of the system and from intruders. In (Woo and Lam 1992) two general types of threats are identified: *host compromise* and *communication compromise*.

The host compromise threat refers to various degrees of subversion of individual hosts. The intruder may manage to control partial state information or even achieve total control of a host.

Communication compromise refers to threats associated with message communication. If we assume that there is a flow of information from a source to a destination over a communication link, an opponent can mount attacks which can be categorised as follows:

- *Interception*. The opponent gains access to the data transmitted over the communication link. We can distinguish between two types of interception:
    - *Release of information*. The opponent obtains information transmitted over the link. A typical example is eavesdropping on a telephone line.
    - *Traffic analysis*. The opponent observes the message patterns and derive information about the identities and locations of the communicating parties, the message frequency and length, etc.
- *Interruption*. The opponent prevents the transmission of information.

- *Modification*. The opponent modifies the messages transmitted, alters their sequence or delays them.
- *Fabrication*. The opponent inserts information into the communication link. A special type of this attack is *replay of old messages* where the opponent copies messages transmitted over the link and re-transmits them in order to mislead the communicating parties.

The interception attack is considered to be a *passive attack* while all the others are considered to be active attacks. In general, passive attacks are difficult to detect but, in many cases can be prevented. On the other hand, active attacks are difficult to prevent since this would require adequate physical protection of the communication links. Active attacks, however, can be detected and in many cases measures can be taken to recover from their effects.

An opponent may use one or more of these attacks to inhibit the normal use of hosts or communication systems (denial of service attack). For instance, the interruption attack can be used to prevent the communication which is normally provided by a system to authorised entities.

2.1.3 Security Functionality

The main security functionality in a distributed system can be summarised as follows:

- *Access Control*. There is a need to protect resources against unauthorised access. The access control components decide whether a human user or a system entity can access a particular resource. This functionality is related to both the secrecy and integrity of information.
- *Authentication*. Verification of the identity of human users and system entities. This is of crucial importance in distributed systems due to the inherent ability of these systems to allow access to *remote* resources via physically untrusted communication environments.
- *Auditing*. Human users as well as system entities that access resources should be accountable. The audit components should record the identities and actions of them.
- *Communication security*. Communication over insecure links is typically the case in a distributed system. Thus there is a need to employ mechanisms that provide the required communication secrecy and integrity.
- *Non-repudiation*. For some applications it is important to provide evidence of actions. Typical examples of this are proof of receipt of a message or proof of sending of a message.

- *Security management.* This is the management of information related to the security of a system. Typically this information determines the security characteristics of a system.
- *Cryptography*. The provision of the above-mentioned functionality is based on cryptography which is essential in distributed systems where communication is based on insecure links.

Figure 2.1 depicts the dependencies amongst all these security-related features. This thesis is mainly concerned with access control, communication security and authentication. In the next sections these features are analysed further. Moreover, we outline the principles of cryptography and discuss some cryptographic tools.

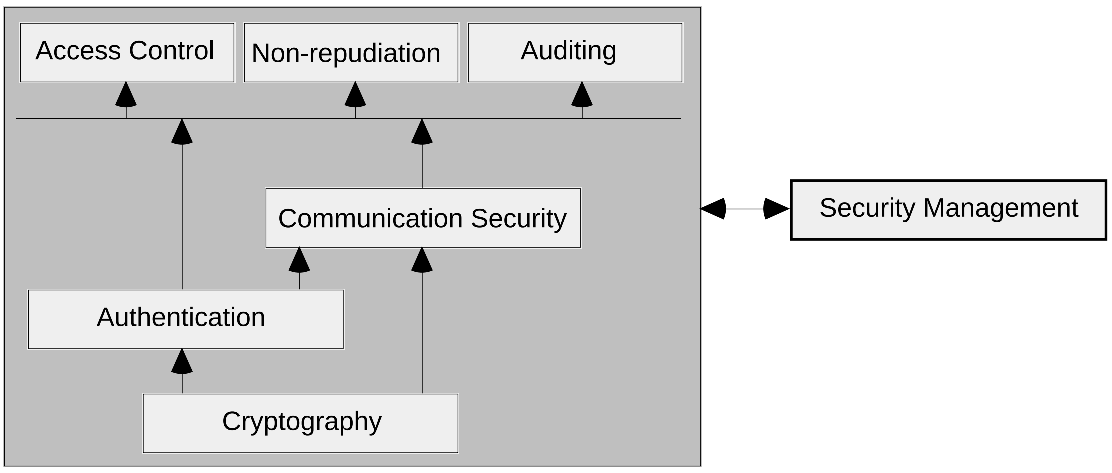


Figure 2.1 Security Functionality and Dependencies

2.2 Access Control Policy

2.2.1 Security Policy

We first outline the notion of a security policy as the access control policy of a system is considered to be part of its security policy.

In general, the term *security policy* refers to the set of rules, laws and practices that regulate how an organisation manages, protects, and distributes sensitive information (Olson and Abrams 1990). When a computer system maintains and manipulates information related to the operation of an organisation, the security policy also affects that system. According to (Clark and Wilson 1987) a security policy must specify the security goals the system must meet and the threats it must resist. This implies that a security policy should determine the type of secure communication required for various transactions, the type of authentication, auditing procedures, recovering techniques and access control.

Typically, a security policy is expressed initially in a *natural language* (English for instance). Natural languages, however, are often susceptible to multiple interpretations. Furthermore, natural language is not helpful enough in cases where we want to reason about the correctness or various properties of a policy. These weaknesses inflict the need for a precise formal language restatement in a formal security policy model (Olson and Abrams 1990)

The formal security policy is implemented by the *Trusted Computing Base* (TCB) of the system. The TCB consists of a number of components (implemented in hardware, firmware and software) which are responsible for enforcing the security policy. In a centralised system the notion of the TCB is clear and it is easy to identify which system components belong to it. In a distributed system which is managed by multiple human users, the notion of the TCB is more complex as different hosts are trusted to different extents.

2.2.2 Access Control Policy

One important part of security policy is access control. In general, the access control policy specifies who can access particular resources and what operations the accessor can perform on those resources.

A number of formal methods have been used to analyse access control policies and argue about their correctness and properties. The following paragraphs outline the most widely used formal models (Landwehr 1981):

- *Finite-state machine model.* This model uses a finite set of states, which represent the possible states of a system, and a transition function. The transition function determines the next state of the system based on a given input and the current state.
- *Lattice model.* A lattice is a finite set with a *partial ordering* on its elements such that for every pair of elements there is a least upper bound and a greatest lower bound. This model is used in military oriented systems because it closely matches the military classification structure.
- *Access Matrix.* It was introduced by Lampson (Lampson 1974; Lampson 1971) as a generalised description of operating system protection mechanisms.
- *Information-flow models.* These models recognise and exploit the lattice structure of security levels. Instead of requiring a list of axioms governing users' accesses, an information flow model simply requires that all information transfers obey the flow relation among the security classes.

In military security, the most widely used model is that of David Bell and Leonard LaPadula (BLP) (Bell and LaPadula 1974). Although the BLP model is widely used, it has limitations. For example, it has little relevance for systems in which users may change their own security levels or those of their files. Furthermore, BLP is inadequate for expressing requirements that certain operations cannot be performed by a single individual working alone. This has been identified as one crucial requirement for commercial systems (Clark and Wilson 1987).

2.2.3 Traditional Access Control Policies

Traditionally, there are two classes of access control mechanisms, namely *Mandatory Access Control* and *Discretionary Access Control*. The Trusted Computer System Evaluation Criteria (TCSEC) (NCSC 1985) defines Mandatory and Discretionary access control as follows:

Discretionary Access Control. *A means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control).*

Mandatory Access Control. *A means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorisation (i.e. clearance) of subjects to access information of such sensitivity.*

These two mechanisms, however, are inadequate to handle many access control requirements. (Olson and Abrams 1990) gives a generalised framework for access control and discusses a variety of policy choices that are not covered by the traditional Discretionary and Mandatory access control mechanisms. Also, in (Abrams 1993) the term *non-discretionary* access control refers to situations in which authority is granted to some users but there are restrictions on delegation and propagation of authority. If the propagation of authority is represented by an authority tree rooted in the security administrator, then:

- *Mandatory* is the case in which the tree has no branches
- *Discretionary* is the case in which the branches extend to every user
- *Non-Discretionary* is the case in which there are branches that do not extend to every user.

2.3 Access Control Enforcement

2.3.1 Reference Monitor

An important notion in access control enforcement is that of the *reference monitor* (Anderson 1972) which acts as a guard for each target object in the system and examines whether access requests for the target should be granted. All operation invocations on a target are intercepted by the reference monitor which allows the invocation only if the applied access control policy permits it.

The ISO Access Control Framework (ISO 1989) also defines the notion of a Reference Monitor. According to that definition, every operation request is intercepted by the *Access Control Enforcement Facility* (AEF) which requests an access control decision from the *Access Control Decision Facility* (ADF). If the ADF decides that the operation is permitted, the AEF forwards the operation to the target object, otherwise an exception is raised (Figure 2.2).

In general, the ADF bases its decisions on the current access control policy, on the properties of the target object and the user, as well as on the system context (for example time).

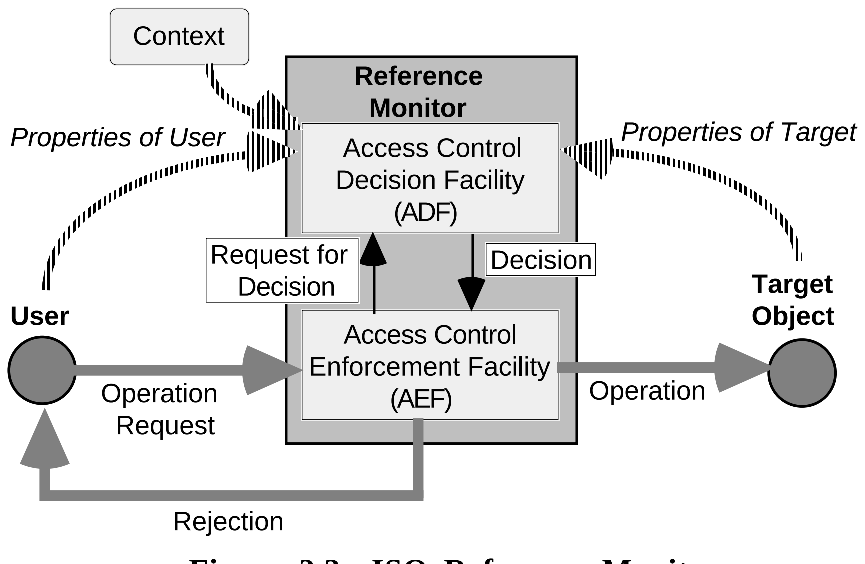


Figure 2.2 ISO Reference Monitor

The reference monitor that enforces a certain access control policy should be trusted by the authority that specifies that policy. In a distributed system it is possible that policies specified by different authorities govern the invocation of a single operation. In this case multiple reference monitors are required to grant permission for the operation. A typical example is illustrated in Figure 2.3 where a user in organisation *A* invokes an operation *OpA* on a target in organisation *B*. If we assume that *OpA* is likely to convey some information to organisation *B*, it is reasonable to assume that the operation has to be authorised by a reference monitor that is trusted to enforce the policy of organisation *A*. In addition, the operation may alter the state of the target and thus it has also to be authorised by a reference monitor that is trusted to enforce the policy of organisation *B*.

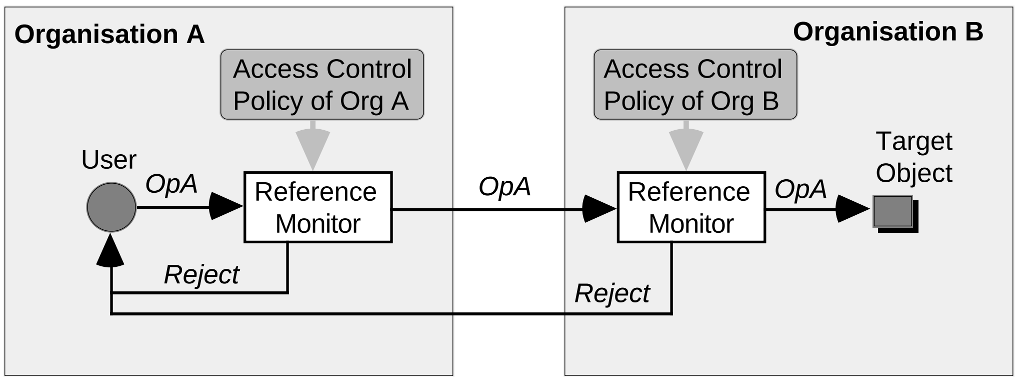


Figure 2.3 Example for Multiple Reference Monitors

2.3.2 Access Control Lists and Capabilities

The concept of the *access matrix* has been used as a reference model for access control since 1971 (Lampson 1974; Lampson 1971). Its practical value, however, is rather small as it exhibits a number of disadvantages if we try to use it in order to record and enforce access control policy in a real system, especially if that is a distributed system (Moffett 1990). An access matrix assumes global knowledge of the users and targets in the system. Furthermore, it assumes a single point that is globally trusted to maintain access control information. In a distributed system, these assumptions are not true.

There are, however, two ways of realising the access control information in a practical way:

- *Access Control Lists* (ACLs). For each target specify which users have access rights to it.
- *Capabilities*. For each user, specify which targets s/he can access with which access rights.

An ACL corresponds to a column of the matrix while a capability corresponds to a row. When a system uses capabilities the access control information is typically associated with the accessor (user or an agent acting on behalf of a user). On the other hand, when ACLs are used, the access control information is associated with the target. This is illustrated in Figures 2.4 (a) and (b) where we assume that the reference monitor is on the target site. In general, the reference monitor and the ACL are located in an area that can be sufficiently trusted to guard the target and is under the control of the authority that also controls the target object.

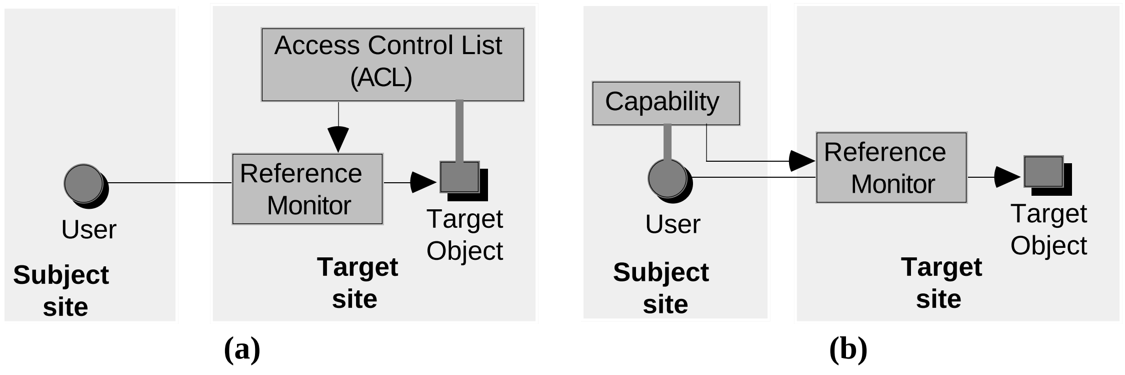


Figure 2.4(a) Use of Access Control Lists
Figure 2.4(b) Use of Capabilities

The capabilities used in a distributed system have to be protected using cryptocraphic techniques to prevent their unauthorised alteration and use. In order for a subject to invoke an operation on a target, it forwards the capability representing its access rights to the target. The reference monitor controlling access to the target verifies the validity of the presented capability as well as the identity of the subject. If both checks are successful, it identifies the operations authorised by the presented capability and makes an access control decision.

In an ACL system, the reference monitor has to check the ACL associated with the target in order to determine the operations the subject can perform and verify the identity of the user.

Both schemes remove the need for global knowledge. With regard to the ACL scheme we can further identify the following:

- System administrators can easily determine which subjects and with what access rights they can access a particular target.

- Revocation of access rights is easier in that the system administrator can modify the ACLs of the target objects.
- Security administrators, on the other hand, cannot directly determine the access rights a subject possesses as these are specified in the ACLs of multiple targets.
- Users are not typically aware of their access rights since these are specified by the ACLs of the target objects.

With regard to the capability scheme, we can identify the following points:

- In some cases, the access control decision making is faster compared to ACLs because the reference monitor has to deal with a smaller search space.
- Revocation of access rights represented by a capability that has been issued is problematic.
- The user is aware of the access rights it possesses as these rights are specified by the capabilities s/he holds.
- The system administrator can determine what access rights a user possesses.
- The system administrator can not easily determine which users can access a particular target with what access rights.

Actual implementations can adopt a hybrid approach; for example a mechanism where ACLs are used to determine the rights of a particular accessor and then these rights are used to create a capability for the accessor.

2.3.3 An Access Control Calculus

Typically, a user interacts with a target object via a number of other system entities which have to be authorised to act on behalf of the user. Furthermore, such an entity may propagate the access rights to other agents to enable them to act on behalf of the user. In addition, a user may restrict her access rights according to the context in which she is acting. This involves delegation of access rights and makes the problem of access control more complex in distributed systems.

In general, multiple entities may be involved in a single transaction. To deal with this problem, the notion of a *compound principal* is introduced in (Abadi et al. 1991; Abadi et al. 1993) within the framework of a calculus for access control. The authors propose an access control system which is based on the ACL paradigm but each ACL entry specifies the access rights for a compound principal. This scheme is also discussed in (Lampson et al. 1992) and is implemented in the Taos Operating System (Wobber et al. 1993; Wobber et al. 1994).

The starting point for our discussion is the notion of a *principal* which is an entity that makes *statements*. Principals are either *simple* (for example human user *Alice*) or *compound* (for example *AliceWS* ***for*** *Alice* meaning the workstation of Alice that makes a request on behalf of Alice using the access rights it has been delegated by her). A number of operators are used for defining compound principals and the notion of a principal has been extended to include:

- *Human users* and *machines.*
- *Channels,* such as input devices and cryptographic channels. Cryptographic channels are identified by cryptographic keys. We say that the key *K* ***says*** *s* where *s* is a statement contained in a message encrypted under the key *K*.
- *Conjunction of principals,* of the form $A \wedge B$. The interpretation of this is that if the compound principal $A \wedge B$ makes a statement *s*, then both *A* and *B* make the same statement *s*. It may be the case that $A \wedge B$ is trusted on *s*, but neither *A* nor *B* it trusted by itself and thus a *joint signature* is required.
- *Groups,* which are used to avoid listing explicitly all of the principals that are trusted in some respect. In this respect, groups provide an indirection mechanism.
- *Principals in roles,* of the form *A* ***as*** *R*. Roles are used as a means of diminishing the access rights of a principal. For instance, a machine *A* may adopt a weak role *R* before running a piece of untrusted software. In this way, the machine restricts its access rights and thus it confines the damage the untrusted piece of software may cause by accessing remote resources.
- *Principals on behalf of principals*, of the form *B* ***for*** *A*. The principal *A* delegates access rights to *B* which can then act on behalf of *A*, using the identity *B* ***for*** *A*. Expressions like *C* ***for*** *(B* ***for*** *A)* are used when *cascaded delegation* takes place.

The calculus uses a partial order $\Rightarrow$ (implication) among principals: $A \Rightarrow B$ stands for $A = A \wedge B$ and means that *A* is at least as powerful as *B*; we pronounce this "*A implies B*" or "*A speaks for B*". Intuitively, "*A speaks for B*" means that if *A* ***says*** the statement *s*, we believe that *B* ***says*** *s* as well. For example, if we know that a cryptographic channel *K* (a key) speaks for a principal *B*, this is $K \Rightarrow A$, we assume that everything we receive from that channel (encrypted under *K*) has been said by *A*.

Implication is often used to reflect group membership. For instance, if *A* is a member of the group *B* then $A \Rightarrow B$.

The calculus also uses the connective | to express the principal *B*|*A* which is obtained when *B* speaks on behalf of *A*, not necessarily with a proof that *A* has delegated authority to *B*; *A* could be mistaken or lying. By definition, *B*|*A* ***says*** *s* if *B* ***says*** that *A* ***says*** *s*. We pronounce *B*|*A* as "*B quoting A*.".

In many situations, a principal may wish to reduce his power. This is related to the enforcement of the *least privilege principle*. For instance, when a principal *A* wishes to delegate to a machine less trustworthy than his own, he should be able to limit the rights passed. This can be done by adopting a weaker role *R* and delegating with the identity *A* ***as*** *R*. In this case the role *R* represents a *set of access rights* which is a subset of the rights the principal *A* possess.

We can also think of a role *R* as a programme that the principal *A* is following; therefore the compound principal *A* ***as*** *R* should not be given more rights than the rights the programme *R* possesses. For example, let *A* be a machine and *R* a piece of software that *A* is running. The requests that *A* makes on behalf of *R* should not come out from *A* with full power. In fact those requests should emanate from *A* with the power that the program *R* has. Therefore, when *A* makes a request on behalf of *R*, it should state that it is actually the principal *A* ***as*** *R* that makes the request. Thus, when *A* makes a statement *s* in this role, we write *A* ***says*** that *R* ***says*** *s* which is *A*|*R* ***says*** *s*. So *A* ***as*** *R* is defined to be equal to *A*|*R*. The above-mentioned example and the fact that roles can be freely adopted justifies this definition.

In (Lampson et al. 1992) the following ACL entry syntax is given:

principal	::= forList | principal $\wedge$ forList
forList	::= asList | forList ***for*** asList
asList	::= properPrincipal | asList ***as*** role
role	::= pathName
properPrincipal	::= pathName | channel

We do not discuss the operator ***for*** and the delegation issue further in this chapter as these are discussed in detail in Chapter 4. We only discuss the use of groups and roles.

A simple example of an ACL entry is *"GroupA"* meaning that any principal in this group can access the associated target object. For an accessor *A* to be permitted access, proof that $A \Rightarrow GroupA$ (that is *A* is a member of *GroupA*) is required. (Abadi et al. 1991) identifies the fact that registrars for groups may be remote to the reference monitor, and thus group membership certificates are required. In the Taos OS (Wobber et al. 1993) an off-line CA signs membership certificates which are used in the ACL checking.

Another simple ACL entry is: *"GroupA* ***as*** *RoleA"* meaning that a member of the group *GroupA* that can play in *RoleA* is permitted to access the target object. For instance, the compound principal *A* ***as*** *RoleA'* is granted access, if there is evidence that $A \Rightarrow GroupA$ and $RoleA' \Rightarrow RoleA$, that is *RoleA'* is stronger than *RoleA*. A certificate stating that $RoleA' \Rightarrow RoleA$ is required.

Note that a principal *A* can freely adopt any role *R*, but this does not make sense if there is no ACL entry permitting this role adoption, either directly (for example *A* ***as*** *R*) or via group membership (for example *G **as** R* where the principal is a member of *G*). This suggests that a system administrator that has to assign a principal (e.g. *John*) to a role (e.g. *Director*), has two alternatives:

- Modifies all the relevant ACLs by adding *John **as** Director*
- Associates a group $G_{director}$ with the role *Director* and subsequently makes *John* a member of $G_{director}$.

Note that a role actually represents a set of access rights and a principal adopts a certain role to declare that she wants to restrict its access rights. Roles, however, have to be defined when ACLs are determined and if there is any need to redefine them all the relevant ACLs have to be modified.

2.4 Basic Cryptographic Tools

As it is difficult to provide physically secure communication links, there is a need to use some form of cryptographic protection to provide communication security for a distributed system. This section outlines a number of cryptographic tools that can be used to provide message secrecy and integrity. It also discusses possible attacks that can be mounted by an intruder and identifies a number of design principles that should be taken into account when a cryptosystem is used to provide secure comminication in a distributed system.

2.4.1 An Example

We base our discussion on a simple example illustrated in Figure 2.5. We assume that there is a message flow from Alice to Bob and vice versa, using trusted cryptographic facilities. Because Alice and Bob cannot perform cryptographic operations, they can only communicate with their cryptographic facilities in *plaintext*, that is in some ordinary language that everybody can understand. We also assume that these two people can communicate with their own cryptographic facilities in a physically secure environment, that is nobody that they do not trust can have access to the messages transferred in plaintext. Each of them can also maintain a set of secrets which are well protected inside their areas of trust.

Messages, however, can only be exchanged between Alice’s and Bob’s areas of trust via an insecure communication environment. That is, an opponent has access to the communication link between Alice and Bob and can mount any of the attacks mentioned in Section 2.1.2. The opponent also possesses cryptographic facilities but he *cannot* have access to Alice's and Bob's secrets. On the other hand, we assume that the opponent knows what kind of cryptographic systems Alice and Bob use to protect their messages[1].

The following sections outline a number of cryptographic tools that Alice and Bob can use in order to achieve message security.

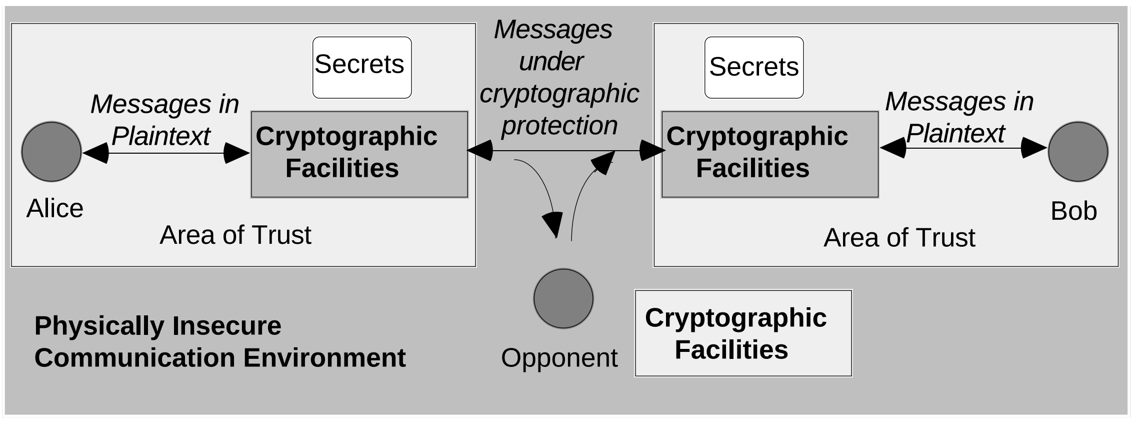


Figure 2.5 An Example for Cryptographically Protected Communication

2.4.2 Symmetric Cryptography

Let us assume that both Alice and Bob can use a pair of functions named *encrypt* and *decrypt*. These functions are implemented by their cryptographic facilities and take as input a message and a value which we call *cryptographic key*, and produce a *ciphertext* (encrypted message) as output. For example *m' = encrypt(m, K).* We also assume that for any message *m* and a suitable key *K*:

- *decrypt(encrypt(m, K), K) = m.* (A1)
- If somebody only knows *encrypt(m, K)* s/he cannot compute *K* or *m*. (A2)
- If somebody only knows *m*, s/he cannot produce a message *m'* such that *decrypt(m', K) = m* (A3)

[1] In some cases this assumption is not valid. Specifically, there are cryptosystems the details of which have not been published. A typical example is the *skipjack* algorithm which has been proposed by the US government and has been implemented in the *Clipper* chip. However, it is common to assume that the cryptographic algorithm used by the communicating parties is known to the opponent and hence the only information that has to be kept secret is the cryptographic key.

"Cannot compute" here means that it is too hard to be feasible in practice. That is, it requires facilities that are extremely costly or impossible to implement with the current technology, or otherwise computation takes an extremely long time or exceeds the life span of the information that has to be protected. We need one more major assumption in order to show that Alice and Bob can use these functions to exchange *secret* messages, that is, messages that cannot be seen by a person that does not have access to Alice's and Bob's areas of trust:

- We assume that Alice and Bob know the same secret key *Kab* and nobody else knows that key. (A4)

At this point we do not concern ourselves with possible techniques these two people can utilise in order to agree on that key; this is discussed in Section 2.5.

If Alice wants to send a secret message *m = "Hello Bob"* to Bob, she uses the *encrypt* function to produce the message *m' = encrypt("Hello Bob", Kab)*. We say that *m'* is the *ciphertext* of *m* which has been encrypted under the key *Kab*. This message is sent over the insecure communication environment to Bob. If Bob can somehow see that this message comes form Alice, he uses the *decrypt* function and the key *Kab* to get *"Hello Bob" = m = decrypt(m', Kab)* (according to A1 and A4). An opponent can get the message *m'* but according to (A2) and (A4) he cannot find out *m* as he does not know the key *Kab*. Thus, the *secrecy* of *m* is preserved as Bob is the only one outside the trust area of Alice that can possibly see it.

In this case, Alice and Bob use the same key for encrypting and decrypting a message. A cryptographic system (cryptosystem) that requires the same key for both encryption and decryption is referred to as *symmetric*, *single key*, *secret key* or *conventional* cryptosystem.

2.4.3 Symmetric Cryptosystems

The most widely used symmetric cryptosystem is the Data Encryption Standard (DES) which was adopted in 1977 by the American National Bureau of Standard (now National Institute of Standards and Technology – NIST) as an encryption standard. It was developed by IBM with technical advice from the National Security Agency (NSA). A detailed description of the DES history is given in (Schneier 1994).

DES uses a 56-bit key and a series of steps to transform a 64-bit input (plaintext) into a 64-bit output (ciphertext). The same key and step series are used to reverse the ciphertext back to the original plaintext. DES is basically a *block cipher* in that it processes one block of 64-bits at a time. It can, however be used as a *stream cipher* when this is required, that is to process input data in real time without padding transmitted messages. In practice, the DES algorithm is used in one of the four modes of operation described in (NBS 1980). These are summarised in Table 2.1.

The nature of DES enables very fast implementations of the algorithm in hardware. For instance, a DES chip developed by DEC (Eberle 1992) can encrypt/decrypt approximately 15.6 million 64-bit blocks per second. This gives an encryption/decryption rate of 1GBit per second. Software implementations are much slower. For example, an IBM 3090 mainframe can perform 32,000 encryptions per second (Schneier 1994).

Electronic Codebook (ECB)	Each 64-bit block of plaintext is processed independently.
Cipher Block Chaining (CBC)	The XOR of the next 64-bit plaintext block and the preceding 64-bit ciphertext block are given as input to the encryption algorithm. It requires an Initialisation Vector (IV).
Cipher Feedback (CFB)	Encrypts *n* bits at a time by XORing the plaintext with the result of the encryption of the preceding ciphertext. It is used for stream oriented data transmission and requires an Initialisation Vector (IV).
Output Feedback (OFB)	Similar to CFB but bit errors do not propagate due to a different feedback loop.

Table 2.1 DES Modes of Operation

Because DES uses a 56-bit key, it is subject to a brute-force attack with an effort of the order of 2^{56}. DES, however, is a *group* which means that a plaintext encrypted twice with two different keys produces a ciphertext that cannot be produced by a single encryption using a third 56-bit key. That is, there is no key *K3* such that *encrypt(m, K3) = encrypt(encrypt(m, K1), K2)* if *K1* is different to *K2*. This indicates that we can increase the resistance of DES to brute-force attacks by using two different keys (double DES). The use of two different keys, however, is vulnerable to a known plaintext attack which is referred to as the *meet-in-the-middle attack* (Diffie and Hellman 1977) The cost of this attack is of the order of 2^{56}.

The meet-in-the-middle attack can be tackled by encrypting a message three times using three different keys. This increases the required effort of an known plaintext attack to 2^{112}, but it also increases the effective key length to 168 bits. Tuchman has proposed an alternative triple encryption method that uses two keys (Tuchman 1979). This also requires an attack effort of the order of 2^{112}. This method is known as *Triple DES*. Triple DES is a popular alternative to DES when increased security is required.

A number of symmetric cryptographic algorithms have been proposed as alternatives to DES and its variants. Among them one that receives a lot of attention is the International Data Encryption Algorithm (IDEA) (Lai and Massey 1992). IDEA, like DES, is a block cipher and encrypts/decrypts a 64-bit block at a time. It, however, uses a 128-bit key which increases the cost of a brute-force attack up to 2^{128}. In terms of speed, IDEA is about as fast as DES.

Two other interesting algorithms are RC2 and RC4 which have been designed by Ron Rivest for RSA Data Security, Inc. An important characteristic of these algorithms is that they are *variable-key-size* ciphers while their speed is independent of the key size. RC2 is a block cipher that operates on blocks of 64 bits and RC4 is a stream cipher. These algorithms have not been published though they are used in some commercial products.

2.4.4 Asymmetric Cryptography

We now return to the scenario described in Section 2.4.1 and assume that Bob has constructed a pair of cryptographic keys K and K^{-1} and both Alice and Bob can use two functions named *encrypt* and *decrypt* with the following properties:

- $m = decrypt(encrypt(m, K), K^{-1})$ (A5)
- $m = decrypt(encrypt(m, K^{-1}), K)$ (A6)
- If somebody knows *encrypt(m, K)* and K s/he cannot compute K^{-1} or m. (A7)
- If somebody knows m and *encrypt(m, K^{-1})*, s/he cannot compute K^{-1} (A8)

If Bob makes the key K publicly known and keeps K^{-1} secret we observe that Alice can encrypt a message m with K. This message can only be read by Bob because he is the only one that knows K^{-1} (according to A5 and A7). Hence the secrecy of the message m is preserved. Note however that anybody that knows K can encrypt messages that can only be read by Bob.

On the other hand, Bob can encrypt messages with K^{-1} which can be read by anybody that knows K. This use of the cryptosystem does not provide message secrecy as we have assumed that Bob has published K, but it can be used to verify message integrity.

Two different keys are needed for encrypting and decrypting a message (c.f. symmetric cryptography Section 2.4.2). If one of these keys has been published while the other has been kept secret, everybody can encrypt a message with the published key, but only the person knowing the corresponding secret key can decrypt that message. In addition, the person knowing the secret key can encrypt messages that can be decrypted by anybody knowing the corresponding published key. Such a kind of cryptosystem is usually referred to as a *asymmetric cryptosystem* because the encryption and decryption require different keys. Sometimes, the terms *two-key* and *public-key* cryptosystem are used. In these systems, the key that has been published is known as the *public key* and the one that is kept secret is referred to as the *private key*.

2.4.5 Asymmetric Cryptosystems

The most widely used asymmetric cryptosystem is RSA (Rivest et al. 1978). It is based on the assumption that factoring of large numbers is difficult in practice. RSA can be used for both message authentication and encryption.

In general, RSA is about 100 times slower in software and 1,000 to 10,000 times slower in hardware than DES (Schneier 1994; Shand and Vuillemin 1993). Furthermore, a small public exponent is usually chosen in order to make the encryption of messages and the verification of digital signatures faster than the decryption and signing respectively.

The generation of the RSA keys is also computationally intensive and it is usually done in advance of any actual need to use a new key pair. Because RSA is relatively expensive, it is typically employed to establish shared keys (session keys) which can be used to exchange messages using a symmetric cryptosystem, for example DES.

Diffie and Hellman proposed the first public-key algorithm in 1976 (Diffie and Hellman 1976). This algorithm can be used by two parties to establish a shared key which can then be used for subsequent encryption of messages. The algorithm is based on the assumption that it is difficult to compute discrete logarithms. Note, however, that the algorithm does not provide authentication of the communicating parties. It can be combined, however, with an authentication technique that enables the two communicating parties to verify each other’s identity. For instance, such a method is proposed in (Krawczyk 1996).

It is worth noting that because the secrecy of the established shared key does not depend on other long-lived keys, the Diffie-Hellman key exchange provides *perfect forward secrecy* (Krawczyk 1996). A key exchange mechanism is said to provide perfect forward secrecy when it protects short-lived keys from compromise even in the case of the exposure of long-lived keys.

The American National Institute of Standards and Technology (NIST) has published a Digital Signature Standard (DSS) which employs the Secure Hash Algorithm (SHA) and an asymmetric algorithm known as Digital Signature Algorithm (DSA) (NIST 1992a; NIST 1993a). The security of the DSA is based on the difficulty of calculating discrete logarithms. This algorithm is for authentication only. Because the only computationally demanding task does not involve the message to be signed, a message can be signed quickly by using pre-computed values, and thus message signing is faster than verification.

2.4.6 Hash Functions

One-way hash functions are used to create a "fingerprint" or *digest* of a message that is unique. One-way hash functions are sometimes referred to as *digest functions*. A hash function *H* operates on a message *m* of arbitrary-length and returns a fixed-length hash value *h*, i.e. $h = H(m)$. An one-way hash function *H* has the following properties (Stallings 1995):

- It can be applied to messages of arbitrary length.
- It produces a fixed-length hash value (output).
- It is easy to calculate a hash value of any given message.
- For any given hash value *h*, it is computationally hard to find a message *m* such that $H(m) = h$.
- For any given message *m*, it is computationally hard to find another message $m' \neq m$ such that $H(m) = H(m')$.
- It is computationally hard to find any pair of messages (m, m') such that $H(m) = H(m')$.

The most widely used hash functions are the MD4 (Rivest 1990), MD5 (Rivest 1992) and the Secure Hash Algorithm (SHA). The MD4 and MD5 have been designed by Ron Rivest and the latter is an improved version of MD4. Both algorithms produce a 128-bit digest of the message given as input. The SHA has been proposed by the NIST for the Secure Hash Standard (SHS) (NIST 1992b; NIST 1993b). and produces a 160-bit message digest.

2.4.7 Message Authentication Code (MAC)

A hash function can be used to generate a Message Authentication Code (MAC) (sometimes called Message Integrity Code – MIC). This is illustrated in Figure 2.6 where Alice and Bob share a secret value *K*. Because the hash function used by Alice and Bob is known, Alice has to concatenate a message *m* with the secret value *K* in order to allow Bob to ensure that *m* has been sent by Alice. Making this secure, requires some care depending on the characteristics of the hash function used (Kaufman et al. 1995).

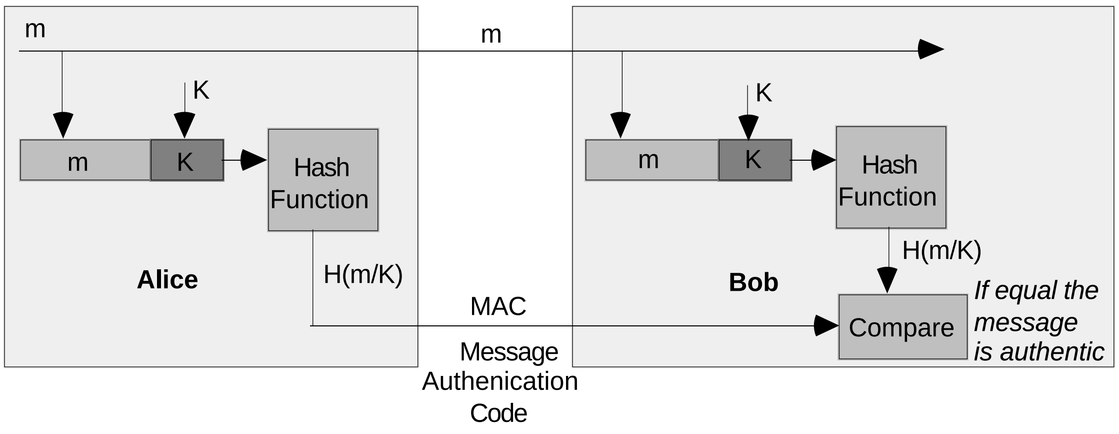


Figure 2.6 Use of a Message Authentication Code (MAC)

A MAC can also be generated using CBC DES. The use of one-way hash functions to produce MACs, however, is more efficient for most applications as the widely used hash functions (such as MD4 and MD5) are faster than DES and they were designed to be implemented efficiently in software while DES was designed to be efficient when done in hardware.

2.4.8 Ensuring Message Secrecy and Integrity

Section 2.4.2 discussed the provision of message secrecy by using symmetric cryptography. But in most practical applications integrity is required in addition to secrecy. Following the discussion in Section 2.4.2, when Bob receives the encrypted message *"Hello Bob"*, he can be sure that this message has been sent by Alice because it is highly unlikely that an opponent can choose a ciphertext which produces (after decryption) a sensible cleartext in English. Note that according to (A3) the opponent cannot derive a ciphertext *m'* that produces a given message *m*, that is *m = decrypt(m', K)*.

It is not always the case, however, that the receiver can easily determine that an incoming ciphertext produces a legitimate plaintext. If, for example, Bob expects a serial number of which the format is unknown to him, he cannot determine whether the ciphertext *m'* has been sent by Alice. Note that the function *decrypt* can accept any ciphertext as input. Further, in most applications, the integrity check should be done automatically and thus we need a mechanism that enables the receiver to verify the integrity of any incoming encrypted message.

One way to achieve secrecy and integrity together is to encrypt the message along with a suitable checksum. This method, however, may exhibit some vulnerabilities (Voydock and Kent 1983) and should be used with care. Figure 2.7 shows how integrity can be achieved by using a hash function. Kerberos V5 uses three different methods named "DES-CBC-CRC", "DES-CBC-MD4" and "DES-CBC-MD5". These methods employ the CRC-32 checksum, the RD4 and RD5 hash functions respectively (Kohl and Neuman 1993).

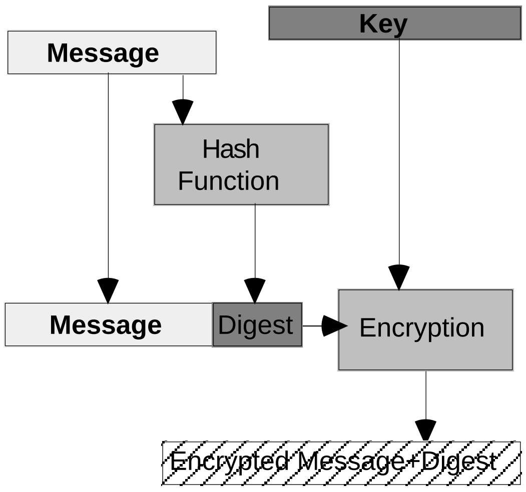


Figure 2.7 Encryption of a Message Along With its Digest

2.4.9 Cryptanalysis

With reference to the example in Section 2.4.1, an opponent may attempt to find out the key Alice and Bob use to encrypt messages or simply discover a message that has been sent encrypted under a key shared between Alice and Bob. The term *cryptanalysis* refers to the process of discovering either a key or messages encrypted under that key. Usually, we assume that the cryptographic algorithms used by the communicating parties are known to the cryptanalyst, though this is not always the case in real applications. A classification of attacks on encrypted messages according to the amount of information known to the analyst follows:

- *Ciphertext only*. In this case the analyst knows the encryption algorithm and the ciphertext of a number of messages. The analyst has no knowledge of the plaintext of these messages
- *Known plaintext*. The analyst also knows a number of messages in plaintext along with the corresponding ciphertext.
- *Chosen plaintext*. In this case the analyst can determine the ciphertext corresponding to plaintext s/he has chosen.

- *Chosen ciphertext*. The analyst can determine the plaintext corresponding to ciphertext s/he has chosen.

A significant cryptanalytic attack is the *brute-force attack*. In the simple view, brute-force attack is the process whereby the cryptanalyst decrypts (or encrypts if this is easier) a message using every possible key until there is a match between the ciphertext and the corresponding plaintext. A cryptosystem, therefore, should have such a large keyspace that this attack is difficult in practice. In some cases keys are chosen from a small keyspace (for instance, they are derived from passwords generated by humans). This results in systems that are susceptible to brute-force attacks. This is particularly true if the opponent can perform a brute-force attack without interacting with a system that can detect the attack. A typical example of a system exhibiting this weakness is Kerberos (Bellovin and Merritt 1991). This issue is discussed in more detail in (Lomas et al. 1989).

Note that the analyst needs to have some knowledge about the plaintext otherwise he would not be able to identify the correct key. Furthermore, the recognition of a legitimate plaintext has to be done automatically. Thus, *a well-designed system should take measures to prevent an opponent from correlating ciphertext with the corresponding plaintext.*

The decision whether a keyspace is large enough depends on the computing power that is available to the opponent. As an example, DES uses a 56-bit key and thus there 2^{56} possible keys that an analyst has to search. If we assume that a chip is capable of checking 265,000,000 keys per second and we have 4,000 of these chips, the whole keyspace is searched in 24 hours (Schneier 1994). Further, the analyst has a 1/24 change to discover the key in one hour. Thus, if the opponent can use such a chip array to mount an attack, a 56-bit key is unacceptable. On the other hand, if triple DES is used (Section 2.4.4), there are 2^{112} possible keys and thus an attack using the same hardware requires approximately $2*10^{14}$ years. Moreover, the probability that the analyst discovers the key in one year is $5*10^{-15}$.

It is important that *a key should be used for a limited period of time* in order to reduce the change that the opponent will discover the key while it is still being used.

Another attack that is applicable to DES and other similar symmetric cryptosystems is *differential cryptanalysis* (Biham and Shamir 1990). It can be shown that such an attack requires 2^{47} *chosen* plaintexts to succeed against DES. Since it is not practical to have this large number of chosen plaintexts, the attack is not considered as successful against DES. It seems that DES was designed to be invulnerable to differential cryptanalysis. Linear cryptanalysis, which is a recent development, requires 2^{47} known plaintexts rather than chosen known plaintexts (Stallings 1995).

In general, *a system should not allow intruders to mount chosen-plaintext attacks*. For example, protocols using the "challenge-response" method should be used with care since it may allow intruders to produce ciphertext of chosen plaintext. In addition, keys that are valid for a long period should not be used for encrypting a large amount of plaintext or plaintext that can be known to an intruder.

2.5 Authentication

In a computer system entities interact by exchanging information. There is a need to verify the identity of the source of a piece of information. The term *authentication* refers to the verification of the identity of an entity. Authentication can easily be achieved in a centralised system where there is a single TCB which is aware of all the entities in the system as well as the users that interact with the system via the TCB.

In distributed systems, however, authentication is a difficult problem and in fact is one of the most crucial issues in distributed systems security. This is because entities interact through untrusted communication links and there is no single TCB that can be trusted globally. This section discusses the authentication problem in distributed systems and outlines a number of approaches that are currently in use.

2.5.1 Basic Aspects

As in Section 2.3.3, we use the term *principal* to refer to an entity that makes statements (Lampson et al. 1992) Examples of principals include human users, processes, communication ports and host machines. A principal that claims a certain identity is referred to as the *claimant* The entity that verifies a claimant's identity is called the *verifier*.

Because principals interact by exchanging information, authentication is typically based on secrets that are associated with the identities of the principals. A typical example is a password which is associated with a user. Password-based authentication is common in centralised systems but it is unsuitable for distributed systems. The user cannot verify the identity of a remote machine which has to be given a password. This implies that a password can be revealed to an untrusted machine. Moreover, passwords that are sent across a physically unprotected network can be stolen by eavesdroppers who can subsequently impersonate the user (Neuman and Ts'o 1994).

Further, in distributed systems, other types of principals in addition to human users have to be authenticated. A typical example is an automated agent which has to invoke management-related operations on a remote machine. Typically, in these scenarios mutual authentication is required rather than one-way authentication which is achieved using passwords.

Two principals, for instance Alice and Bob, that share a key can use a cryptosystem to check the integrity of the messages they exchange. If Bob believes that the key *Kab* is known only to Alice and himself, he believes that a statement signed with that key has been uttered by Alice, and vice versa. So, if Alice and Bob see each other in the flesh and agree on a shared key, they can subsequently verify the source of the messages they exchange using that key. We assume that when these two people meet each other, they communicate in a physically secure communication environment and thus nobody else can find out that shared key. We also assume and they can easily verify the identity of each other.

In practice, however, it is rarely the case that any two principals wishing to communicate can agree on a shared key by communicating in a physically protected communication environment as, in most cases, physically protected communication implies off-line communication. Thus, there is a need to provide the means for two principals to agree on a shared key over a physically unprotected communication link.

Furthermore, if we consider asymmetric cryptography, the source of the messages can be verified by decrypting them with the sender's public key. Even in this case, the receiver (and verifier) needs a trustworthy means to ensure that he uses the public key that corresponds to the private key of the claimant. Again, there is no straightforward solution to this problem if the claimant cannot give his public key to the verifier in a way that preserves integrity.

In 1978, (Needham and Schroeder 1978) proposed a number of authentication protocols aiming to provide authentication in a physically unprotected communication environment. Since that paper, a variety of authentication protocols have appeared in the literature. Many of these establish shared keys that the communicating principals can subsequently use to verify the source of the exchanged messages. These protocols are based on the use of an authority that is trusted by the communicating principals.

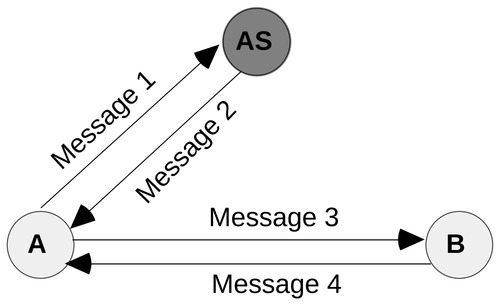
An authentication protocol which is based on symmetric cryptography typically involves a trusted *authentication server* which shares a key with each principal. The authentication server is trusted to keep principals' keys secret. A protocol based on asymmetric cryptography involves a *certification authority* (CA) which certifies the public keys of the principals. There is no need for the public keys of the principals to remain secret but the mapping from the identities to the public keys should be protected against unauthorised alteration.

Note that there is a need for a principal to be authenticated when the authentication server or the CA records its secret key or its public key respectively. Typically, this initial authentication is done off-line and involves a trusted security administrator.

A major issue with regard to authentication protocols is the verification of the *freshness* of the information upon which an authentication decision is based. Authentication protocols can be divided into two categories according to the mechanism they employ in order to ensure message freshness. The first category comprises protocols that use *timestamps*. Timestamps require synchronised clocks which should be used with care (see for example (Gong 1992)). The second category comprises protocols that use *nonces*. A nonce is information that is used only once. Usually, nonce-based protocols require more communication steps.

2.5.2 Authentication Based on Symmetric Cryptography

Authentication protocols that are based on symmetric cryptography involve an AS which is trusted to generate a good key to be used by the two authenticated principals. In order to illustrate the rationale behind the operation of an authentication protocol, this section presents a typical authentication protocol which uses timestamps (Figure 2.8). In fact, it is a simplified version of the protocol employed by Kerberos V4 (Miller et al. 1987) and is based on the Needham-Schroeder protocol (Needham and Schroeder 1978) which, nevertheless, uses nonces.



Message 1: A -> AS:	$A,\ B$
Message 2: AS -> A:	$\{TS1,\ Kab,\ B,\ \{TS1,\ Kab,\ A\}_{Kb}\}_{Ka}$
Message 3: A -> B:	$\{TS1,\ Kab,\ A\}_{Kb}\ ,\ \{A,\ TS2\}_{Kab}$
Message 4: B -> A:	$\{TS2 + 1\}_{Kab}$

Figure 2.8 Kerberos V4 Authentication Protocol (simplified)

Initially we assume that the authentication server (AS) shares a key *Ka* with principal *A*. *A* believes that only itself and AS know this key. Thus, if a cryptosystem that provides both secrecy and integrity is used, a statement encrypted under *Ka*, has been uttered either by AS or by *A* and nobody else can read it. Note, however, that this alone does not guarantee that a message is fresh, i.e. is not a replay by an attacker who does not know *Ka*, but he can record messages and insert them into the communication link later. Similarly, AS shares a key *Kb* with *B* and hence a similar assumption applies for the messages encrypted under *Kb*.

Principal *A* sends message 1 to AS requesting the establishment of a shared key (session key) between *A* and *B*. AS generates such a key *Kab* and sends message 2 back to A. The content of message 2 is encrypted under *Ka* and thus only *A* can read it. This contains a timestamp *TS1*which enables *A* to ensure that this message is fresh, i.e. is not a replay of an old message. Message 2 also contains the key *Kab* and the identity of *B*. The interpretation of this is that *Kab* has been chosen by AS as a key to be used for secure communication between the receiver of the message (*A*) and *B*. The same message also contains $\{TS1, Kab, A\}_{Kb}$ which cannot be read or modified by *A* as it is encrypted under the key of *B*. This token contains the key *Kab* which has to be given to *B*. So, this token is given to *B* in message 3 along with $\{A, TS2\}_{Kab}$ which has been constructed by *A*. *B* can now retrieve *Kab* and it believes (after checking *TS1*) that this key has been chosen by AS for communication between *A* and *B*.

At this point, *A* and *B* believe that these two principals are the only ones (apart from AS) that *possibly* know *Kab*. *B* uses *Kab* to decrypt $\{A, TS2\}_{Kab}$. If the decryption is successful and the *TS2* is fresh, *B* believes that this encrypted token has been sent by *A* and thus *A* knows *Kab*. Subsequently, in message 4, *B* replies by sending $\{TS2 + 1\}_{Kab}$. This enables *A* to ensure that *B* also knows *Kab*. At this point mutual authentication has been achieved and the two principals can be sure of the source of messages encrypted under the session key *Kab*. The last communication step is optional in Kerberos and it is only used when mutual authentication is required.

In the original Needham-Schroeder protocol, on which the presented protocol is based, an intruder that has found an old session key can reuse it as *B* does not check the freshness of the token containing the session key in message 3 (Burrows et al. 1990). That protocol is rectified in (Needham and Schroeder 1987). The Kerberos protocol, however, uses timestamps and avoids this weakness on the assumption that the principals' clocks are loosely synchronised.

An improved version of this protocol is used in Kerberos V5 (Kohl and Neuman 1993; Neuman and Ts'o 1994). Specifically, in message 2 the token $\{TS1, Kab, A\}_{Kb}$ is not encrypted with *Ka* and the message 4 becomes $\{TS2\ \}_{Kab}$. In fact there is no reason for the timestamp *TS2* to be incremented as *A* can recognise this message as fresh anyway (Burrows et al. 1990). Note that$\{TS2\ \}_{Kab}$ cannot be a reflected message since *A* has never emitted $\{TS2\ \}_{Kab}$.

2.5.3 Authentication Based on Asymmetric Cryptography

Public-key authentication is based on the principle (Woo and Lam 1992): "If a principal can correctly sign a message using the private key of the claimed identity, this act constitutes sufficient proof of identity". If we assume that each principal possesses a private/public key pair, authentication can be achieved by employing a CA which is trusted to certify the binding between public keys and principals' identities. The CA certifies such a binding by signing a certificate using its private key. We assume that all principals know the corresponding public key of the CA and thus they can verify the integrity of a certificate signed by the CA.

It is worth noting that the CA can sign certificates for all principals in advance of any actual need for using them and consequently publish these certificates in a publicly accessible database (which may be replicated to increase the availability). Hence, the CA can operate off-line which makes it easier to increase its physical protection. The on-line certificate database, however, has to be trusted to timely remove revoked certificates (Lampson et al. 1992).

Authentication that is based on asymmetric cryptography is better suited for store and forward applications such as electronic mail or for applications where a signature has to be verified by multiple receivers (Neuman and Ts'o 1994). On the other hand, asymmetric cryptographic operations are expensive and hence public-key authentication may give rise to efficiency problems when a single server has to perform many authentication operations (i.e. to verify a large number of certificates).

More details about the use of asymmetric cryptography for authentication can be found in (Abadi et al. 1990; Gasser and McDermott 1990; Lampson et al. 1992; Liebl 1993; Linn 1990; Needham and Schroeder 1978; Tardo and Alagappan 1991; Wobber et al. 1993; Woo and Lam 1992; Wray 1994).

2.5.4 Relays

It is possible to simulate the effect of asymmetric cryptography with regard to message integrity by using symmetric cryptography and employing trusted machines that act as *relays*. This is proposed in (Davis and Swick 1990) and it is further discussed in (Lampson et al. 1992). The relay decrypts a message encrypted under the key of the sender and re-encrypts it with the key of the receiver. The receiver can read the "translated" message and verify its integrity on the assumption that the relay is sufficiently trustworthy.

In order to make this scheme scalable, (Davis and Swick 1990) proposes the use of replicated relays that disperse their security database (that is the keys of the registered principals) by issuing *Private-Key Certificates*. A private-key certificate (PKC) contains the key and the name of a registered principal and it is encrypted under the *master key* of the relay to preserve secrecy (using symmetric cryptography). In this way, a relay maintains no state other than its master key, which facilitates the replication of the relays. The PKCs of both the sender and receiver have to be presented to a relay in order for a message translation to take place. Relays can also be used for two-way authentication which results in the establishment of a shared key between the two authenticated principals.

2.5.5 Multiple Authentication Servers and CAs

In the previous sections we assumed that the two principals running an authentication protocol are known to the same authentication server or CA. It is not reasonable, however, to assume there is just one authentication server or CA in a large distributed system that possibly crosses organisational boundaries. A distributed system often consists of multiple subsystems administered by different authorities. These authorities typically represent different interests and hence it is difficult to imagine a single authentication authority that is globally trusted.

We use the term *realm*, as in the Kerberos system (Neuman and Ts'o 1994), to refer to a set of principals registered with a particular authentication server or CA. The term inter-realm authentication refers to authentication accomplished between two principals belonging to different realms. In order to achieve inter-realm authentication, trust between the two involved realms has to be established so that a realm can accept a certificate issued by the remote realm regarding a principal in that realm. Further, there is a need to establish a secure communication channel between these two realms. If symmetric cryptography is used, this can be achieved by establishing a shared key between the authentication servers in the two realms. If asymmetric cryptography is used, the two involved CAs have to exchange their public keys.

Note that if authentication among n different realms has to be achieved, each realm has to exchange keys with $(n-1)$ remote realms and $O(n^2)$ keys have to be exchanged amongst all realms. As this is not scalable, a *hierarchy* of authentication servers or CAs can be constructed. In such an arrangement, each realm shares keys with its children and parent realms and an authentication between two principals A and B involves the authentication servers or CAs on the path from A's realm, up to the least common ancestor of A's and B's realms and down to B's realm. This is usually called a *certification path*. *Cross-links* between realms can be added so that shorter certification paths can be used and perhaps bypass CAs or authentication servers that are not sufficiently trusted.

Naming is another important issue with regard to inter-realm authentication since all principals should be uniquely identifiable across realms. For a further discussion on inter-realm authentication see (Birrell et al. 1986; Lampson et al. 1992; Tardo and Alagappan 1991).

2.5.6 Other Authentication Issues

In general, there is a trade-off between the availability and security of an authentication service. The performance of a distributed system that makes use of an authentication service (provided either by authentication servers or CAs) partially depends on the availability of that service. It is therefore desirable to increase the availability of the authentication service. Typically this is achieved by employing replicated servers, but this may affect the security of the system in that it increases the probability that a server is being compromised. When asymmetric cryptography is used, this problem can be alleviated by employing a replicated database holding certificates signed by the CA. In this way there is no reason for the CA to be highly available. On the other hand, if symmetric cryptography is used, the authentication servers should be on-line. (Gong 1993) proposes an approach whereby replicated authentication servers are used but a minority of compromised servers cannot affect the security of the system.

Authentication protocols are prone to design errors. According to (Gollman 1996) this is partly due the lack of a proper translation method of the high-level protocol objectives to a low-level, implementable protocol. A number of papers trying to identify design principles have also appeared recently in the literature (Abadi and Needham 1994; Simmons 1994; Woo and Lam 1994). Others propose the use of formal methods for reasoning about the correctness of the authentication protocols. (Meadows 1992) defines four approaches that have been adopted for the analysis of authentication protocols. The most common approach is the one that uses logic developed for the analysis of knowledge and belief. A typical example of such an approach is the BAN logic (Burrows et al. 1990) which is extended in (Gong et al. 1990). A survey of formal methods for the analysis of authentication protocols is given in (Rubin 1993).

2.6 Chapter Summary

This chapter has outlined basic security concepts with regard to distributed systems. It has also introduced basic terminology that is used in the following chapters of this thesis.

Although security is a well understood aspect in centralised systems, distributed systems pose problems mainly due to their size, distributed management and communication over physically unprotected networks. Access control has to be specified by multiple authorities and the access control mechanism has to take into account the fact that a single request may involve multiple principals that are possibly trusted to different extents. The operation of the reference monitor has to be supported by an authentication mechanism that verifies the identities of remote principals. Traditionally, access control enforcement systems use either capabilities or Access Control Lists (ACLs). Both schemes exhibit advantages and disadvantages and in practice a combination of these two is often used.

We have outlined an access control scheme which is based on the notion of a compound principal and the use of ACLs. The description of this system is supported by an access control calculus which is further used in Chapter 4 to analyse the delegation issue.

A number of cryptographic tools can be employed to provide communication security. These are based either on asymmetric or symmetric cryptography. In general, asymmetric cryptography (for example RSA) is suitable for distributing keys or establishing shared keys. Also, the Diffie-Hellman key exchange can be used when perfect forward secrecy is required. Symmetric cryptography is better suited for encryption of data traffic since it is approximately 100 times faster than asymmetric cryptography when both are implemented in software and 1,000 to 10,000 times faster when both are implemented in hardware. Further, digest functions such as MD4, MD5 and SHA are often used to ensure data integrity. Cryptographic tools, however, should be used prudently as they may seriously affect the performance of the system (particularly those based on asymmetric cryptography).

Provision of authentication is achieved by employing trusted authentication servers or Certification Authorities (CAs). CAs are based on asymmetric cryptosystems and are better suited for store and forward applications or for applications that require signature verification by multiple receivers. Authentication servers which are based on symmetric cryptosystems are better suited for applications where a single server has to perform many authentication operations. Trusted servers that can “translate” messages encrypted under a shared key can simulate the effect of asymmetric cryptography. These servers are referred to as relays. Relays can disperse their security database using Private-Key Certificates in order to maintain minimal state which facilitates replication. The use of authentication protocols typically results in establishing shared keys (session keys) which are subsequently used for secure communication between the authenticated principals. Multiple authentication servers or CAs are employed in large distributed systems as it is not appropriate to register all principals with a single authority. Inter-realm authentication is typically achieved by arranging realms in a hierarchical structure.

Chapter 3

Domain-Based Access Control

This chapter presents the concept of a domain and discusses how it can be used to group objects in hierarchical domain structures and specify policy scopes. Domains and domain-based policies have been used in a number of research projects, such as the SysMan and ISDM Esprit-funded projects, as a means of managing large inter-organisational distributed systems. This chapter focuses on access control polices specified in terms of domains to provide the background that will enable the reader to follow the main part of the thesis which concentrates on domain-based access control policy enforcement.

3.1 The Domain Concept

Large distributed systems consist of a variety of objects such as servers, client interfaces, objects representing data, communication links, etc. These objects interact with each other in various ways making the management of a large system difficult in practice. A distributed system typically supports a large number of objects as well as users. In addition, a distributed system may cross organisational boundaries. This inflicts distribution of the management. Management must be structured to partition and demarcate responsibility amongst the multiple managers (Sloman 1994). Further, a variety of managers fulfilling different roles, representing different interests and operating in different contexts may have responsibility for the same object. Thus, there is also a need to model overlapping responsibilities (Becker et al. 1993).

This section presents the concept of a *domain* which has been developed as a means of grouping objects in hierarchical structures for management of large systems.

3.1.1 Objects

An object-based approach has been adopted to building domain-based access control systems. All entities in the system are considered to be objects. In this thesis we assume that an object encapsulates a *state* and provides a single *interface* on which operations on its state can be invoked. We assume that an object has a unique name, called an **object identifier** – OID, which contains the type of the object, the name of its host, the name of its server, and a unique identifier (UID) (see Figure 3.1). We assume that if the OID of an object is known, operations can be invoked on the object's interface. Objects are maintained within processes which are called *servers*. A server may maintain objects of different type.

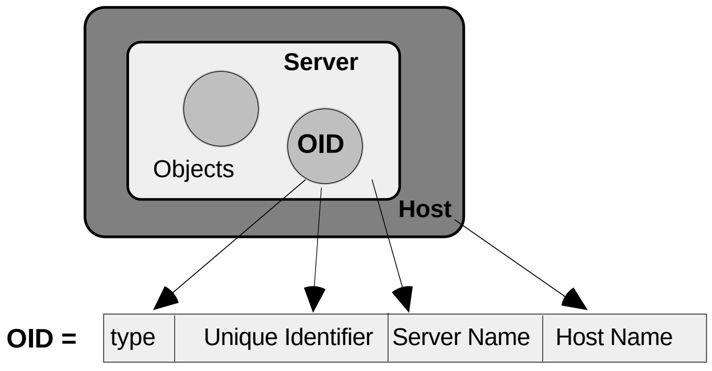


Figure 3.1 Object Identifier

3.1.2 Domains

A **domain** is an *object* which maintains a list of references to objects that have been explicitly grouped together for the purposes of management (Sloman 1994). If a domain holds a reference to an object, the object is said to be a *direct member* of that domain and the domain is said to be its *parent*. Since a domain is itself an object, it may be a member of another domain and is said to be a *subdomain* of its parent.

An object is said to be an *indirect member* of a domain A if (recursive definition):

i) it is a direct member of a domain which is a direct member of A or

ii) it is direct member of a domain which is an indirect member of A.

When there is no need to distinguish between direct and indirect members, the term *member* is used.

An object may be a direct member of multiple domains. In this case, its parent domains are said to *overlap*. Overlapping domains can be used to reflect overlapping responsibility for a group of objects.

Figure 3.2 illustrates two graphical representations of the same domain structure where DomA, DomB, DomC, DomD and ObjX represent OIDs. In this thesis, the second type of representation (directed graph) is used when there is a need to represent complex domain structures. In the illustrated domain structure, DomB and DomC are direct members of DomA. The object ObjX and the domain DomD are indirect members of DomA. The domains DomB and DomC overlap as they both maintain a reference to DomD.

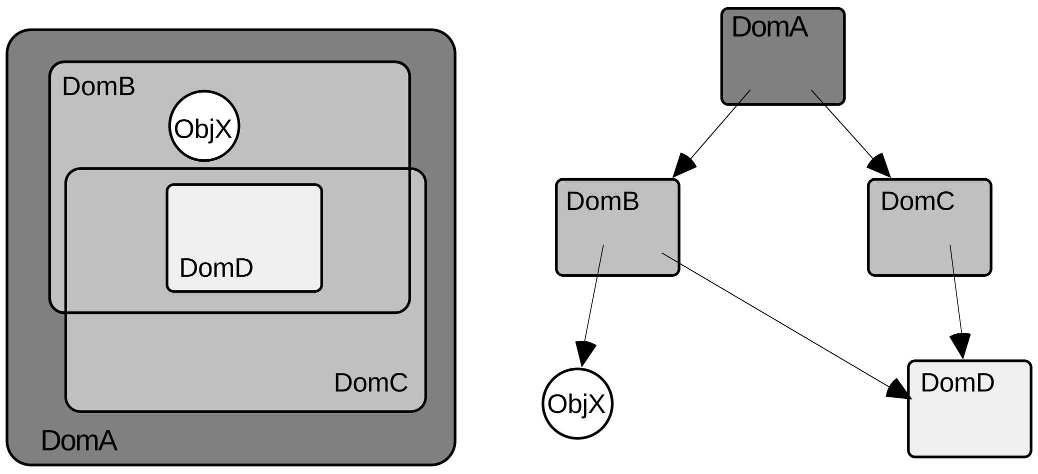


Figure 3.2 Two Graphical Representations of a Domain Structure

From the user's point of view, a domain holds *local names* referring to its direct members. The local name should be unique within a domain but it does not have to be unique amongst different domains; that is two different domains may use the same local name to refer to two different objects. An object that is a direct member of multiple domains may have different local names in each parent domain. The set of local names of the direct members of a domain is said to be the *policy set* of the domain.

Domains provide a means of naming objects that are members of domains (Twidle 1993). A top level domain or *root* domain is required to be used as an absolute reference point. If a root domain has been specified, a *path name* of an object is defined to be the ordered concatenation of domain local objects of ancestor domains up to the root plus the local name of the object itself (Becker et al. 1993). A path name of an object can be used as a name of the object. As objects may be members of multiple domains, an object may have multiple path names.

Domains are used to determine the scope to which a policy applies. They can also be used to group objects for other reasons and to provide local names for objects. For these reasons, a domain may hold local names referring to objects to which policies should not apply. Such a local name is called *alias* and the set of aliases held by a domain is called the *alias set* of the domain. The union of the policy and alias sets is said to be the *name set* of a domain.

Aliases can be used in object path names and they provide a convenient means of referring to objects that belong to domain structures of other organisations in case of federation (Twidle 1993). Managers of an organisation cannot specify policies for objects belonging to another organisation. Nevertheless, they may refer to objects in other organisations using aliases (c.f. cross mounting file systems and cross references between independent directory services).

From the implementation point of view, a named reference to an object is associated with the OID of the object. Thus a domain holds a list of (name, OID) pairs. Normally users do not have to deal with OIDs. They use local names and path names. The domain service (see Section 3.1.3) is used to translate pathnames to OIDs.

3.1.3 Domain Service

Domain objects are maintained within *domain servers*. A *domain service* is provided by a number of domain servers that are configured to suit a variety of parameters such as physical location, number of domains, etc. (see Figure 3.3). The distribution of the domain service reflects the distribution of management and increases the

- *availability* because failure of a single domain server does not imply failure of the whole domain service
- *performance* because the requests are served by a number of servers rather than a single server.
- *security* in the sense that if a domain server is compromised, the whole domain structure is not compromised.

3.2 Domain-Based Policies

A policy scheme has been developed and used within the framework of a number of research projects such as the SysMan and the ISDM. This scheme aims at managing large distributed object systems and is based on the use of domains to group objects. This section focuses on access control policies.

3.2.1 The Principle

Policies influence the interactions amongst objects in a system. A complex interaction can be expressed in terms of simpler interactions between (subject, target) pairs. A subject invokes operations on the interface of the target, while an object may act as a subject and as a target at the same time. We consider a policy in its simplest form to be a relationship between a subject and a target. When a policy specifies what actions the subject must (or must not) perform upon certain conditions, it is said to be an *obligation policy*. Obligation policies are beyond the scope of this thesis and they are not described in more detail. A detail description of this kind of policy can be found in (Marriott and Sloman 1996).

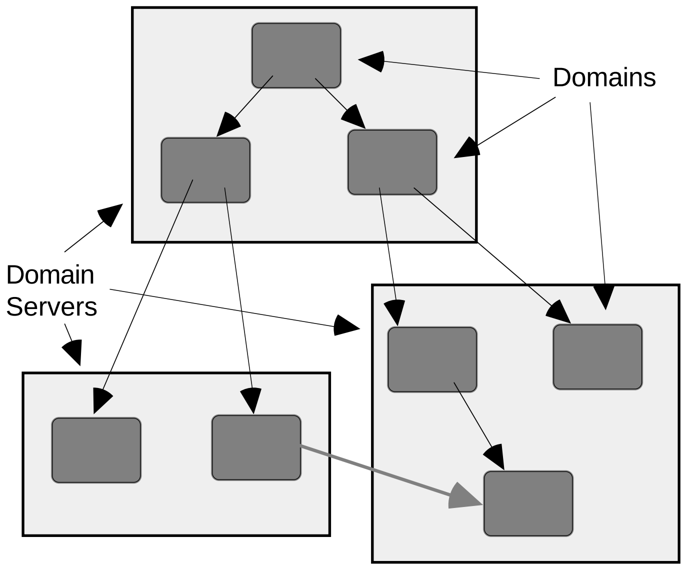


Figure 3.3 Domain Service Provided by Multiple Domain Servers

When a policy determines what actions the subject is authorised or forbidden to perform is said to be an **authorisation policy** or an **access control policy**. A policy that specifies what actions are permitted is called *positive*. A policy that specifies what actions are not permitted is called *negative*. Examples of simple access control policies are:

- Policy 1: *John is permitted to invoke operation read on file foo* (positive)
- Policy 2: *Mark is not permitted to invoke operation write on file foo* (negative)

It is useful to specify policies that apply to a group of subjects and a group of targets. In such a case the subjects are determined by the *subject scope* of the policy, while the targets are determined by the *target scope* of the policy.

As domains provide a means of grouping objects we use them to specify the subject and the target scope of a policy. An example of such a policy could be:

- Policy 3: *Any member of DomA is permitted to invoke Op1 on any member of DomB.*

If DomA and DomB hold references to subdomains, Policy 3 *propagates* to all subdomains (i.e. to all direct or indirect objects) of DomA and DomB. In some cases, it is useful to be able to restrict the propagation of the policy to direct members only or to a certain combination of subdomains and to specify policy scopes in terms of more than one domain. For these reasons *domain scope expressions* are used to determine the scope of a policy. Domain scope expressions are defined in Section 3.2.2.

Sometimes it is convenient for human managers to specify abstract policies that cannot be interpreted and enforced by the system. Usually these policies represent organisational polices that have to be enforced by the system. However, these policies can be refined gradually into more specific abstract policies and eventually into policies that can be interpreted and enforced by the system. A *hierarchy* of policies can thus be constructed representing the refinement of the organisational policies into leaf-level policies that can be interpreted by the system. A more detailed description of this scheme is given in (Marriott and Sloman 1996).

3.2.2 Domain Scope Expressions

Given a domain structure, a *Domain Scope Expression* (DSE) defines a *set of objects* in terms of domains and/or non-domain objects. In this thesis, we use Domain Scope Expressions that have the following syntax:

```
DSExp ::= ANY |
      *object |
      *N object |
      @object |
      { object } |
      DSExp + DSExp |
      DSExp - DSExp |
      DSExp ^ DSExp |
      (DSExp)
```

where:

- the interpretation of an expression is from left to right.
- `object` is a reference to an object, either an OID or pathname

- ANY refers to any object, that it defines a set containing all objects.
- and the *operators* are defined are follows:

+	set union
-	set difference
^	set intersection
*	if applied on a domain object returns a set that contains all *direct and indirect* members of the domain and the domain itself; otherwise returns a set that contains the object itself. When an integer N >0 is present, the returned set contains all the members of the domain as far as the Nth level.
@	if applied on a domain object returns a set that contains all *direct* members of that domain; otherwise Ø is returned.
{ }	returns a set that contains the objects on which it is applied.

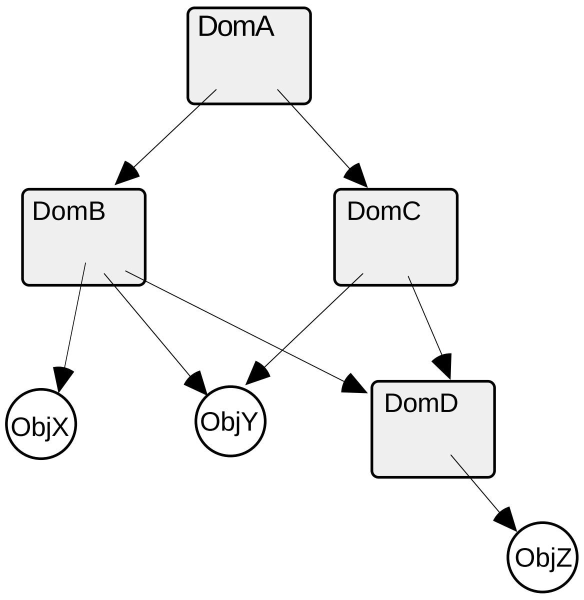


Figure 3.4 A Domain Structure

Assuming the domain structure shown in Figure 3.4, we give a number of examples to illustrate the use of the Domain Scope Expressions.

- The DSE *DomA defines the set {DomA, DomB, DomC, DomD, ObjX, ObjY, ObjZ}, that is the set containing all indirect and direct members of DomA and DomA itself.
- The DSE *2DomA defines the set {DomA, DomB, DomC, DomD, ObjX, ObjY}, that is the set containing all up to 2nd order members of DomA and DomA.

- The DSE @DomB defines the set {Objx, ObjY, DomD}, that is the set containing all direct members of DomB.
- The DSE *DomB ^ *DomC defines the set {ObjY, DomD, ObjZ}, that is the set containing all objects that are members of both *DomA and *DomC.
- The DSE *DomB - *DomC defines the set {DomB, ObjX}, that is the set containing objects in *DomB except those in *DomC.
- The DSE *DomB - {DomD} defines the set {DomB, ObjX, ObjY, ObjZ}, that is the set containing objects in *DomB except the domain-object DomD.
- The DSE {ObjX} + {ObjY} defines the set {ObjX, ObjY}.

Note that it is possible that the set defined by a DSE changes when the domain structure changes. For instance, the set defined by the DSE *DomB^*DomC becomes {ObjX, DomD, ObjZ} if ObjY ceases to be a member of DomC. This property is of importance as it necessitates the development of an efficient mechanism that can detect changes in the domain structure and re-calculate the set defined by the DSEs. Such a mechanism is discussed in Chapter 6.

3.2.3 Access Rules

Access rules are *low-level authorisation policies* that can be interpreted by the access control enforcement system (leaf-level policies). An access rule specifies a relationship between a subject and a target scope which authorises objects in the subject scope to invoke operations on objects in the target scope. Domain scope expressions are used to specify the scopes of the policy and constraints may impose restriction on the applicability of the access rule (Sloman 1994).

Although negative access rules (policies) can be useful from the security administrator’s point of view in the sense that they provide a convenient way of expressing organisational polices (e.g. students are not allowed to access the registration files) or instantly revoking access rights (Moffett 1990), we only consider implementation of *positive* access rules.

The reason we concentrate on positive access rules is that the use of negative ones may give rise to enforcement-related problems. The first problem is that, if negative rules are allowed, conflicts may occur. It may be the case that a positive rule permits a certain operation while a negative one forbids the same operation. One way to resolve this problem is to give priority to either the positive rules or to the negative ones. Alternatively, some other attributes of the policies can be used to resolve the conflict, for example the latest created rule takes priority or the policy applying to a closer parent domain of the target takes priority. Another way to tackle the conflict problem is to employ special tools that detect conflicts and inform the security administrator who can then make decisions to resolve the detected conflict.

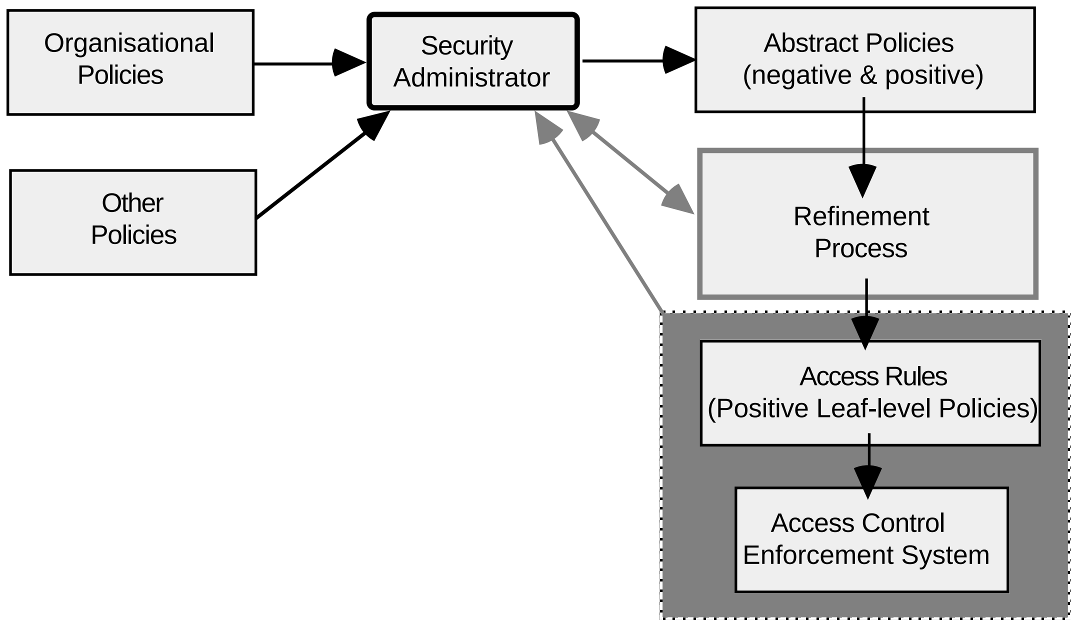

Organisational Policies
Other Policies
Security Administrator
Abstract Policies (negative & positive)
Refinement Process
Access Rules (Positive Leaf-level Policies)
Access Control Enforcement System

Figure 3.5 Policy Refinement and Enforcement

Another problem is that the enforcement of negative access rules would require verification of non-membership in domains. As rule scopes are defined in terms of domains, the access control decision mechanism may have to verify that the subject is not member of certain domains. This is equivalent to verifying non-membership in groups, in general, is not practically feasible in distributed systems (Abadi et al. 1993). This problem is further discussed in Chapter 5.

Finally, the use of negative access rules may complicate the access control decision mechanism and affect its time-efficiency. This is because it may be the case that a larger rule space has to be searched before a decision can be made.

For these reasons, we have decided to design and implement an access control mechanism on the following assumptions:

- only positive access rules are given to the access control enforcement system
- if there is no access rule permitting an operation, the operation is forbidden.

These assumptions do not necessarily debar the use of negative abstract policies at a higher level. In this case, however, a refinement process (Becker et al. 1993) has to be employed to produce positive access rules as is illustrated in Figure 3.5. The refinement process is out of the scope of this thesis and it is not discussed further.

It is worth noting that the use of the set difference operator for defining policy scopes does not imply the specification of negative access rights. That operator only limits the scope of a policy; it does not specify negative access rights for any object. Therefore no conflicts occur due to the use of set difference operators for specifying policy scopes.

A graphical representation of an access rule (Moffett 1990; Moffett and Sloman 1993) is shown in Figure 3.6. An access rule can also be represented as a tuple of the form:

(Subject Scope, Target Scope, Operations, Constraints)

The Operations field defines the operations a subject is permitted (by this policy) to invoke on a target. It gives a list of operation names along with the types of the objects on which the operations can be invoked. For instance, an operation list could be (type1:Op1, type2:Op1, Op2). Note that objects of different type may support operations that have the same name.

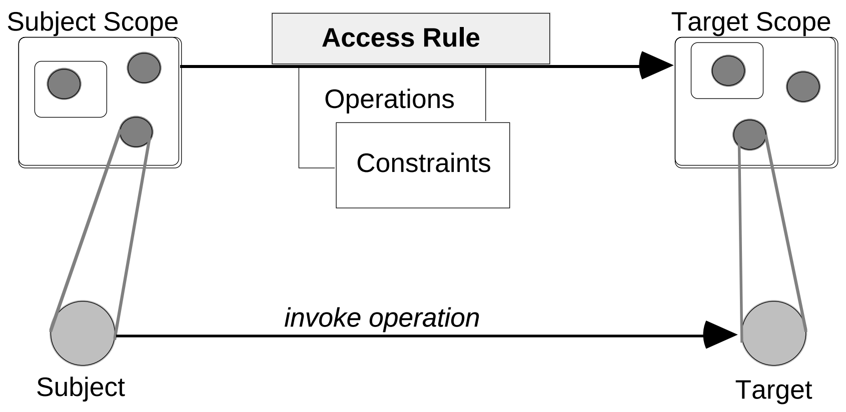


Figure 3.6 Graphical Representation of an Access Rule

The Constraints field may impose restrictions on the applicability of the rule. This field is defined as a predicate referring to global attributes or to attributes related to a specific connection between a subject and a target (see Chapter 5). For instance, a constraint may limit the applicability of the policy between 9.00PM and 5.00AM, Monday to Friday. Another example is a spatial constraint: the subject should be located on a host inside the university campus. The Constraints field may also define the required encryption for the invocation and reply data.

3.2.4 Policies Applying to Users

In the previous sections we assumed that polices give access privileges to objects. In a real system, however, most of the polices should give access privileges to users. This section describes how the same type of polices can be used to specify access privileges for users.

We take an approach to specifying access privileges for users in which each human user is associated with a specific object in the system in the sense that the object is the persistent representation of the user in the system and it is trusted to the extent that the user can be trusted. This implies that the object representing a user should be maintained on a host that is sufficiently trustworthy to act on behalf of the user.

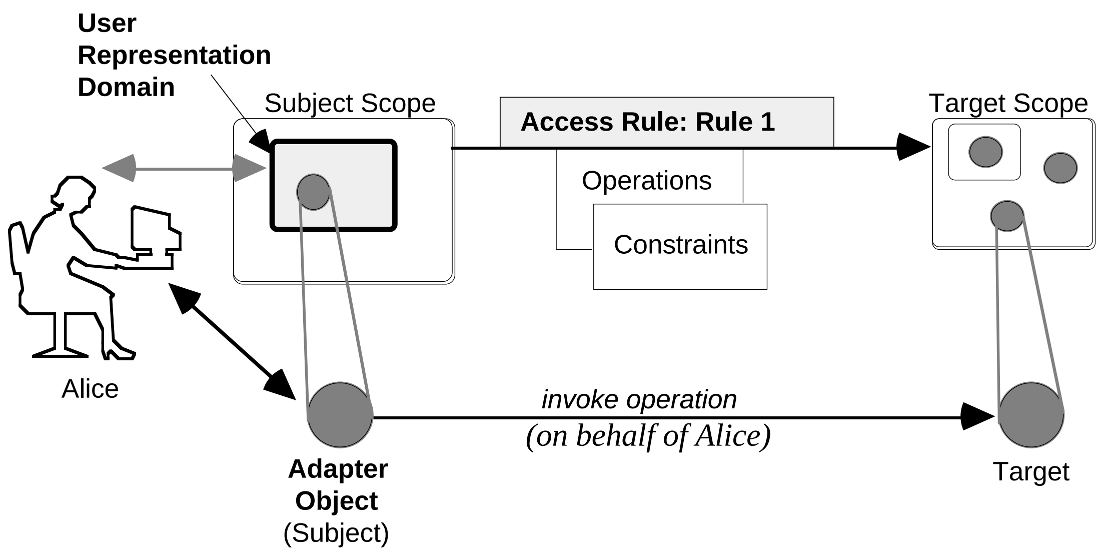


Figure 3.7 Adapter Object Acting on behalf of a User

In this thesis we assume that the object representing a user is a *domain* called the **User Representation Domain** (URD) (Sloman 1994; Twidle 1993) The policies applying to that domain as a subject, define the access rights the associated user possesses. Normally, users use a variety of tools to interact with the system. These are called *adapter objects*. If a user makes an adapter object member of her URD, the object inherits the access rights specified for the URD. Thus, the adapter object can invoke operations with the access privileges of the user. This is illustrated in Figure 3.7 where the access rule Rule 1 defines access rights for Alice's URD. Alice uses an adapter object which is a member of her URD and thus belongs in the subject scope of Rule 1. Normally, an adapter object resides on the login workstation.

3.2.5 Policy Service

Policies are represented within the system by policy objects. These are maintained by a number of *policy servers* which collectively provide the distributed *policy service* (see Figure 3.8). An administrator that possesses the necessary access rights can create policy objects and then edit, activate, disable and delete them. When a policy is activated it is distributed to the relevant components of the system to be enforced. This applies to both the obligation and authorisation policies but the actual mechanism employed in order to the distribute a policy may vary. Chapter 6 describes a distribution technique for access rules that has been implemented in the Orbix^{TM} environment.

The administrator interacts with policy objects within a policy server using a policy editor which provides a sophisticated graphical interface. That editor enables the user to see the state and the attributes of various policies, to monitor and control policy refinement procedures as well as to view the domain structure in order to specify policy scopes (Marriott and Sloman 1995; Marriott and Sloman 1996).

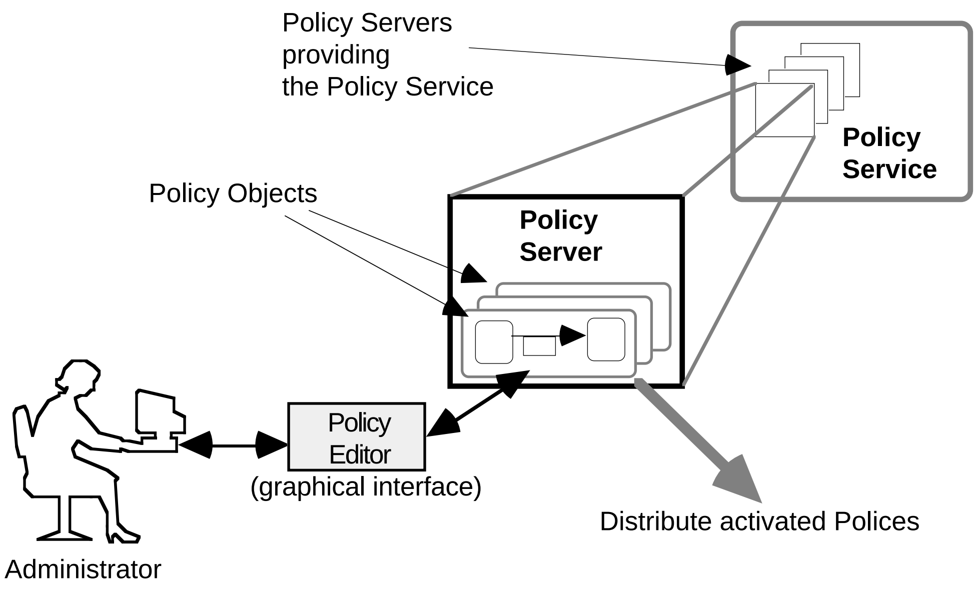


Figure 3.8 Administrator Interacting with the Policy Service

3.2.6 Authorisation Policies for Security Administrators

A large distributed system is normally managed by multiple administrators who are responsible for different parts of a domain structure. Therefore, there should be a way to specify who has the right to specify polices for certain groups of objects. In (Becker et al. 1993) the notion of an *Authorisation Policy for a Security Administrator* is discussed.

Such a policy is actually a meta-policy about managing policy objects and specifies a number of scopes which limit the access rules that can be created. Figure 3.9 shows such a policy which specifies to whom access can be given, the group of objects to which access can be given as well as the set of operations that can be included in an access rule.

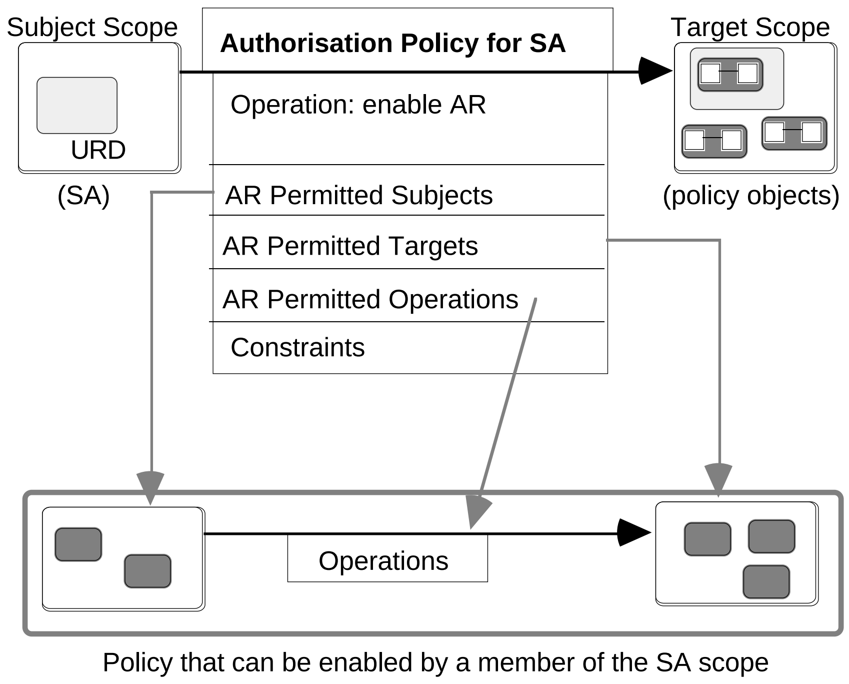


Figure 3.9 Authorisation Policy for a Security Administrator (SA)

3.2.7 Roles

Research work is currently carried out in order to support role-based access control and role-based management using the concept of a domain. In a simple view, a role is composed of a *Role Position Domain* (RPD) and a set of access control and obligation policies. An RPD represents a position in an organisation such as security administrator, or departmental manager. The main advantage of using roles is that individuals can be assigned to or withdrawn from the role positions without having to respecify policies. A detailed description of the role framework is given in (Lupu and Sloman 1996; Lupu et al. 1995).

Although we do not have to say much about roles in this thesis, it should be pointed out that the architecture presented in this thesis can also support role-based access control as the use of roles does not change the form of the access rules which are still defined in terms of domains (Yialelis et al. 1996). In addition, roles provide an elegant way to refer to *sets* of access rights. This can be used to define the access rights that an object intends to delegate to a proxy server. Delegation of access rights is discussed in Chapters 4 and 6.

3.3 Chapter Summary

Distributed systems may support a very large number of objects. In addition, a typical distributed system spans a number of different organisations. This necessitates the distribution of management to reflect the distribution of the system being managed.

In order to make the management of these systems feasible in practice, the concept of a *domain* has been used in a number of research projects. This chapter presented this concept and described how domains can be used to group objects in hierarchical structures and specify policies in terms of groups rather than in terms of individuals. A *domain* is an object that maintains references to objects that have been explicitly grouped together for the purposes of management. Since a domain is itself an object, it may be a member of other domains and thus hierarchical domain structures can be constructed.

This chapter has also discussed access control policies and especially *positive access rules*. An access rule is a low-level access control policy that specifies a relationship between a subject and a target scope in terms of operations that objects in the subject scope are permitted to invoke on objects in the target scope. The scopes of an access rule are defined in terms of domains using Domain Scope Expressions. These expressions, along with domains, provide a very flexible mechanism to group objects. A variety of *constraints* can also be specified in order to limit the applicability of the access rules.

Further, other concepts that are relevant to the security architecture presented in this thesis have been outlined. The domain service is provided by a number of domain servers maintaining domain objects and is distributed in order to increase its availability, performance and security and to reflect management distribution.

The policy service which is provided by a number of policy servers maintaining policy objects also affects the design of the security architecture. The access rules (maintained by the policy service) should be distributed to the components providing access control services to the objects in the scopes of these rules. Because the domain structure is not static, there is a need to provide a mechanism that dynamically re-evaluates the rule scopes when the domain structure changes.

Chapter 4

Delegation of Access Rights

This chapter discusses the issue of delegation of access rights in a distributed environment. Delegation is of crucial importance in distributed systems as resources are distributed over multiple hosts which are trusted to different levels. This chapter discusses related work and proposes an extension to the concept of an access rule that enables security administrators to control the delegation of access rights. Delegation protocols, which typically involve the authentication system, are discussed in Chapter 7.

4.1 Introduction

In distributed systems the resources required for certain operations are often distributed over multiple hosts. The invocation of a single operation on a target may result in further invocations made by the target and this may be repeated a number of times creating a tree of operation invocations before the first operation is completed (Figure 4.1). A typical example of *cascaded invocations* is a DBMS that makes invocations on file servers in order to serve queries made by its clients. Another one is the case in which a customer requests her agent to perfom a purchase but the agent, in order to complete the purchase, has to debit the customer's account (maintained by a bank server).

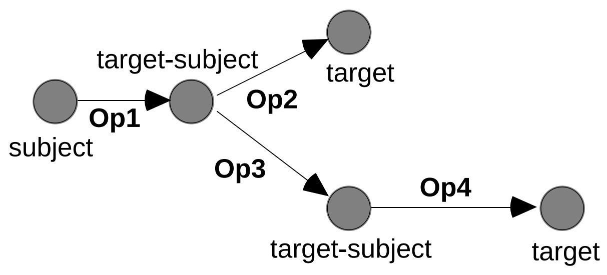


Figure 4.1 A Tree of Cascaded Invocations

It will soon become apparent that cascaded invocations complicate the provision of access control. In addition, they introduce problems with respect to authentication. This chapter deals mainly with the access control related problems though we cannot avoid some references to authentication issues. Later, in Chapter 7, an authentication scheme that supports cascaded operations in a distributed environment is covered in detail.

4.1.1 The Problem

We assume a simple scenario of cascaded invocations where the target invokes an additional operation in order to complete the operation requested by the subject. In Figure 4.2, the first object *A* invokes an operation *Op1* on *B*. This cannot complete the operation itself so it invokes *Op2* on the third object *C*. If we assume that *B* has the right to invoke the second operation, then this scenario does not exhibit special difficulties and the usual access control model can be applied. It is interesting to examine the case in which *A* possesses the right to invoke *Op2* while *B* does not. This case is more realistic in that *A* invokes *Op1* in the framework of a task and therefore *A* is the one that typically possesses the right required to complete the task consisting of *Op1* and subsequent operations invoked by *B*. The assumption that *B* possesses the right to invoke *Op2* while it is acting on behalf of *A* is not realistic in a distributed environment that consists of mutually suspicious entities. According to most current techniques used in centralised systems, the intermediate is either considered privileged and take the identity of the subject or relies on system access rights. This violates the *least privilege principle* (LPP) according to which an object should have the minimum access rights it needs to perform its task. Though the violation of this principle may be acceptable in some centralised systems or small distributed systems where all the hosts are managed by a single authority that is globally trusted, it is unacceptable in large inter-organisational distributed environments. In these environments, the intermediate target may serve a number of clients that possess rights given by authorities different from the authority giving access privileges to the intermediate. Consider for example an ‘customer-agent-bank’ example where the agent serves more than one customer. Customers have been given by their banks the right to debit their accounts but it would not be reasonable to assume that these rights can also be possessed by the agent.

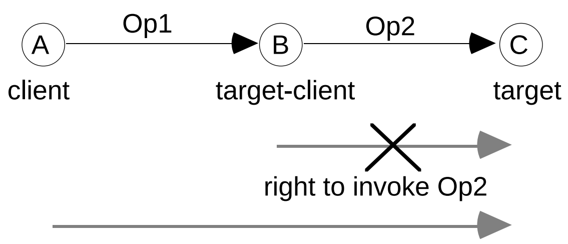


Figure 4.2 A Cascaded Invocation Scenario

In many situations, however, limited trust between the subject and the intermediate target can be established and thus the subject can authorise the target to act on its behalf for a limited period of time. Consider the 'customer-agent-bank' example, where the customer is familiar with the agent and so she can trust the agent to possess the right to debit her account for a limited period of time and perhaps up to a certain amount. In this case the subject can *delegate* its access rights to the target which can subsequently invoke operations using these rights. In fact, this process is usual in the real world; for instance Alice has to travel away for a couple of months and she authorises her friend Bob to deal with her bank account in that period.

4.1.2 Some Definitions

In this thesis we use the term *delegation* to refer to the process whereby a principal authorises an agent to act on its behalf to access certain services or resources for a limited period of time.

The principal that delegates is said to be the *grantor*. The principal that is authorised to act on behalf of the grantor is said to be the *grantee*. The target that accepts the last operation invocation of a cascaded operations chain (for example object *C* in Figure 4.2) is called the *end-point.*

In the literature, the grantor sometimes is called *initiator* delegator or originator. The grantee is sometimes called *intermediate proxy* or *delegate* and the end-point may be called *final target* (Kaijser et al. 1994). In this thesis we always use the terms grantor, grantee and end-point.

4.1.3 Cascaded Delegation

It is possible that a grantee further delegates to a second grantee the access rights it has been delegated, the second delegates to a third, etc. In this case it is said that *cascaded delegation* takes places. This is illustrated in Figure 4.3.

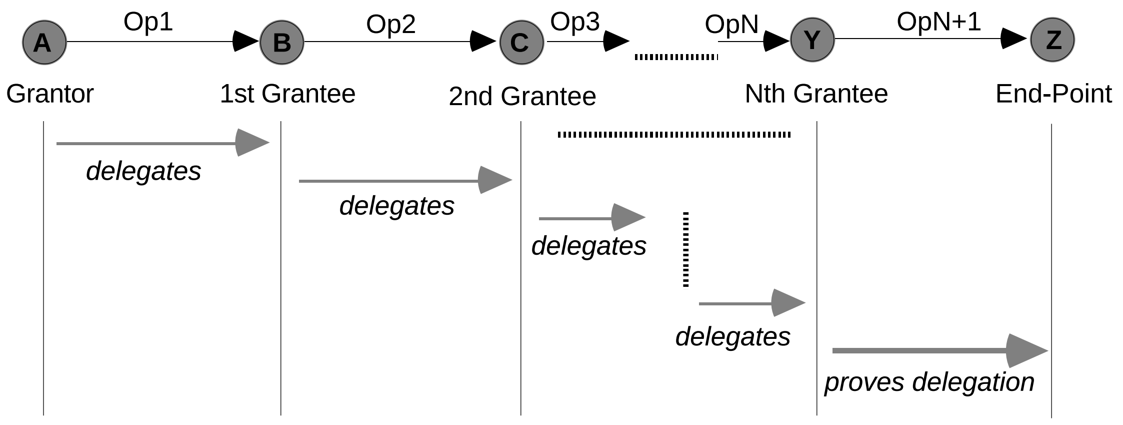


Figure 4.3 Cascaded Delegation

Some grantees may forward the delegated access rights to multiple successor grantees which results in a tree of delegation steps. It is possible that the grantees (except the first one) have to verify the preceding delegation steps. This is true when the invocation made by the preceding grantee requires access rights that have been delegated. A grantee may delegate all or a subset of the delegated access rights as well as the rights it possesses.

4.1.4 Discussion

The delegation issue turns out to be quite complex. Figure 4.4 sketches the main aspects of the issue. A basic problem which has not received much attention is the specification of a policy that controls delegation, i.e. which delegations are considered to be valid and therefore can be allowed.

Another problem is the restriction of the delegated rights. In the simplest case, the grantor delegates all its access rights to the grantee. But this violates the LPP. An obvious improvement is for the grantor to delegate only the rights required for the task the intermediate has to perform. In practice, however, this is not easy because the subject cannot predict the exact access rights required for the task the grantee has to perform. Therefore, the determination of the rights to be delegated is an issue.

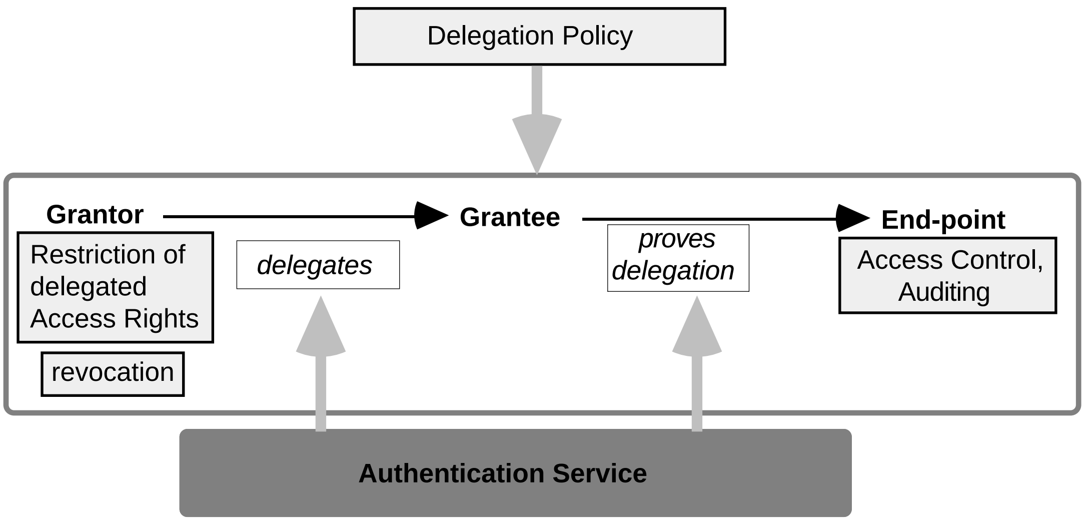


Figure 4.4 Sketch of the Delegation Issue

Another aspect of the delegation issue is the access control mechanism employed to decide whether a delegation chain is acceptable and therefore access can be granted.

Revocation of delegated access rights is also important. A grantor may want to revoke access rights that it has delegated either because the requested task has been accomplished or because it ceases to trust the grantee to make correct use of the delegated rights. This also applies to a grantee that forwards delegated access rights to subsequent grantees.

Finally, the end-point has to verify the claimed delegation as well as the identities of the involved principals in order to grant access to the grantee. The verification of the delegation steps as well as the process of the delegation involve the authentication system and they are discussed in Chapter 7. This chapter deals mainly with the specification of delegation policy, access control decision making and restriction of delegated access rights.

4.2 Related Work

The issue of delegation has received considerable attention in recent years due to the increasing use of distributed systems. Most of the work, however, is about mechanisms that achieve delegation verification. Little progress has been made in developing mechanisms that enable the security administrator to control the delegation of access rights in a flexible and efficient way. This chapter attempts to review work related to delegation and especially to delegation policy, access control when delegation is involved and restriction of delegated access rights.

4.2.1 General

Karen Sollins recognises in (Sollins 1988) that the problem of cascaded authentication has arisen in distributed systems that consist of mutually suspicious components. Cascaded authentication is defined to be the provision of authentication supporting access control and accounting when cascaded invocations take place in a distributed environment.

The author defines six goals of a cascaded authentication scheme. We summarise these goals using the terminology introduced in Section 4.1.2.

- *Unforgeability*: there should be a mechanism that enables the end-point to verify the integrity of the information used for access control and accountability.
- *Accountability*: it is important to provide identification of all the grantors in a delegation chain. The end-point should be able to track the route of cascaded delegations.
- *Discretionary restriction*: each grantor should be able to restrict the access rights it forwards to the next grantee.
- *Modularity*: it is important that a subject does not need to know the internal structure and implementation of the service it invokes. The subject may not know the successive grantees or the end-points
- *Independence*: a subject may not be available when authentication is required. The end-point should be able to verify the identity of the initial grantor as well as the identities of the preceding grantees even if they are not available.
- *Combining of identity*: it is often necessary to be acting as a combination of oneself and a subject. In some cases the end-point will only grant access because a grantee acts on behalf of a certain grantor.

The author proposes a cascaded authentication scheme which is based on the use of nested digitally signed tokens named *passports*. The verification of the passports is based on symmetric cryptography, though it could be modified to make use of an asymmetric authentication mechanism.

Sollins does not address the crucial problem of *revocation* of delegated access rights. (Varadharajan et al. 1991) discusses the problem of revocation in detail, compares a number of different approaches and proposes two delegation protocols that can be based either on symmetric or asymmetric authentication systems. The second of these schemes is similar to the one proposed in (Sollins 1988).

C. Neuman discusses the use of restricted proxies (Neuman 1991; Neuman 1993) . A *proxy* is defined to be a token that allows a principal to operate with the access rights of the principal that granted the proxy. The author identifies the need to control the access rights represented by a proxy. The term *restricted proxy* refers to a proxy that imposes conditions on its use and thus restricts the access rights delegated to the grantee. The author shows how restricted proxies can support authorisation and accounting services, and discusses a number of different types of restrictions that can be imposed on the use of a proxy.

Kerberos V5 allows a client to grant a proxy to a server which can take on the identity of the client to perform operations on its behalf. Note that the end-point is not provided with the identity of the grantee but it can realise that the client acts on behalf of a grantor by checking a flag named *PROXY* in the presented ticket. Thus, the end-point may require additional authentication from the grantee for auditing purposes (Kohl and Neuman 1993).

(Gasser and McDermott 1990) give a detailed description of a delegation mechanism which is integrated with the DSSA access control model that is based on the Access Control List (ACL). A similar scheme is adopted in the Taos Operating System (Wobber et al. 1993). The Calculus described in (Abadi et al. 1993) gives a formal description of that scheme which is also discussed in (Lampson et al. 1992).

4.2.2 Delegation in Taos

In (Abadi et al. 1993) and (Lampson et al. 1992) delegation is expressed with the operator ***for*** (see also Chapter 2). The fact that *B* acts on behalf of *A* is expressed as *B **for** A*. In (Lampson et al. 1992)the operator ***for*** is axiomatised by the axioms:

Axiom 1 $A \wedge B|A \Rightarrow B\,\textbf{\textit{for}}\,A$

Axiom 2 ***for*** is monotonic and distributes over $\wedge$

Axiom 3 $A\,\textbf{\textit{for}}(B\,\textbf{\textit{for}}\,C) \Rightarrow (A\,\textbf{\textit{for}}\,B)\,\textbf{\textit{for}}\,C$

Axiom 4 $(A\,\textbf{\textit{for}}\,B)\textbf{\textit{as}}\,R = A\,\textbf{\textit{for}}(B\,\textbf{\textit{as}}\,R)$

In (Abadi et al. 1993) ***for*** is defined in terms of $\wedge$ and | and a fictional principal *D* which is considered to be a globally trusted delegation server. In both approaches, when *A* has to delegate to *B*, it makes *A **says** B|A* $\Rightarrow$ *B **for** A*. Then *B* must accept the delegation by making *B|A **says** B|A* $\Rightarrow$ *B **for** A*. Thus ***for*** equals delegation plus quoting. (Lampson et al. 1992) points out that when the compound principal *B **for** A* says something, the intended meaning is that both *A* and *B* contribute, and hence both must consent. This implies that both principals are responsible for the actions of the grantee and the compound principal *B **for** A* can be used for auditing purposes. We can now further discuss the ACL scheme presented in Chapter 2.

The syntax of an ACL entry is given below (Lampson et al. 1992):

principal	::= forList | principal $\wedge$ forList

forList	::= asList | forList ***for*** asList
asList	::= properPrincipal | asList ***as*** role
role	::= pathName
properPrincipal	::= pathName | channel

A request is granted if the requester (a compound principal) implies one of the ACL entries. A typical example of an ACL entry is:

$(G' \textbf{ as } R_B)$ **for** $(G \textbf{ as } R_A)$

meaning that access is given to a principal if it has adopted the role R_B (or a stronger one), is a member of the group *G'* and acts on behalf of a principal that is member of the group *G* and has adopted the role R_A (or a stronger one). If we assume the delegation scenario shown in Figure 4.5, the Reference Monitor controlling access to the end-point *Z* grants access to *B* as a result of the above-mentioned ACL entry provided that it can verify certificates and credentials stating that:

- $A \Rightarrow G$, that is *A* speaks for *G*, which in this context implies *A* is a member of the group *G*
- $B \Rightarrow G'$, that is *B* is a member of the group *G'*
- $R_A' \Rightarrow R_A$, the role R_A' is stronger than the role R_A
- since the request comes from the compound principal $(B \textit{ as } R_B)$ **for** $(A \textbf{ as } R_A')$, delegation credentials are required to prove that *B* ***as*** R_B has indeed been delegated by *A* ***as*** R_A'

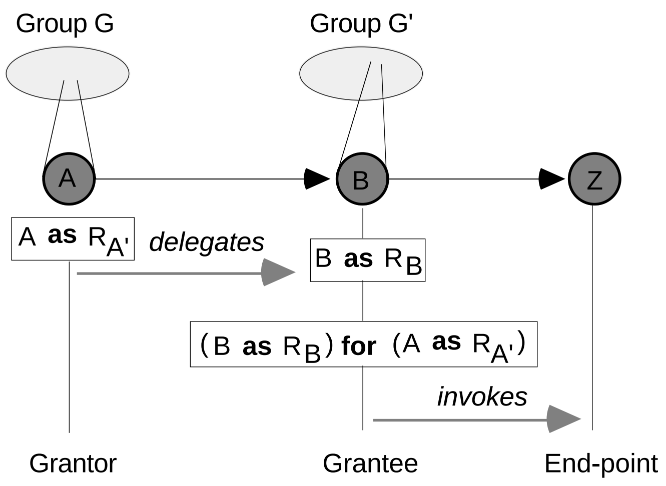


Figure 4.5 One-hop Delegation

We can make the following remarks:

- **R1** The Reference Monitor knows the identity of the grantor as well as the identity of the grantee on whose behalf the invocation request is made.
- **R2** The Reference Monitor grants access to a grantee only if there is an ACL entry permitting the claimed delegation. The security administrator has to specify which delegation sequences are acceptable. Proof that a claimed delegation has indeed taken place is not sufficient. This means that if an ACL entry reads: *A* ***as*** R_A and the subject *B* requests access as a grantee of *A* in role R_A, that is *B* ***for*** *(A* ***as*** R_A*)*, access will not be granted (at least not as a result of this ACL entry) even if *B* can prove that it has been delegated by *A* ***as*** R_A. This implies that principals cannot delegate the rights they possess to other principals unless this is permitted by the relevant ACLs.
- **R3** A principal can determine the access rights it wants to delegate by adopting a certain role. In this example the principal *A* has adopted the role R_A and thus the rights that are delegated to *B* are those associated with the role R_A.
- **R4** An ACL entry can specify possible grantors and grantees in terms of groups. For example, according to the ACL entry *G'* ***for*** *(G* ***as*** R_A*)*, any member of the group *G'* can act as grantee of any member of the group *G* delegating rights associated with the role R_A.
- **R5** An ACL entry specifies the access rights that can be delegated in terms of roles.
- **R6** *A* delegates to the principal *B* in role R_B. This means that if there is no ACL entry that enables *B* to act in that role or a stronger one, the delegation is considered to be invalid and the reference monitor does not grant access. That is, a grantor may impose requirements about the trustfulness of the grantor that are checked by the reference monitor.

We now give a more complex example that will enable us to make further comments on the Taos delegation scheme. Let us assume that an ACL entry reads:

(*G'' as* R_C*)* ***for*** *((G' as* R_B*)* ***for*** *(G* ***as*** R_A*''))*

This ACL entry matches the compound principal *(C* ***as*** R_C*)* ***for*** *(((B* ***as*** R_B*)* ***for*** *(A* ***as*** R_A*))* ***as*** R_A*')* (see Figure 4.6) if the reference monitor is provided with certificates and credentials which state that:

- $A \Rightarrow G$
- $B \Rightarrow G'$
- $C \Rightarrow G''$
- $R_A \Rightarrow R_A''$

- $R_{A'} \Rightarrow R_{A''}$
- (A ***as*** R_A) has delegated (B ***as*** R_B)
- (((B ***as*** R_B) ***for*** (A ***as*** R_A)) ***as*** $R_{A'}$) has delegated to (C ***as*** R_C)

We can notice that:

- **R7** *B* delegates as *((B **as** R*$_B$*) **for** (A **as** R*$_A$*)) **as** R*$_{A'}$ which is equivalent to *(B **as** R*$_B$*) **for** (A **as** R*$_A$ *as R*$_{A'}$*)* (see Axiom 4). If the role *R*$_{A'}$ is weaker than *R*$_A$, *(B **as** RB)* actually *restricts* the access rights of *(A **as** RA)* that are forwarded to *(C **as** R*$_C$*)*. Therefore, each grantee can restrict the access rights it forwards to the next grantee.
- **R8** The ACL entry defines the exact number of delegation hops that are permitted.
- **R9** The ACL entry defines the exact sequence of the permissible grantees in a delegation chain.

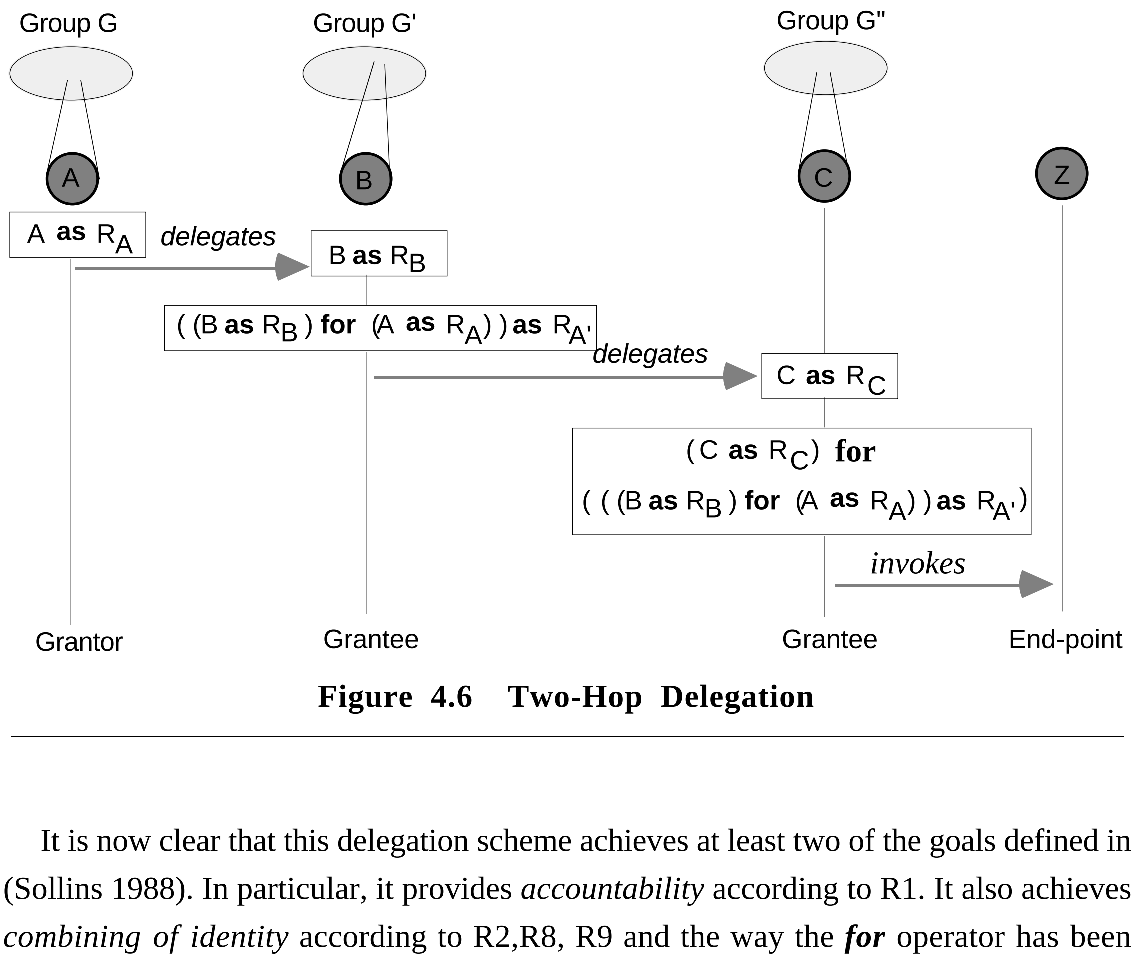


Figure 4.6 Two-Hop Delegation

It is now clear that this delegation scheme achieves at least two of the goals defined in (Sollins 1988). In particular, it provides *accountability* according to R1. It also achieves *combining of identity* according to R2,R8, R9 and the way the ***for*** operator has been defined. It also fulfils partially the *discretionary restriction* requirement according to R3 and R7. The grantor or the grantee can restrict the delegated access rights by adopting a weaker role.

It is interesting to comment on the use of roles for restricting access rights. According to (Varadharajan et al. 1991) and (Neuman 1991) the use of roles as a means of restricting access rights is too restrictive, as they are not manipulated as easily as are access rights. Roles have to be defined in advance and it is not easy to alter the ACLs when a role is created or modified. Furthermore, it is not reasonable to assume that the corresponding role has been defined for each set of access rights that a principal may intend to delegate. What is more, a grantor has no way to define special restrictions such as the maximum number of times a grantee can use a service. This weakness is also mentioned in (Gasser and McDermott 1990) (which describes the DSSA delegation mechanism in detail) and the authors state that a more dynamic restriction scheme may be adopted if the need arises.

Although the lack of a more flexible mechanism for restricting access rights is a weakness, we believe that in practice the role scheme provides a satisfactory solution. It is difficult for a grantor, either a human or an entity within the system, to determine in advance the precise access rights required by the grantor. Thus, it is likely that in most cases the grantor delegates a superset of the rights that the grantee really requires to perform a task. Roles provide a convenient means of referring to a set of access rights which is usually associated with a position, for instance security administrator, manager etc. Therefore, it is much easier for a principal to determine the role that is appropriate for a task rather than explicitly specify the exact access rights required by the grantee to perform the task.

Ideally, though, a user should be able to specify sets of access rights required for certain tasks in addition to the existing roles without having to alter the ACLs of the end-points. This cannot be done in the above-mentioned scheme. A user has to refer to roles which are usually pre-defined by the security administrators.

Another interesting characteristic of that delegation scheme is that an ACL entry mentions the exact number and order of the permissible delegation hops (see R8 and R9). Though this provides a high degree of security we have reservations about the practicability of specifying the exact order of permissible delegation steps. It complicates the specification of the access control policy, as it is not easy for a security administrator to foresee all the possible delegation chains required by various tasks. Furthermore, in many cases the order of the delegation steps is not of importance and it is more convenient to specify a group of principals that are trusted to be delegated a particular set of access rights rather than specifying a number of permissible delegation sequences.

According to (Abadi et al. 1993) the meta-logical operator $(\ )^+$ can be used to denote any positive number of iterations of a principal. For example *A* ***for*** G^+ is a shorthand for the list: *A* ***for*** *G, (A* ***for*** *G)* ***for*** *G, ((A* ***for*** *G)* ***for*** *G)* ***for*** *G, ...* We believe that the use of this operator could make the specification of the ACLs much easier in that the administrator does not have to specify the exact number and order of the permissible grantees.

4.3 Extended Access Rules

This section introduces an extension to the notion of the access rule that has been discussed in Chapter 3. This extension enables the security administrators to control the delegation of access rights. We analyse the problem of specifying policy about delegation of access rights and explain how the proposed extension to the access rules copes with this problem.

4.3.1 Controlling Delegation in Distributed Systems

When a principal delegates, it trusts the grantee to make correct use of the delegated access rights. It is the responsibility of the grantor not to delegate to a grantee that is not sufficiently trusted. However, in distributed systems where a large number of mutually suspicious entities are involved, it is not easy to make decisions about which remote principals are trustworthy. Thus, an administrator who determines the access rights possessed by principals (humans or system objects) may want to restrict the possible grantees of certain access rights so that the normal possessors of these rights cannot delegate them to untrusted principals, either by mistake or deliberately.

It may also be the case that possessors of the same access rights are not trusted equally to judge the faithfulness of possible grantees. In addition, different principals may have different needs in terms of delegation. For these reasons, it is important that the security administrator can also specify who can delegate access rights. Finally, the security administrator may wish to impose special restrictions on the delegation of rights. For instance, s/he may want to restrict the maximum delegation period to one hour. Therefore, we conclude that a security administrator should be able to specify:

- Which access rights can be delegated
- Who can delegate these access rights (possible grantors)
- To whom the access rights can be delegated (possible grantees)
- Special restrictions on delegation, such as time restrictions, maximum delegation period, etc.

4.3.2 Extended Access Rules

In order to allow security administrators to control the delegation of rights, we have extended the notion of the access rules to include a scope called *grantee scope*. So an *extended access rule* is represented by a tuple of the form:

(Subject Scope, Target Scope, Grantee Scope, Operations, Constraints)

The grantee scope is specified using scope expressions (see Section 3.2.2) while the constraints field may contain *delegation constraints* in addition to the constraints a normal access rule contains (see Section 3.2.3). The semantics of an extended access rule is that objects in the subject scope can delegate to objects in the grantee scope the right to invoke operations specified in the operation field on objects in the target scope. This is illustrated in Figure 4.7 where a graphical representation of an extended access rule is given.

An extended access rule also permits cascaded delegation provided that all grantees are in the grantee scope of the rule (see Figure 4.8).

Let us now assume that an extended access rule is translated into an ACL entry associated with an object in the target scope. We try to express that entry using the access control calculus defined in (Abadi et al. 1993).

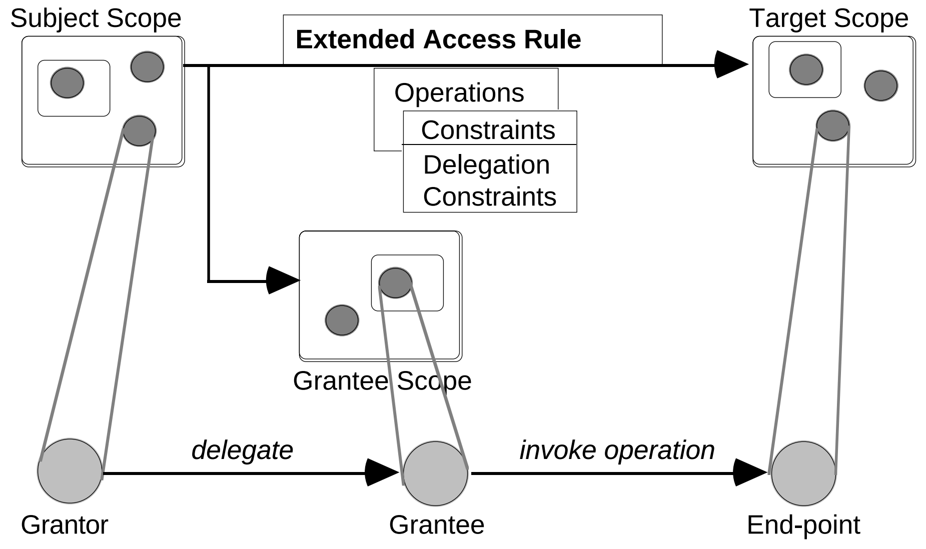


Figure 4.7 Graphical Representation of an Extended Access Rule

The scopes of the rule actually define sets of objects (see Section 3.2.2). Therefore, we can assume that the subject scope is represented by a group *Gs* while the grantee scope is represented by a group *Gg*. Furthermore, an access rule specifies a set of access rights. We assume that this set is represented by a role *R*. Thus, as an object in *Gs* can directly invoke operations, the ACL entry should read: *(Gs **as** R)*. If we allow one step delegation, we should add a new ACL entry:

*(Gg **as** R) **for** (Gs **as** R).*

That is, any object in *Gg* (grantee scope) can act on behalf of any object in *Gs* (subject scope). If we now allow two step cascaded delegation, we should add a third ACL entry:

*(Gs **as** R) **for** ((Gg **as** R) **for** (Gs **as** R)).*

If we keep increasing the number of permitted delegation steps, we add new ACL entries:

*(Gs **as** R) **for** ((Gs **as** R) **for** ((Gg **as** R) **for** (Gs **as** R)))*

*(Gs **as** R) **for** (Gs **as** R) **for** ((Gs **as** R) **for** ((Gg **as** R) **for** (Gs **as** R))))*

….

If we introduce the operator $^{+}$ that denotes any positive number of iterations of a principal, all these entries can be replaced by the following two:

- *(Gs **as** R)*
- *(Gg **as** R)*$^{+}$ ***for** (Gs **as** R)*

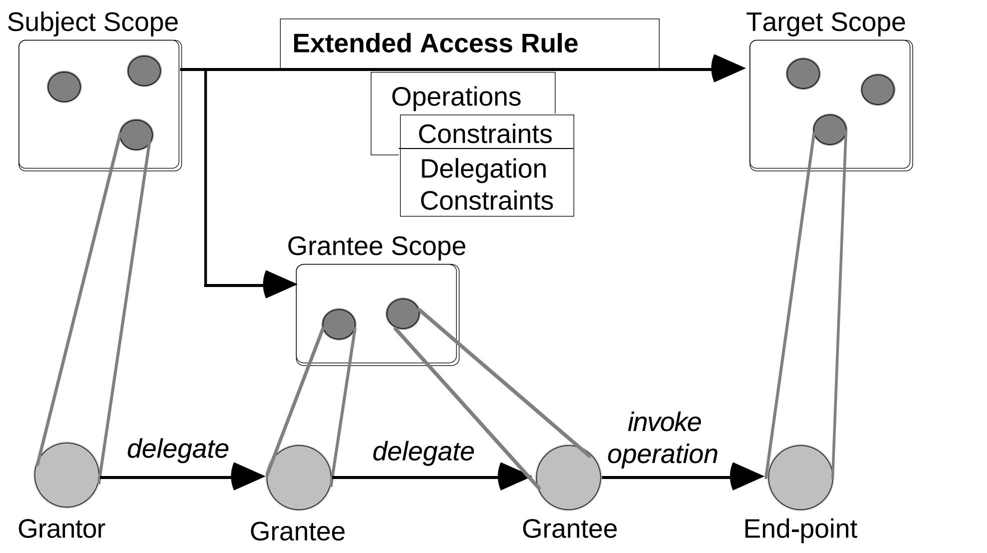


Figure 4.8 Cascaded Delegation Permitted by an Extended Access Rule

Thus an Extended Access Control Policy does not specify the order of possible grantees and a delegation chain may involve an arbitrary number of grantees. The administrator can only specify a group of possible grantees that can be trusted to be delegated certain access rights.

Though the notion of an Extended Access Control Policy can be modified to specify an arbitrary number of Grantee scopes which can be used to specify the order of possible grantees, this would complicate the specification of policy from the administrator’s point of view. As mentioned in Section 4.2.2, in many cases it is not easy for the administrator to foresee the exact order of delegation steps required for various tasks. In addition, we believe that the fact an object can be delegated certain access rights is more important than its exact position in a delegation sequence.

On the other hand, the administrator can impose other restrictions on delegation using the Constraints field. For example, one can set a maximum number of delegation hops, restrict delegation within working hours, or limit the possession of delegated access rights to one hour maximum.

Chapter 6 discusses how an extended access control policy is disseminated to the distributed access control components that have to enforce it. In addition, it shows how a grantor can restrict the access rights it delegates. Chapter 7 discusses delegation from the point of view of authentication.

4.3.3 An Example

We discuss a simple example to show how extended access rules can be used in practice. Let us assume that a user uses a remote server (e.g. DBMS) which accesses files on behalf of her. Further, at some point the user requests the server to print a certain file. The DBMS forwards the request to a printer server. Table 4.1 gives a set of extended access rules. These rules refer to a hypothetical domain structure which is shown in Figure 4.9.

Initially we assume that an adapter object *A* in the URD of Bob (*A* represents Bob in the system and inherits all his access rights) invokes a query on *DBMS_1* (Figure 4.10). According to *AR3*, *A* has the right to use *DBMS_1*. *A* also delegates its rights to *DBMS_1* which can now read *File_B* according to *AR2*. Subsequently, *A* requests a printout of *File_B*. This request is forwarded to the print server *Printer_2*. In addition, *DBMS_1* forwards *A*'s access rights to *Printer_2*. *DBMS_1* has been delegated *A*'s access rights and therefore it can invoke operations on *Printer_2* according to *AR4*. *Printer_2* invokes a *read* operation on *File_B* which is authorised because of the delegation chain *(A -> DBMS_1 -> Printer_2)* and *AR2*.

We assume in this example that *A* delegates all its access rights to *DBMS_1*. If *A* chooses to delegate only the access rights defined by *AR2*, *DBMS_1* will not be able to invoke a print operation on *Printer_2*. Furthermore, note that *A* has the right to invoke a *write* operation on *File_B* (*AR1*). It cannot, however, delegate this right to *DBMS_1*. Finally, neither *A* nor any grantee of it can use *Printer_1* which belongs to the *Trusted_Printers* domain.

Alice, who is a trusted user, can use *Printer_1* and she can also read *File_A*. She can delegate these rights to *DBMS_1*. She cannot, however, delegate her right to read *File_A* to *Printer_2* because *Printer_2* is not a member of the domain *Trusted_Printers*.

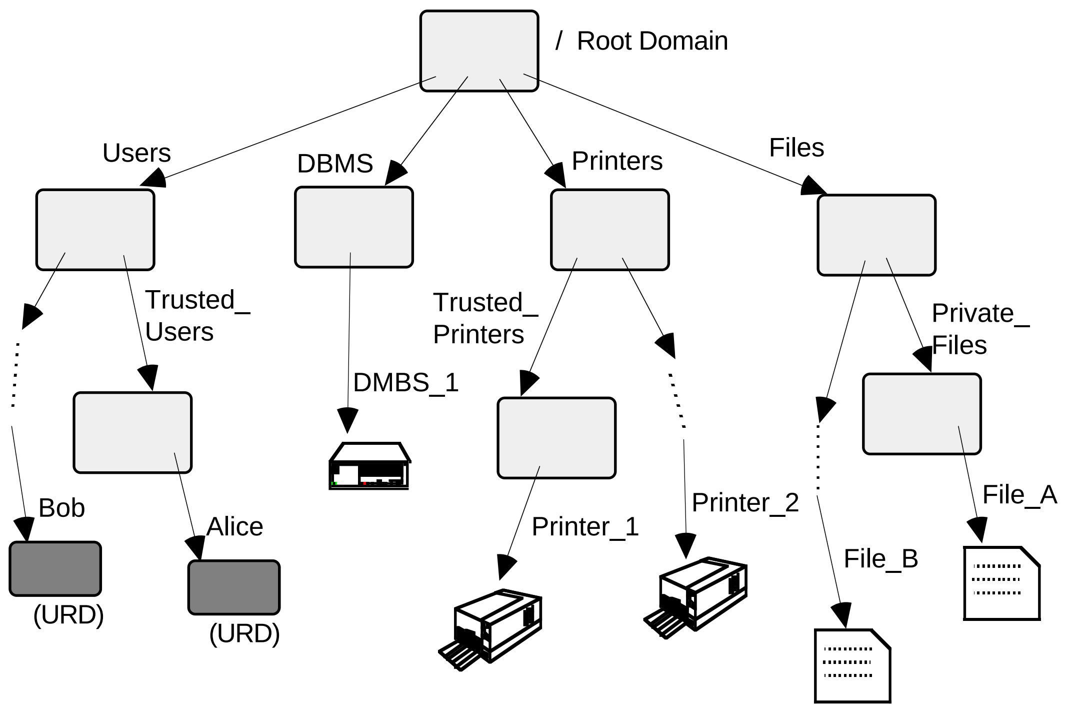


Figure 4.9 Partial View of an Hypothetical Domain Structure

	Subject Scope	Target Scope	Grantee Scope	Operation Set
AR1	*Users	*Files - *Private_Files		File:Read, Write
AR2	*Users	*Files -*Private_Files	*Printers + *DBMS	File:Read
AR3	*Users	*DBMS		DBMS:ALL
AR4	*Users	*Printers-*Trusted_Printers	*DBMS	Printer:Print
AR5	*Trusted_users	*Private_Files		File:Read, Write
AR6	*Trusted_users	*Private_Files	*Trusted_printers + *DBMS	File:Read
AR7	*Trusted_users	*Trusted_Printers	*DBMS	Printer:Print

Table 4.1 A Set of Extended Access Rules

Note, that the access rules in Table 4.1 define the objects that are trusted to be delegated certain access rights. The exact delegation sequence or the length of a delegation chain is not defined. *A*, for example, may choose to delegate its right to read *File_B* to *Printer_2* or to *DBMS_1* which can subsequently forward it to *Printer_2*.

According to the scheme described in (Lampson et al. 1992), all possible delegation sequences should be mentioned in the ACL of the end-point. For instance, the ACL associated with *File_B* should contain the following entries:

- G_U ***as*** *R1*
- *(*G_P ***as*** *R1)* ***for*** *(GU* ***as*** *R1)*
- *(*G_{DBMS} ***as*** *R1)* ***for*** *(GU* ***as*** *R1)*
- *(*G_P ***as*** *R1)* ***for*** *(*G_{DBMS} ***as*** *R1)* ***for*** *(*G_U ***as*** *R1)*

where G_U, G_P, G_{DBMS} are the sets of objects defined by the DSEs **Users*, **Printers* and **DBMS* respectively, and *R1* is a fictional role associated with the right to read *File_B.*

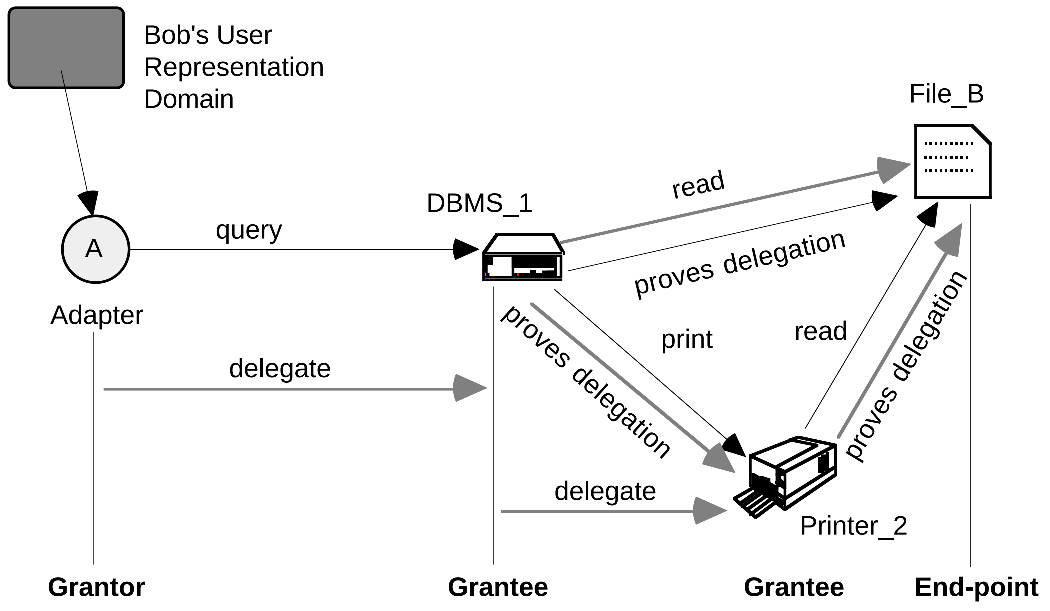


Figure 4.10 Cascaded Delegation Controlled by the Rules in Table 4.1

4.4 Chapter Summary

This chapter has discussed aspects of the delegation issue in distributed systems. Delegation of access rights is of crucial importance in distributed systems where resources are distributed over multiple hosts that are mutually suspicious. The main aspects of the delegation issue are:

- Access Control when delegation is involved
- Delegation policy specification
- Restriction of delegated access rights
- Revocation
- Proof of delegation

The last two aspects of the problem, namely revocation and proof of delegation involve the authentication system and are discussed in Chapter 7. This chapter reviewed research work concerning the first three aspects of the delegation problem. We used the calculus discussed in (Abadi et al. 1993) and the Taos delegation scheme as a case study in order to analyse the delegation problem.

We proposed a delegation policy specification scheme which enables security administrators to control the delegation of the rights they define for human users or system objects.

That scheme enables the administrators to determine:

- Which access rights can be delegated
- Who can delegate these access rights (possible grantors)
- To whom the access rights can be delegated (possible grantees)
- Special restrictions on delegation, such as time restrictions, maximum delegation period, etc.

Delegation is controlled by using Extended Access Rules which define a grantee scope. This scope is specified in terms of domain scope expressions and determines a group of objects that are trusted to be delegated the access rights given to the objects in the subject scope of the rule. The use of the extended access rules was demonstrated by a simple example which involves two-step cascaded delegations.

Chapter 5

Overview of the Security Architecture

This chapter outlines the security architecture. It shows the interaction between the main building components, describes the trust model on which the architecture is based and discusses related work.

5.1 Introduction

We take an approach to building the system in which distributed security is provided at two levels:

- In the address space of the *host manager*, which provides the basic authentication and access control functionality. In this way we avoid replication of security components and information amongst multiple application servers.
- In the address space of the application server. Security facilities, linked as a library for the application server, provide communication security and access control for individual operations. In this way we avoid the overhead that would be introduced by invocations between the address spaces of the host manager and the application server.

A *host manager* is a user-level process which is present on all hosts in the system. It supports two security agents: the *authentication agent* (AA) and the *access control agent* (ACA). All objects on a host are registered with their AA and ACA. The AA is responsible for verifying the identity and domain membership of remote objects as well as supporting delegation of access rights. It also identifies local objects to remote AAs. The ACA holds access rules applying to objects on its host and determines whether a rule exists that enables an object (possibly remote) to invoke operations on a local target object. The access control decisions made by the ACA are based on authentication, delegation and domain membership statements provided by the AA of the same host.

The security agents of the subject and the target establish a *secure channel* between these two objects which represents access control, membership and delegation information as well as cryptographic information used for secure communication between the two objects. Each channel is given a unique *channel identifier* (CHID). Once a secure channel has been established, the subject can invoke operations on the target provided that these operations are permitted by one or more access rules that have been associated with the established channel.

Each application server accommodates two *application server security facilities* which include the *Reference Monitor Facility* (RMF) and the *Cryptographic Facility* (CF). These facilities are linked into the server as a library and their operation is supported by the security agents. When the secure channel has been established, the application security facilities are given the relevant access control and cryptographic information which is used in order to exchange cryptographically protected messages and make access control decisions (the target RMF) on a per operation basis.

The operation of the security agents is supported by the authentication service, the policy service and the domain service. The authentication service is used to authenticate remote AAs and verify membership certificates as well as delegation tokens. The policy service maintains and provides the access rules applying to objects and the domain service is used to certify the domain membership of objects. The domain service is also used to determine the objects belonging to the scopes of the active access rules. A sketch of the security architecture is given in Figure 5.1.

The two security agents belong to the Trusted Computing Base (TCB) of the host. The application objects trust the security agents on their host. In addition, an object trusts a remote agent to serve an object on the same remote host. The system software and hardware are also trusted in the same way.

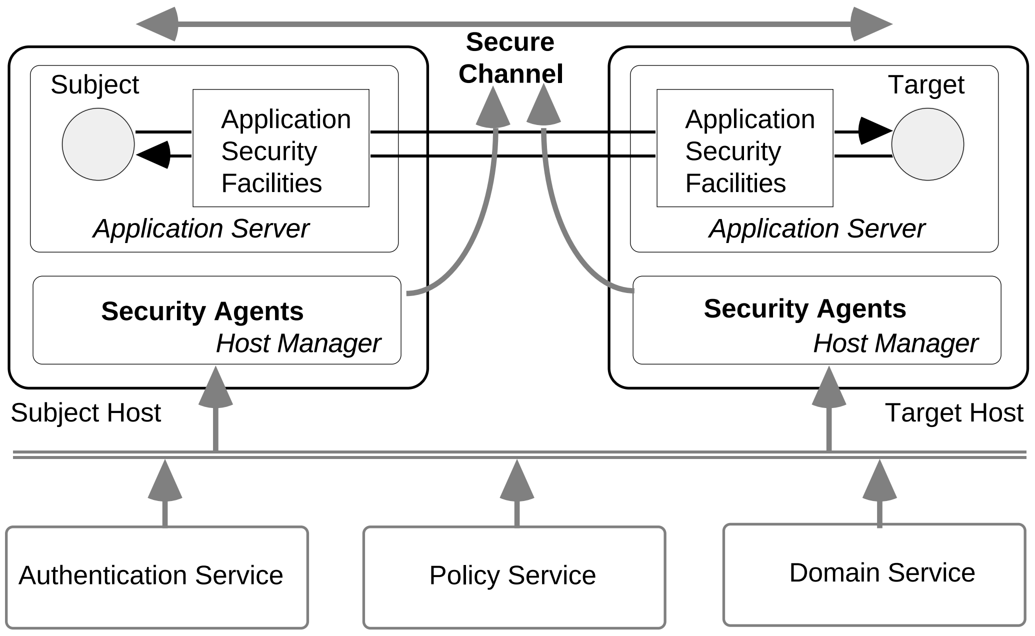


Figure 5.1 Sketch of the Security Architecture

A prototype of the policy distribution and access control mechanism has been implemented on the CORBA-compliant Orbix^{TM} platform. Another approach would be to provide security at the ORB level. This, however, requires access to the ORB which we were unable to achieve. The provision of security at the ORB level is investigated by the Object Management Group, Inc. (OMG) (Olson and Abrams 1990) and in (Deng et al. 1995).

5.2 Access Control

The access control system is based on the ACL paradigm. There are two main reasons that justify this choice. The first is the inherent weakness of the capability paradigm to efficiently cope with revocation of access rights. The second one is that a capability given to subjects cannot directly specify a set of targets as the domain structure may change during the life time of the capability resulting in a non-static target set.

The capability paradigm, however, exhibits an interesting advantage; it enables the target to make quick access control decisions as it does not have to search the ACL entries to determine whether access can be granted to the subject. The proposed system uses *pseudo-capabilities* as hints to enable the access control components of the target to minimise the access rule space that has to be searched. This is important because an access rule specifies subjects in terms of domains, and thus an access control decision involves verification of membership. The pseudo-capability scheme minimises the number of domain membership certificates that have to be used in order to determine the access rights of a subject.

Access rules are represented by policy objects which are maintained within policy servers. When a rule is activated by an administrator, it is disseminated to the ACAs that are responsible for the objects in the target scopes of the policy. A similar mechanism disseminates the rule to the ACAs supporting objects in the subject and grantee scopes of the policies. This enables subject and grantee ACAs to specify the pseudo-capabilities to be sent to the target ACA when a secure channel has to be established. In addition, it enables a subject to determine the access rights it possesses. As shown in Figure 5.2, the distribution of the rules involves the domain service which informs the ACAs about the OIDs of the rules applying to objects on their hosts. The rules themselves are given by the policy server to the ACAs that request them. This mechanism can cope with changes in the domain structure as well as with revocation of access rules. The effect is that each ACA is continuously aware of the rules applying to the objects on its host.

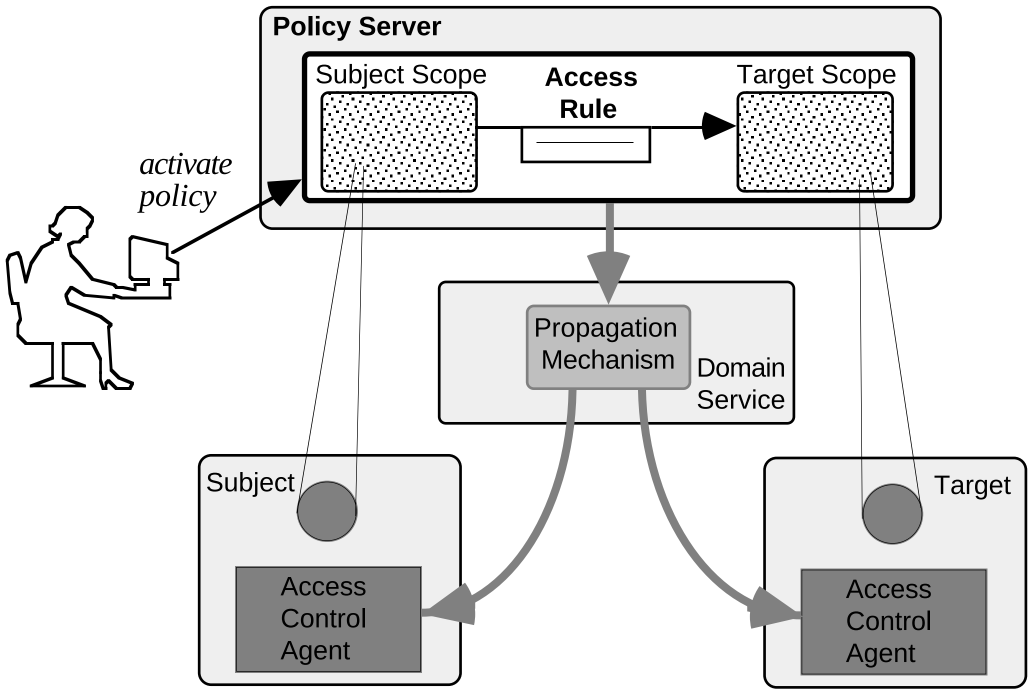


Figure 5.2 Dissemination of Access Rules (simplified)

The access control mechanism copes with delegation of access rights and enforces extended access rules. Since in many cases the delegation of access rights should be controlled by the application layer, an API enables subjects to determine the access rights they possess and restrict delegated access rights.

Access control is achieved in two phases. The first phase takes place when a secure channel is being established and the target ACA specifies the policies that enable the subject of the channel to access the target. This is done with the co-operation of the subject ACA which forwards the pseudo-capabilities that specify the access rights of the subject. Note that the pseudo-capabilities are not trusted and are only used as hints by the target ACA. The policies enabling the subject to invoke operations on the established channel are given to the reference monitor facilities (RMF) in the address space of the target. The selection of these policies by the target ACA is based on authentication and membership statements given by the target AA. Once the RMF has received these policies it can make access control decisions on a per invocation basis. In this way the size and complexity of the RMF can be confined to a minimal level. Further, the RMF does not have to invoke operations on the target ACA upon arrival of invocations on the established secure channel.

5.3 Authentication System

The access control mechanism is supported by an authentication system capable of verifying the identity of remote AAs and users. As the access control rules are specified in terms of domains, there is a need to support verification of the integrity of the membership statements made by the domain service. The access control system supports delegation of access rights, thus there is also a need to support verification of delegation. Authentication is based on symmetric cryptography to minimise the cryptographic overhead and a *relay* service enables the translation of membership statements and delegation tokens.

5.3.1 Registration of Principals

The authentication agents are the only components that interact with the AS. An authentication agent represents its host and all the other objects on it and thus only users and authentication agents are registered with the AS. This decreases the size and the update rate of the security database maintained by the AS (i.e. principal identities and keys). This database has to be updated upon the insertion or removal of a host or a user and is unaffected by the creation and deletion of other objects. It would be impractical to register every single object as the object population is expected to be high and change continuously.

Since only AA objects are known to the AS, authentication protocols can only be executed between AAs. The successful execution of an authentication protocol results in the establishment of a key shared by the two participants. This key is subsequently used for secure communication between the two AAs.

As the OID of an object contains the name of the object's host, a principal knows which AA supports that object. That AA is trusted to speak on behalf of the object. Thus, two mutually authenticated AAs can generate a key to be used for secure communication between two objects on their hosts.

5.3.2 Verification of Domain Membership

Authentication agents also undertake the verification of the domain membership of remote objects. This is based on membership statements made by the distributed domain service. Each domain server is trusted to certify the membership of objects in the domains it maintains. In general, the domain membership of an object has to be certified by multiple domain servers. A direct domain membership is certified by a signed certificate of the form *[B < A]* meaning that the object *B* is a direct member of the domain *A*. Membership certificates are re-encrypted with the secret key of the verifier by the relay service. The verifier (an AA) may combine a number of membership certificates in order to verify indirect membership. For example, two membership certificates of the form *[B < A]* and *[C < B]* imply that *[C << A]* where "<<" denotes indirect member.

The system, however, does not support verification of non-membership. This is because verification of indirect non-membership is not practical since all the possible membership paths have to be searched. This issue is also identified in (Abadi et al. 1993) from a different point of view. For example, let us assume that a verifier has to ensure that *K* is not a member of domain *C* in the structure given in Figure 5.3. In this case the whole domain structure underneath *C* has to be searched to ensure that there is no membership path between *C* and *K* (for instance, a possible path which is shown in Figure 5.3 is *[C, D, F, H, K]*). Such a search process could involve multiple domain servers leading to a high encryption and communication overhead.

5.3.3 Delegation

The authentication agents are also involved in the verification of delegation of access rights. Specifically, the AA of the grantor signs a *delegation token* (DT) which is forwarded to the grantee. This token contains the identities of the grantor and grantee, restrictions on the delegated access rights and other relevant information to be used by the end-point. The grantee can then present this token to the end-point to prove delegation. A delegation token is signed with the secret key of the grantor AA and is re-encrypted with the secret key of the end-point by the relay service. Cascaded delegation is also supported by this mechanism.

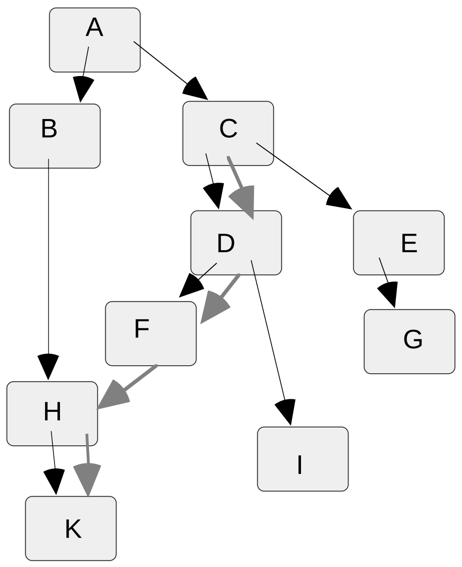


Figure 5.3 A Possible Membership Path

5.3.4 User Authentication

Users are permanently represented inside the system by their URDs (see Section 3.2.4). The access rights of a user are defined by the policies applying to her URD. When a user logs into the system, she interacts with the system via a number of adapter objects which should be members of her URD to inherit the necessary access rights. Thus, authentication of the user should result in the insertion of the adapter objects in her URD.

The AS registers users and secret values are used as proof of identity. These can be short enough to be remembered by human users or proper keys to be used with a cryptosystem. Normally users cannot remember long secret keys so such a secret value has to be stored on a device, such as a smart card, that can keep it secure. The smart card stores a key which is given to the card reader or the login workstation in order to perform cryptographic operations. In a more sophisticated scheme the smart card itself performs the required cryptographic operations without revealing the key to any other entity.

An adapter object can be included in the URD of the authenticated user. This inherits all the user’s access privileges so it can invoke operations on their behalf.

The fact that the adapter object has to prove membership in the URD provides sufficient ground to audit the identity of the user, should this be required.

5.4 Secure Channels

The notion of a secure channel has a crucial role in the proposed architecture. It is used as a means for correlating cryptographic, access control, membership and delegation information associated with a subject-target pair.

Note that the same object can be associated with different access rights in different cases. It may be the case that different levels of security are required between the same subject/target pair. In our systems these different situations are represented by different secure channels that integrate all the relevant information.

5.4.1 A Typical Scenario

This section presents a typical scenario where a secure channel is established between a subject and a target. Let us suppose that the object *ObjA* on *Host_A* (see Figure 5.4) intends to invoke operations on *ObjB* which resides on a remote host *Host_B* (Figure 5.5). *ObjB* stands for the OID of that object and we assume that it is known to *ObjA*. We also assume there is an access rule *AR1* permitting *ObjA* to invoke operations on *ObjB*. In a more complicated scenario, *ObjA* could be the grantee of another object that has delegated access rights to it. For the sake of comprehension, we do not discuss the delegation mechanism here.

In figure 5.5, the subject object requests its AA to establish a secure channel between itself and the target *ObjB*. At this point, *ObjA* can specify the type of secure channel it requires. For instance, *ObjA* may request a channel that provides data integrity but not data secrecy. Further, the subject may request that it intends to use a certain subset of its access rights. The subject AA and ACA in co-operation with the target AA and ACA establish a secure channel between *ObjA* and *ObjB*.

A *Resolved Channel Policy List* (RCPL) is determined by the target ACA and is associated with the channel. This list contains the access rules permitting the subject to access the target of the channel. It is based on the authenticated OID and domain membership of the subject, the access rules applying to the target as well as on the pseudo-capabilities given by the subject ACA. The RCPL is given to the RMF in the address space of the target which can then permit or reject access requests that are made on the established channel (Figure 5.5). In our example this list contains the rule *AR1*.

Membership verification is also performed by the two AAs in the framework of the established channel. The target ACA determines the domain membership of the subject (*ObjA*) that is required by the policies in the RCPL. Furthermore, the subject object may request verification of certain membership of the target. The verified membership of the subject and target is associated with the established secure channel.

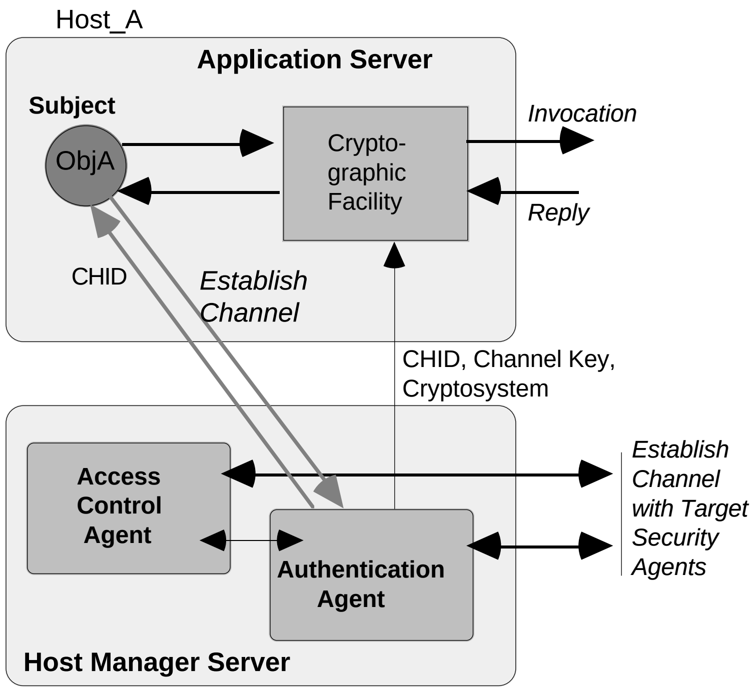


Figure 5.4 Subject Host and Secure Channel Establishment

Once the channel has been established, the cryptographic facilities in the address space of the two application objects are given the *channel key* and the cryptosystem type chosen by the AAs along with the identifier (CHID) of the established channel. The channel key can then be used to exchange messages between the two application objects.

The CHID of an established channel is given to the subject and is used to invoke operations. The target can retrieve the CHID of the channel through which an invocation has been made when delegation of access rights is taking place. In addition, the target may retrieve the channel attributes in order to make further access control decisions. An Application Programming Interface (API) is provided for this.

The CHID is used as an identifier of the channel key and it is transmitted in plain text along with the encrypted or signed invocation data. The subject CF decrypts and signs the invocation data using the channel key and the cryptosystem associated with the CHID specified within the invocation data.

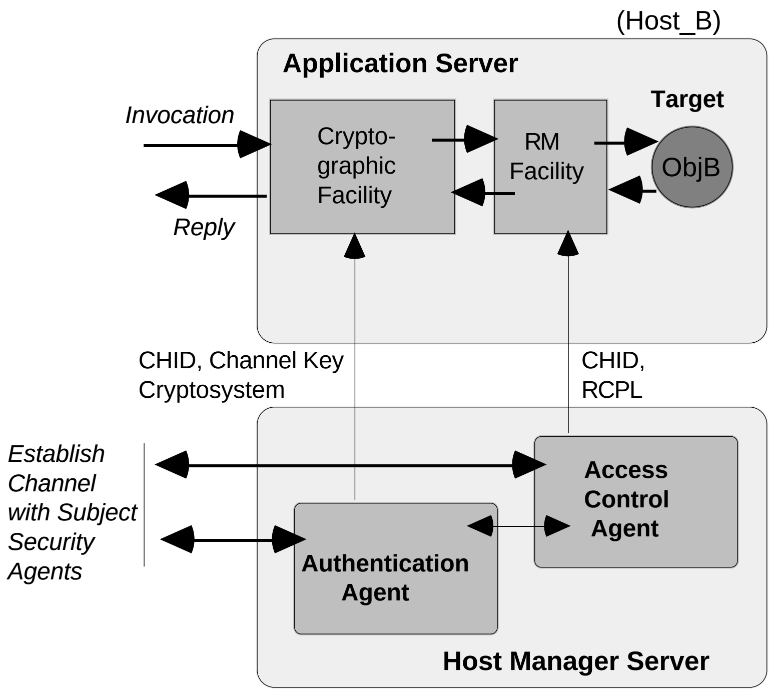


Figure 5.5 Target Host and Secure Channel Establishment

Subject AA	Subject CF	Target AA	Target ACA	Target RMF	Target CF
Subject OID, Membership		Subject OID, Membership	Subject OID, Membership		
Target OID, Membership		Target OID, Membership	Target OID, Membership		
			RCPL	RCPL	
Channel Key	Channel Key, Crypto-system	Channel Key	Security Attributes		Channel Key, Crypto-system

Table 5.1 Distribution of Information Related to a Secure Channel

The target CF uses the CHID that is transferred in plain text to determine the key and the cryptosystem it has to use to decrypt the incoming invocation request. When the message has been decrypted and the integrity and freshness checks are successful, the invocation data (containing the CHID) are passed to the RMF. This retrieves the operation name and the CHID. At this stage, the RMF believes that the invocation request has been made by the subject associated with the CHID. It therefore checks whether there is a rule in the RCPL associated with the CHID that permits the operation. The target and subject CFs swap roles when a reply has to be transferred back to the subject. Table 5.1 illustrates how the information associated with a channel is distributed among the main security components. This table does not include the delegation information that can be associated with a channel.

5.5 Related Work

This section discusses work that has been done in the area of secure distributed object systems.

(Deng et al. 1995) discuss a general security architecture for incorporation into CORBA based systems. The authors introduce the notion of an Object Security Controller (OSC) which acts as a reference monitor for target objects on its ORB node. The access control decisions are based on ACLs which are stored in a local repository named ACL Base (ACLB). Authentication is based on asymmetric cryptography and a certificate hierarchy is maintained by the “Naming Service” (NS). Each object is associated with a public/private key pair and a certificate. This applies to both persistent and dynamic objects. Objects employ the GSS_API (Linn 1993) in order to achieve authentication. The authors do not discuss the delegation issue though they identify the fact that a server object may act as a client of other objects. Furthermore, they do not discuss the ACL construction issue.

In comparison with our architecture, we can make the following points:

- A public/private key pair is generated for *each* object even if the object is expected to have a very short life cycle. This introduces cryptographic overhead as asymmetric key generation and cryptographic operations are very expensive. Note that each time a key pair is generated for a new object, a certificate has to be signed by the local Security Manager (SM) whose public key is certified by a CA. This implies that authentication of a dynamic object requires the verification of at least two certificates. Note that the fact each object possesses a key does not prevent the system hardware/software and the security components of a host from masquerading as any object on their host. In any case, the system hardware/software and the security components should be sufficiently trustworthy to serve the objects on their host.
- In our system this overhead is avoided because the AA is employed to identify local objects to remote authenticated AAs. Authentication between two remote AAs is performed only once even if multiple channels are established between objects on their hosts.
- The use of the GSS_API provides a degree of security transparency to the application level but this API does not deal with access control. The client object

has to request encryption of the invocation data (GSS_Seal) before making the invocation. The invoked object requests decryption of the incoming message (GSS_Unseal) and passes the decrypted request to the OSC which makes an access control decision based on the ACL associated with the target.

- In our system, all these operations are transparent to the application objects. An incoming invocation does not reach the target object if the subject does not possess the necessary access rights and an exception is returned by the RMF. Encryption/decryption operations are also transparent to the application object.

(Doorn et al. 1996) also discuss a security architecture for distributed object systems. The design of that system is similar to ours in that it employs an authentication agent on each host. That agent undertakes the authentication task on behalf of the local objects. That architecture also makes use of the notion of a *secure channel* but this is used for secure communication between two address spaces and does not incorporate any access control information. A single secure channel can be used for sending secure invocations to multiple targets in the address space of the same server. Authentication agents also generate channel keys for use by the communicating application address spaces. So, a single authentication between a client and a server enables secure invocations to multiple objects in the same server.

Access control is achieved by employing ACLs and capabilities. Capabilities are generated by the server of the target and are given to clients that are permitted to invoke operations on the target. A capability consists of an identifier, a key and an expiration time. The expiration time provides a means of revoking the capability. A capability is always transferred under the cryptographic protection of the channel key shared between the client and the server. The client signs (using a hash function) an invocation with the capability key and the shared channel key so that the target can check the source of an invocation. If secrecy is required, the invocation data and their signature are encrypted under the channel key.

The proposed architecture has been implemented using Modula-3 but it does not support delegation of access rights and the access control mechanism is based on the assumption that each address space is running on behalf of a user and thus possesses her access rights. The identity of an address space consists of a user name and a host name. This may be restrictive in distributed environments in which a user may choose to operate with different sets of access rights or different address spaces are trusted to different degrees.

The design of that system is partially based on the Taos Operating System which employs a more elaborate authentication agent but does not use capabilities (Wobber et al. 1993). According to (Doorn et al. 1996), the delegation mechanism of the Taos OS, which has been discussed in Chapter 4, can be incorporated in the above-mentioned system.

An interesting discussion of security issues with regard to the CORBA specification is given in (OMG 1995). This proposes a general security framework for incorporation in the CORBA specification and it does not provide specific solutions to issues like authentication, delegation and access control. It emphasises the need for supporting both security unaware and aware application objects and proposes the insertion of security services in the ORB. These services are hidden from security unaware objects but objects that want to make further access control decisions or control the delegation of access rights can use an API to access and manipulate security-related information. There are some similarities between this specification and our architecture. Specifically, the security services in our architecture are transparent to the application which, nevertheless, can use an API to access the information associated with a secure channel. This enables objects to restrict the delegated access rights and make further access control decisions if this is required.

5.7 Chapter Summary

This chapter has provided an overview of the security architecture which is described in detail in the following chapters. The main components of this architecture are:

- an Authentication Service which is based on symmetric cryptography to reduce the cryptographic overhead. A relay service is employed to translate membership certificates and delegation tokens.
- a Policy Service that maintains and distributes access control policy objects (access rules).
- a Domain Service that is provided by a number of servers which may be trusted up to different extents. They provide membership certificates that are used to verify domain membership of objects.
- Security agents that are employed on a per host basis to serve all the objects on their hosts and undertake the creation of secure channels between objects. Their use also minimises the security components and information that have to be replicated amongst the application servers.
- Application server security facilities that include the reference monitor and the cryptographic facilities. These are linked as libraries or the application servers and enable the secure invocation of operations when a secure channel has been established.

A secure channel represents access control, delegation and membership information as well as cryptographic information that enables secure communication between objects and the enforcement of the appropriate access control polices. Each channel is given a unique channel identifier (CHID) which can be used as a reference to the information associated with the it. An API enables the application layer to control the creation of secure channels and access information associated with them.

Chapter 6

Access Control

This chapter deals with the mechanisms employed to disseminate and enforce access control rules. It describes secure channel establishment from the points of view of access control and delegation. It also discusses an Application Programming Interface (API) that enables the application level to access the information associated with a secure channel and control the delegation of access rights.

6.1 Introduction

As discussed in Chapter 3, administrators create policy objects on policy servers. There is a need to distribute the active access control rules (low-level positive policies) to the security components that control access to objects. In addition, there should be a means of retracting policies when they are disabled. As the domain structure is not static and the set of objects in the scope of a policy changes, we need a mechanism that can dynamically determine the objects in the scopes of the active policies. Such a mechanism is proposed in (Twidle 1993) and (Twidle and Moffett 1992) but it can only handle simple Domain Scope Expressions (DSEs). For instance, it can cope with the expression **DomA* - **DomB* but not with the expression **DomA - (*DomB* ^ **DomC)*. Section 6.2 describes the principle of a policy dissemination mechanism that can handle any DSE. This mechanism enables the ACAs to determine and obtain the policies applying to objects on their hosts. More technical and implementation details of the proposed mechanism are given in the Appendix.

Access control decisions are made in two phases. The first takes place when a secure channel is being established between a subject and a target, and the target ACA determines the access rules enabling the subject to invoke operations on the target. The list of these policies is given to the reference monitor facility in the address space of the target server in order to make access control decisions on a per invocation basis (second phase).

The access control mechanism deals with cascaded delegation of access rights. The initial grantor, as well as the intermediate grantors, can restrict the access rights they delegate. An API enables the security-aware applications to control the use and delegation of their access rights, and make further application dependent access control decisions.

6.2 Policy Distribution

6.2.1 Dynamic Policy Scopes

The scopes of a policy are defined using Domain Scope Expressions (DSEs) (Section 3.2.2). The object set defined by a DSE depends on the domain structure to which the expression is applied. It is possible that the domain structure changes, new objects are inserted in some domains or others are removed during the life cycle of a policy. This implies that the set of objects to which a policy applies is not static and therefore a single evaluation of it when the policy is activated is not sufficient. There is a need to provide a means to re-evaluate this set every time it is affected by a change of the domain structure.

Consider for example the expression **DomA* ^ **DomB*. Let us assume that this defines the target scope of an access rule *AR1* which is active while the domain structure illustrated in Figure 6.1 changes. When that rule is activated, the domain structure has the form (a) which means that the rule applies to the targets *ObjY* and *ObjZ*. In state (b), *DomD* ceases to be a member of *DomB* and therefore *ObjZ* is not in the target scope of the rule. In state (c), *ObjY* is removed and *ObjZ* becomes a member of *DomB* . In this case, AR1 applies to *ObjZ* .

As policies are enforced by the ACAs, a mechanism is required that enables these agents to determine which policies apply to objects on their hosts each time the domain structure changes. One possible solution to this would be to report any change of the domain structure to the policy server which subsequently re-evaluates the scopes of the active policies and informs the ACAs of the affected objects. This, however, implies that policy scope evaluation is achieved in a centralised way and requires that the policy server knows the affected part of the domain structure which is inefficient for large domain structures that change frequently.

This section presents a policy dissemination mechanism that updates the state of the relevant ACAs each time a rule is activated, retracted or the domain structure changes and affects the policy scopes. In particular, the mechanism employs the domain service in order to inform the ACAs about the access rules applying to the objects on their hosts.

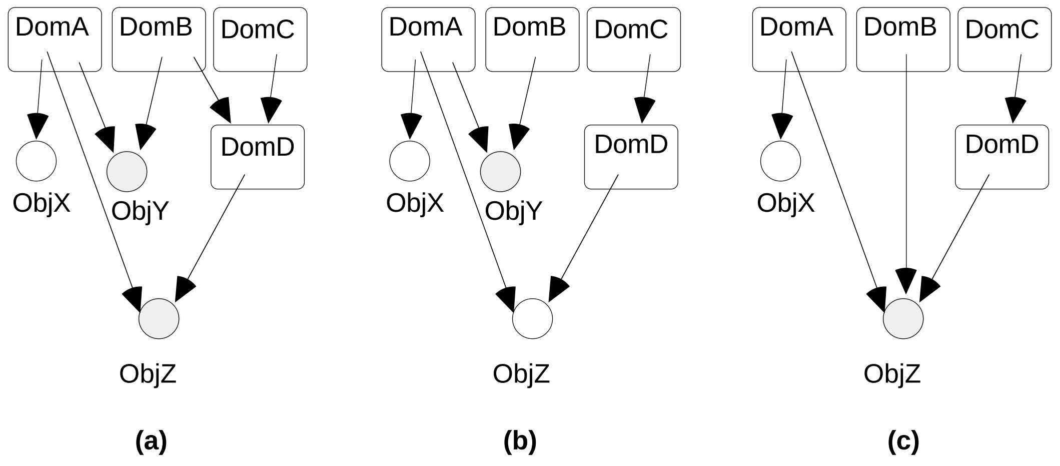


Figure 6.1 Three States of a Domain Structure

The policy dissemination is performed in two phases (see Figure 6.2). In the first phase, special tokens are propagated down the domain hierarchy when a rule is activated. These tokens eventually reach the ACAs of objects that are *possibly* in the target scope of the rule. The information carried within these tokens enables the ACAs to determine whether targets on their hosts are in the scope of the rule. In this phase, the rule itself is not propagated. In the second phase, the ACAs that support objects in the target scope of the rule request it from the policy server that triggered the propagation in the first phase.

6.2.2 Propagation of Policy Scope Evaluation Tokens (PETs)

In the first phase of the policy distribution process, *Policy scope Evaluation Tokens* (PETs) are propagated down the domain hierarchy starting from the domain objects mentioned in the target DSE. We provide an outline this propagation mechanism; a detailed description can be found in the Appendix.

A policy scope evaluation token (PET) carries three pieces of information:

- The OID of the policy object
- The target DSE
- Propagation Control Flags

Each *Propagation Control Flag* (PCF) is originally generated by an *object operator subexpression* in the target DSE, for instance @A, *B, *4C or {D}. The subexpression that generates a PCF is referred to as the *generator* of the PCF. Note that for a given domain structure an object operator subexpression defines a set of objects. A PCF is propagated down to all the ACAs of the objects that are in the set defined by its generator. The propagation depth is controlled by the object operator (@, * *or* { })which is carried within the PCF.

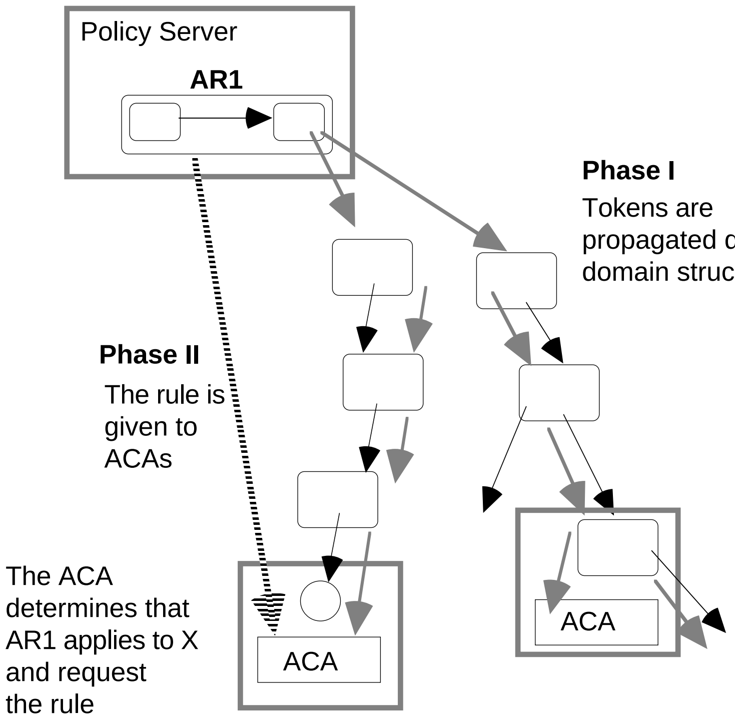

Policy Server
AR1
Phase I
Tokens are propagated down the domain structure
Phase II
The rule is given to ACAs
The ACA determines that AR1 applies to X and request the rule
ACA
ACA

Figure 6.2 Two-phase Policy Dissemination

As PCFs always “travel” within PETs which contain the target DSE, the recipient of a flag can associate it with the target DSE to determine whether the object, on behalf of which the flag has been received, is in the target scope. For instance, consider the PCF propagation illustrated in Figure 6.3. Initially, the policy server maintaining the policy object *AR1*, generates two PCFs: *PCF1* and *PCF2*. These correspond to the generators **DomA* and **DomB* respectively and are propagated within *PET1* and *PET2* to the domains *DomA* and *DomB*. These domains hold the PETs they receive and further propagate copies of the PCFs (always inside PETs) to their subdomains and to the ACAs that are responsible for their non-domain objects. The propagation continues until the flags reach the ACAs of all the objects in the sets defined by the flag generators. For example, the ACA of *A* receives *PCF1* as *A* is in the set defined by **DomA* (the generator of *PCF1*). The ACA can now determine that *A* is in the target scope of *AR1*.

The ACA of object *C*, on the other hand, receives both flags and determines that *C* is in both the sets defined by the generators **DomA* and **DomB* and thus is not in the set defined by the DSE **DomA* - **DomB.*

In Figure 6.3, each domain holds a PET containing the flags it has received as a result of the activation of *AR1* (these PETs are shown inside squares). In fact, each domain holds a PET for each policy from which it has received at least one PCF. Because policies may control access to domain objects, the PETs representing a policy that applies to a domain, are also given to the ACA of the domain server. This is illustrated in Figure 6.3 for the domain *DomA*.

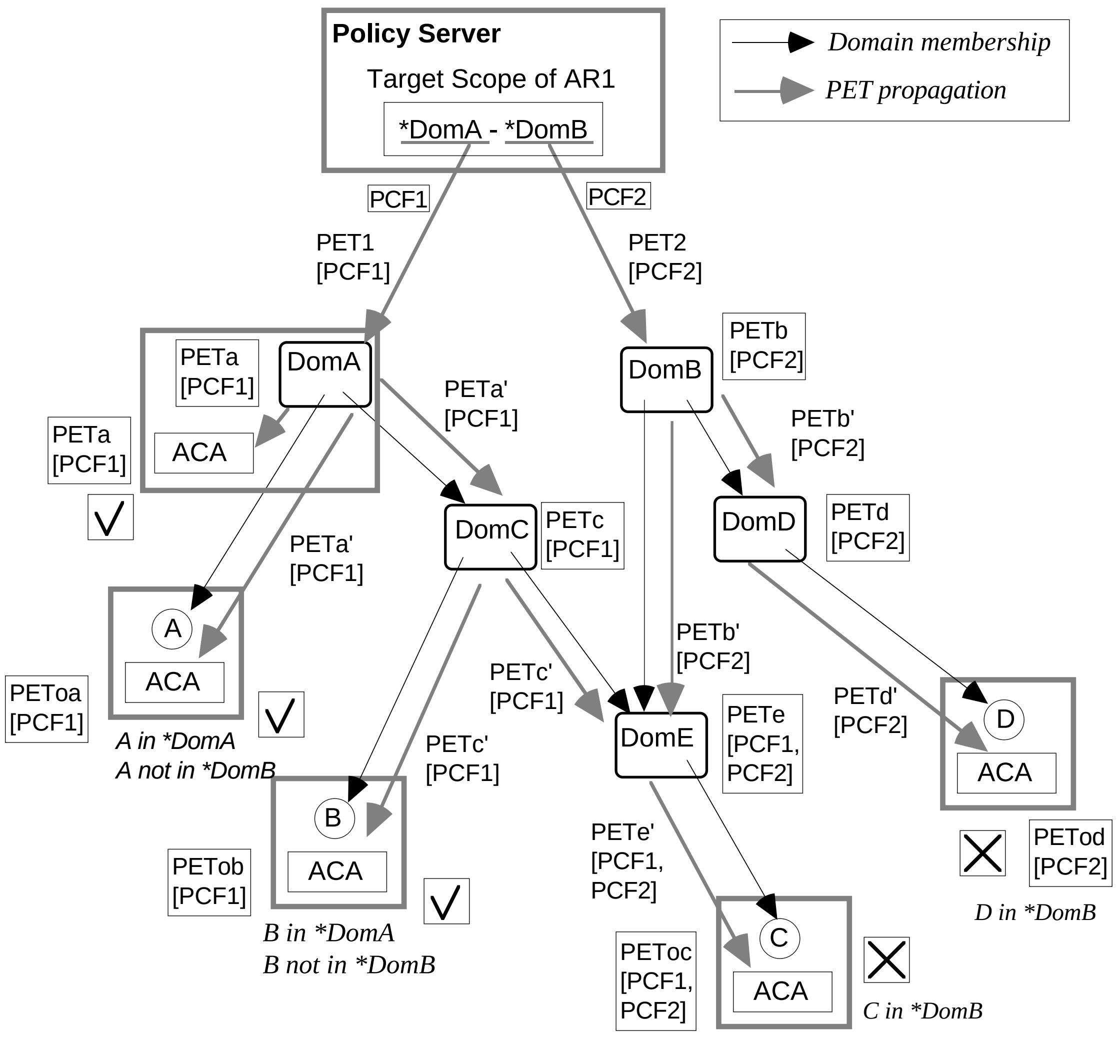


Figure 6.3 An Example of PCF Propagation

6.2.3 Dynamic Domain Structures

The propagation mechanism has been designed to cope with domain structures that change when policies are active. A simple change the mechanism can encounter is the insertion of a non-domain object in a domain. In this case, the domain propagates the appropriate PCFs to the ACA of the new member. For instance, if an object *E* becomes a new member of *DomC* in Figure 6.3, this domain propagates the *PETc'* (which contains *PCF1*) to the ACA of *E*.

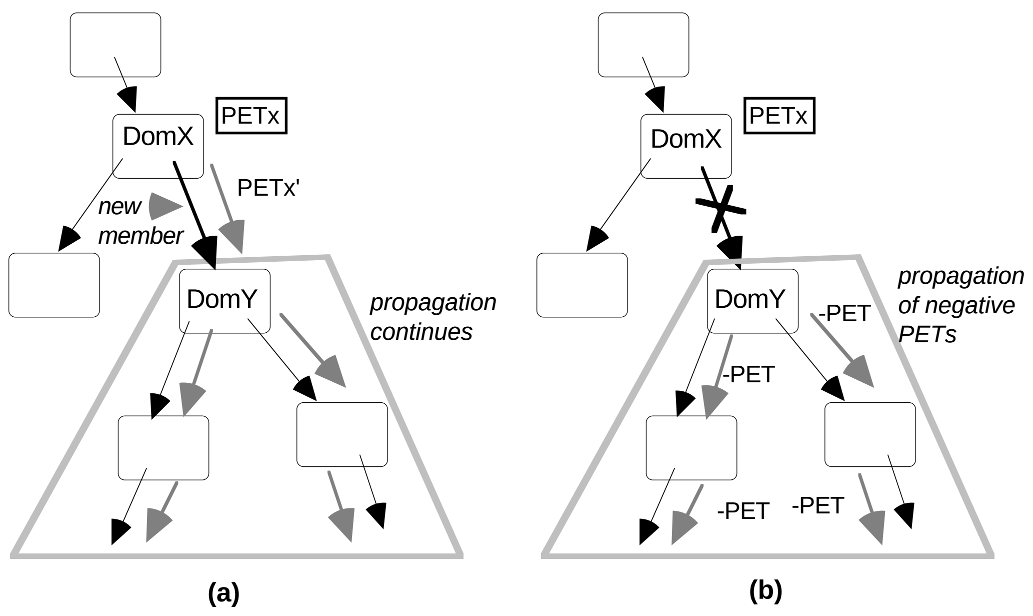


Figure 6.4(a) Example for PET Propagation When Insertion of a Subdomain Takes Place
Figure 6.4(b) Example for PET Propagation When Removal of a Subdomain Takes Place

If a non-domain object is removed from a domain, the responsible ACA deletes the PCFs that have been received from that domain and checks the remaining PCFs (if any) against the target expression to determine whether the set of policies applying to the object has changed.

A more complex case is the insertion of a domain into another domain. In this case the parent domain propagates the appropriate PCFs to its new subdomain which further propagates them to its subdomains, etc. This illustrated in Figure 6.4(a) where *DomY* has become a member of *DomX*. For a more concrete example, we assume that *DomC* becomes a member of *DomB* in Figure 6.3. In this case, *DomC* receives a PET containing *PCF2*. This flag eventually reaches *DomE* and the ACAs of *B* and *C*. The

ACA of *B* can now determine that *B* is no longer in the target scope of *AR1* and it stops enforcing that rule. Note that this propagation does not affect the presence of *DomE* and *C* in the target scope of the rule since they have already inherited *PCF2*.

Finally, the most complex case is the removal of a subdomain from a domain. In this case the subdomain deletes the PCFs it had received from its former parent. Furthermore, it propagates *negative* PETs containing the PCFs that have to be removed from its members. A negative PET has a special field indicating that the flags it contains have been revoked and therefore they should be deleted by the receivers of the PET. Figure 6.4(b) illustrates the propagation of negative PETs when *DomY* is removed from *DomX*. In figure 6.3, if *DomE* is removed from *DomB*, *PCF2* is removed from the PET stored in *DomE*. Furthermore, *DomE* propagates a negative PET containing *PCF2* to the ACA of *C*. This results in the removal of *PCF2* from the PET held by the ACA which subsequently determines that *C* is in the target scope of *AR1* and requests this rule from the policy server.

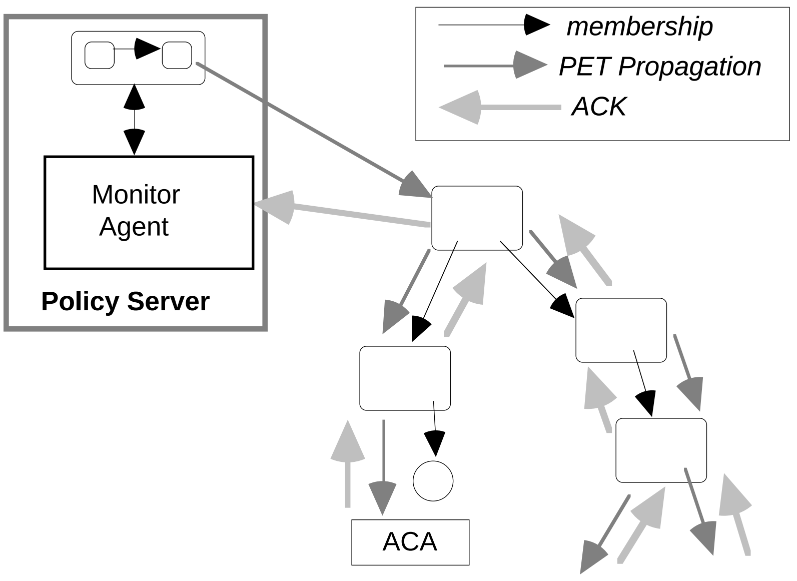


Figure 6.5 Propagation of Acknowledgements

6.2.4 Monitoring the Propagation Process

As the domain service is distributed, it is possible that the propagation process is interrupted by a communication problem or domain server failure. For this reason it is essential to provide a mechanism that enables the security administrator to monitor the propagation status of the active policies. This is particularly important when a subdomain is removed and negative PETs are propagated down the hierarchy underneath the removed subdomain (see Section 6.2.3). If the propagation of negative PETs is interrupted, it is possible that some ACAs keep permitting access to target using a policy that does not apply to them any more.

The monitoring process consists of two phases. The first phase provides information about the outcome of the propagation triggered by the policy server (see Figure 6.5). Each object that receives a PET, sends back an acknowledgement when further propagation is successful. Thus, the *monitor agent* within the policy server receives an acknowledgement when the propagation has completed. If a problem occurs, the problem is reported back to the parent domain until it reaches the policy server.

The second phase reports problems occurring when propagation is triggered by changes in the domain structure (see Figure 6.5). The domain that triggers such a propagation reports problems back to the policy server maintaining the policy whose distribution has been affected by propagation problems.

6.2.5 Policy Hand Over and Retraction

Once an ACA establishes that an access control policy applies to at least one application object on its host, it requests that policy from the policy server (see Figure 6.7). The identity of the policy server maintaining a policy is implied by the policy OID as it contains the names of the server and its host. Note that the ACA has to verify the source of the policies it receives in order to avoid enforcing policies that have been specified by principals other than the trusted policy server. So the ACAs have to establish secure channels with the policy servers. These channels should normally provide message authenticity but they can also provide secrecy to prevent the unauthorised disclosure of the policies. Also, if policy secrecy is required, the policy server should check whether the ACA is entitled to receive a policy and thus the server should be provided with the necessary membership certificates proving that a target object supported by the ACA is in the scope of the policy.

The distribution manager of the policy server checks whether the propagation of the corresponding PETs was successful and hands over the policy. In addition, it maintains the identities of the ACAs that have been given a policy so that it can inform them when the policy has to be retracted. Note that, in general, the distribution manager does not hand over a policy if the propagation process has not terminated successfully.

Consider, for instance, *DomE,* in Figure 6.3, which receives two PETs containing the same policy OID. These are merged into one (*PETe*) that contains both PCFs generated by the target DSE of *AR1*. In Figure 6.3, *DomE* propagates one PET (*PETe'*) containing *PCF1* and *PCF2*. In a real system, however, it is possible that these two flags are propagated in two steps since *PETb'* and *PETc'* may not reach *DomE* simultaneously. This may give rise to the following problem: if *PCF1* arrives at the ACA of *C* before *PCF2*, there is a time interval during which the ACA believes that *C* is in the target scope of *AR1* and therefore access could be incorrectly granted if the policy has already been handed over.

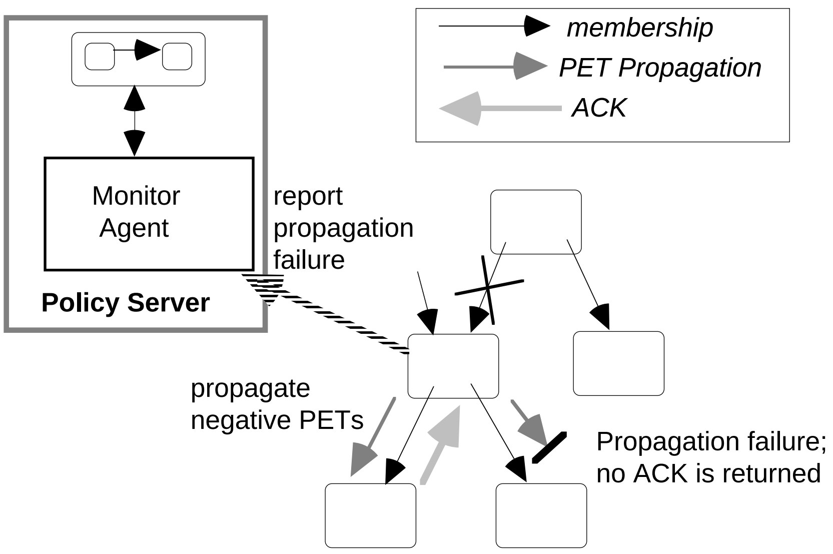


Figure 6.6 Reporting a Propagation Failure

A policy server maintains the identities of the ACAs that have been given a policy so that it can inform them when the policy has to be retracted (Figure 6.7). The policy retraction, however, is vulnerable to denial of service attacks and communication problems. As an additional protection, each ACA can periodically request a policy validity confirmation. If an ACA fails to get a confirmation, it ceases to enforce the policy until it receives a confirmation. Again, the ACA should verify the integrity of these confirmations.

In addition to the direct policy retraction, the policy server propagates negative PETs (similar to those propagated when a subdomain is removed from its parent domain) in order to delete the PETs stored by the domain service.

6.3 Propagation of Pseudo-Capabilities

The propagation of the PETs generated by the target expressions of the access control policies ensures that the ACAs enforce the policies applying to the targets on their hosts. There are reasons, however, to apply a similar propagation mechanism in the subject and grantee scopes of the policies in order to allow the ACAs to determine the policies applying to subjects and grantees on their hosts. The main reasons for this are:

- The subjects and grantees can get the policies enabling them to invoke access rights and determine their access rights.
- The subject and grantee ACAs can provide the target ACA with a list of policy OIDs that can be used as hints in order to minimise the policy space that has to be searched to specify the policies applying to a secure channel.

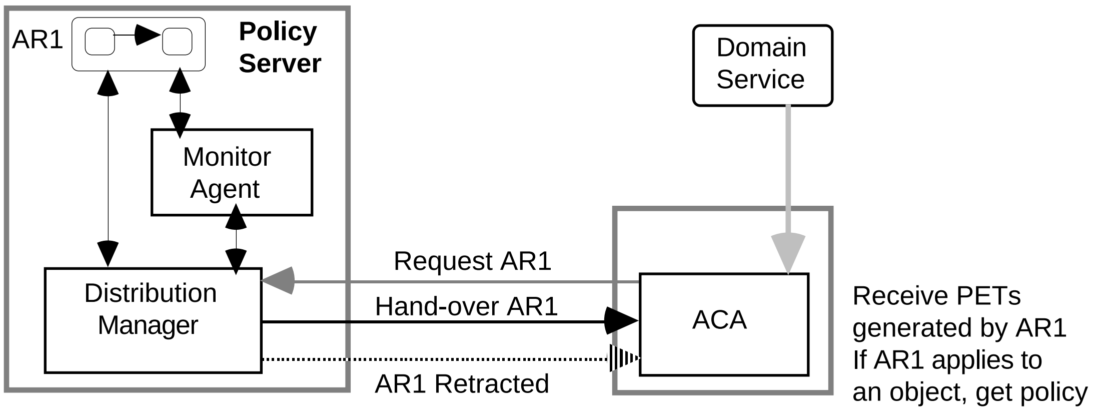


Figure 6.7 Policy Hand Over and Retraction

Because the policy OIDs that are propagated in the subject scope define the access rights of the subjects and are given to the target ACAs to be used as hints, we call them *pseudo-capabilities*. Similarly, the policy OIDs propagated in the grantee scope are called *grantee pseudo-capabilities*. A pseudo-capability indicates the access rights given to the subject by the corresponding policy. A grantee pseudo-capability indicates that the object receiving it can act as a grantee of the objects in the subject scope of the corresponding rule. A detailed description of their use is given in Section 6.4.

6.3.1 Determination of the Pseudo-Capability Certificate Sets

The propagation mechanism in the subject and grantee scope is similar to the propagation mechanism in the target scope. That is, PCFs generated by the subject and grantee DSEs are propagated inside PETs until they arrive at the ACAs that are responsible for the objects in the subject and grantee scopes of the policy. These PCFs, however, carry an additional piece of information. They record their propagation path so that the ACAs can specify the domain membership they have to prove in order to enable the use of the policies applying to subjects and targets. This is illustrated in Figure 6.8 where propagation of PCFs takes place in the subject scope of the rule *AR1*. All the PCFs are propagated within PETs containing the policy OID (*AR1*). The ACA of *X* can determine that *X* is in the subject scope of *AR1* and therefore it can make use of the access rights defined by that rule. *AR1* is a pseudo-capability for *X* and it can be presented to a target

ACA. The target ACA, however, has to ensure that *X* is in the subject scope of *AR1*. Therefore the subject ACA has to present domain membership certificates stating that *[X << DomA]* and *[X << DomB]*. The set of these certificates is called the *Capability Certificate Set* (CCS).

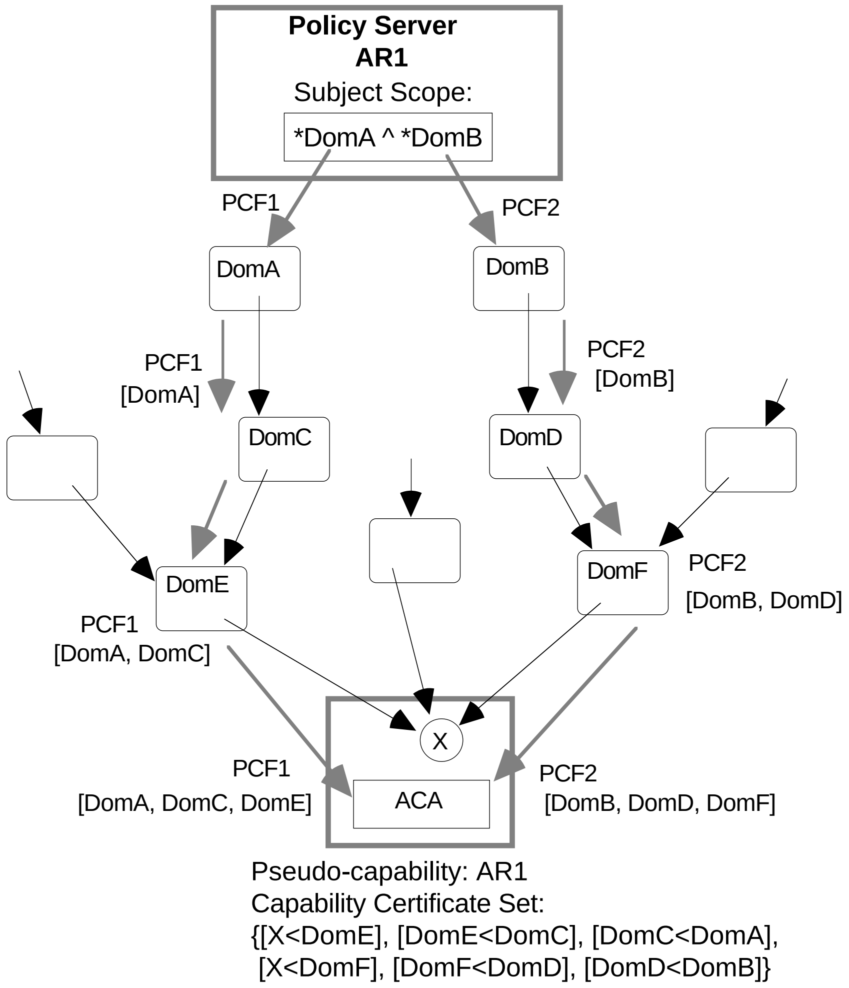


Figure 6.8 Propagation in the Subject Scope and Determination of the Capability Certificate Set (CCS)

In general, it is not easy to determine the CCS associated with a pseudo-capability because the ACAs do not hold the parent domain hierarchies of the objects on their nodes. One way to determine this set in the example shown in Figure 6.8 is to use the domain service to identify the paths from *DomA* and *DomB* down to *X*. This, however, is not efficient since the domain service is distributed and such a query may involve multiple domain services.

The PCFs that record their propagation paths provide a more efficient way to determine the CCS. As it can been seen in Figure 6.8, these paths define the members of the CCS that supports the pseudo-capability *AR1*. The ACA, therefore, knows the members of the CCS once it has received the PCFs propagated in the subject scope of *AR1*. The same technique is used when PCFs are propagated in the grantee scope of an extended access rule so that the ACAs can identify the members in the *grantee capability certificate set* (GCCS).

6.3.2 Special Cases

In some cases, PCFs generated by the same generator may follow different paths before they reach the same ACA. This is because domains may overlap and thus a subject or a grantee may inherit copies of the same PCF from multiple direct or indirect parent domains. In such cases, the pseudo-capability can be supported by more than one different CCS. The ACA supports the pseudo-capability by constructing the smallest CCS.

When the subject scope of a rule is specified by the DSE “ANY”, propagation of PCFs in that scope can be avoided. There is no need for the subject ACA to prove any domain membership as any object possesses the access rights defined by that rule. The same applies to grantee scopes defined by the DSE “ANY”.

Although the propagation mechanism supports DSEs containing the set difference operator (-), these expressions cannot be used to specify subject and grantee scopes as the domain membership verification system does not support verification of non-membership (Section 5.3.2).

6.4 Secure Channel Access Control Information

A secure channel represents cryptographic, domain membership and access control information. The channel access control information is used by the target side in order to make access control decisions on a per invocation basis. This section describes how this information is used during the establishment of a secure channel which involves both the subject and target ACAs.

6.4.1 Access Control Information Held by the ACAs

In general, a single object can act as a subject, grantor or grantee as well as a target. Therefore, an ACA may receive PETs propagated for an object in the subject, grantee and target scopes of various policies. The ACA holds the received PETs and employs a simple scope evaluation algorithm (which is given in the Appendix) in order to determine the policies applying to objects on its host.

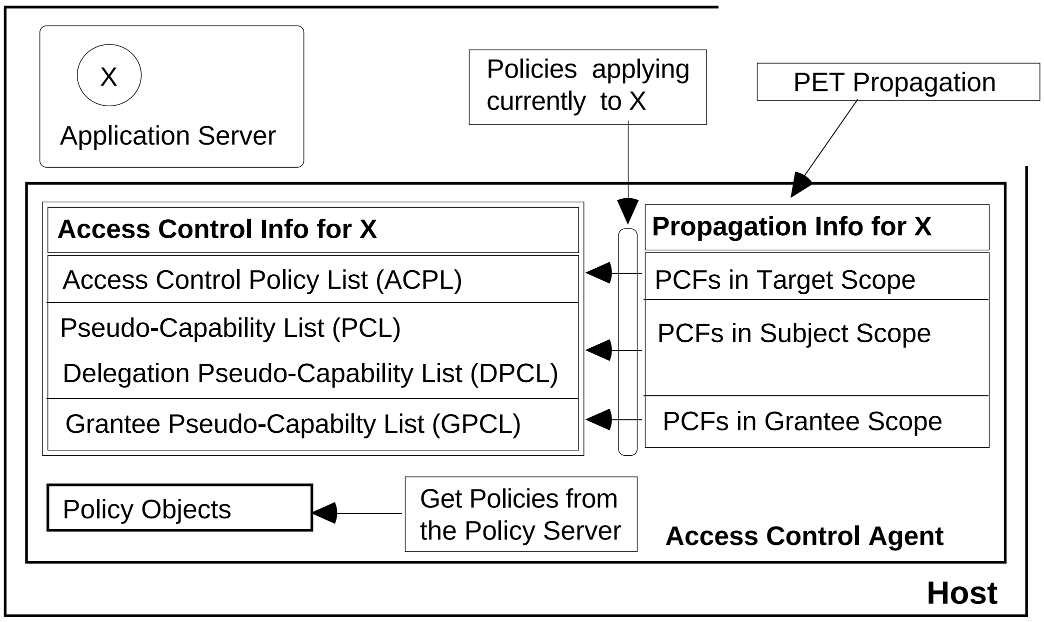

X
Application Server
Policies applying currently to X
PET Propagation
Access Control Info for X
Access Control Policy List (ACPL)
Pseudo-Capability List (PCL)
Delegation Pseudo-Capability List (DPCL)
Grantee Pseudo-Capabilty List (GPCL)
Propagation Info for X
PCFs in Target Scope
PCFs in Subject Scope
PCFs in Grantee Scope
Policy Objects
Get Policies from the Policy Server
Access Control Agent
Host

Figure 6.9 Access Control Information Held by an ACA

As it can been seen in Figure 6.9, an ACA maintains a structure holding lists that contain the OIDs of the policies applying to each object on its host. Specifically, there are four lists:

- **Access Control Policy List** (ACPL). This contains the OIDs of the policies applying to the object as a *target*.
- **Pseudo-Capability List** (PCL). This contains the OIDs of the policies applying to the object as a *subject*. In other words, this list determines the policies specifying access rights for the object.
- **Delegation Pseudo-Capability List** (DPCL). Contains the OIDs of the policies applying to the object as a *grantor*. This is actually a subset of the PCL and it is not implemented necessarily as a separate list.
- **Grantee Pseudo-Capability List** (GPCL). Contains the OIDs of the policies applying to the object as a grantee. It determines the policies permitting the object to act as a grantee.

These lists are updated as necessary every time a new PET (either positive or negative) is received. Note that a policy may apply to many objects on the same host so there is a separate structure maintaining a single policy object currently applying to objects on the host. A policy is requested from its policy server once the ACA detects that it applies to at least one object (Section 6.2.5)

6.4.2 Determining the Privileges of the Subject

The policies in the ACPL specify possible subjects and their access rights. The fact that the subjects are specified in terms of domains makes the access control decision more complex than it is in conventional access control systems since the target ACA needs evidence that the subject requesting access is a member of particular domains. The domain membership that has to be proven depends on the policies applying to both the subject and target. The subject cannot identify these policies and therefore it cannot determine the precise set of membership certificates required to access the target. In general, the parent domain hierarchy of the subject is complex and it is not practical to get membership certificates proving any possible domain membership. But even if we assume that the subject collects all those certificates, the access control facilities of the target have to search the whole ACPL in order to determine the policies permitting the subject to invoke operations on the target. This is time consuming if the parent domain hierarchy of the subject is large.

The proposed system avoids this problem by using the pseudo-capabilities that the subject ACA receives and stores in the PCL. These pseudo-capabilities are given to the target ACA in order to determine the policies applying to both the subject and target. This reduces the number of certificates that have to be collected by the subject and the number of policies that have to be checked by the target ACA. A similar technique is used when delegation takes place. We first describe how the mechanism works when no delegation is involved.

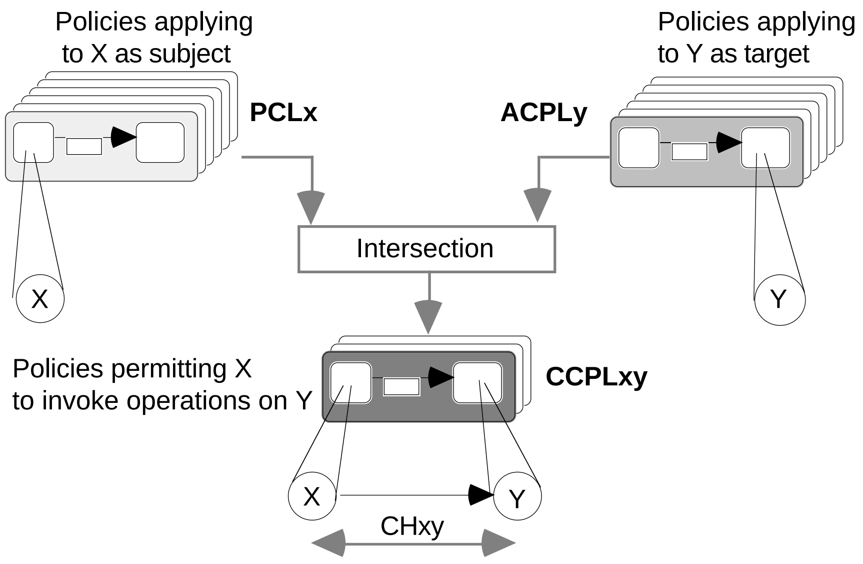

Policies applying to X as subject
PCLx
ACPLy
Policies applying to Y as target
Intersection
X
Y
Policies permitting X to invoke operations on Y
CCPLxy
X
Y
CHxy

Figure 6.10 Construction of the Candidate Channel Policy List

Once the subject requests the establishment of a secure channel between itself and a remote target, the subject ACA sends the PCL associated with the subject to the target ACA (see Figure 6.11). This, compares the PCL with the ACPL of the target and produces a list that contains the policy OIDs that are in both the PCL and the ACPL. This is the intersection of the PCL and the ACPL, and it is referred to as the *Candidate Channel Policy List* (CCPL), since it specifies the policies that are *possibly* those that will enable the subject to invoke operations when the channel has been established. Figure 6.10 illustrates the rationale behind the construction of the CCPL. Note that the target ACA cannot base its decisions on the received PCL as there is no guarantee that these policies apply to the subject. It needs evidence that the subject is indeed in the subject scopes of the policies in the CCPL.

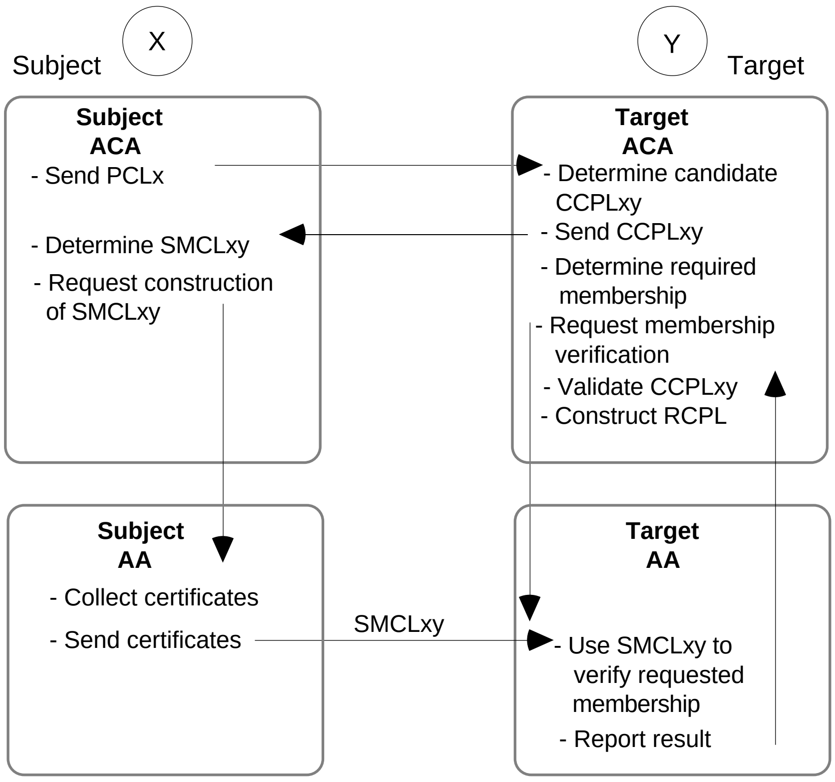


Figure 6.11 Construction of the RCPL

For this reason, the CCPL constructed is sent back to the subject ACA which determines the membership certificates that prove the membership required to support the policies in the CCPL. The subject ACA determines these certificates along the lines described in 6.3.1; that is, it uses the CCSs associated with the policies in the CCPL. This certificate list is given to the subject authentication agent which collects the specified certificates and forwards them to the target AA. The list of these certificates is called *Subject Membership Certificate List* (SMCL). The target AA verifies the integrity of these certificates and reports the result to the target ACA which validates the policies in the CCPL for which the required domain membership has been ensured. The validated policies in the CCPL are entered in the *Resolved Channel Policy List* (RCPL) which is eventually given to the RMF in the address space of the target. The RCPL contains information that enables the RMF to make access control decisions on a per-operation basis.

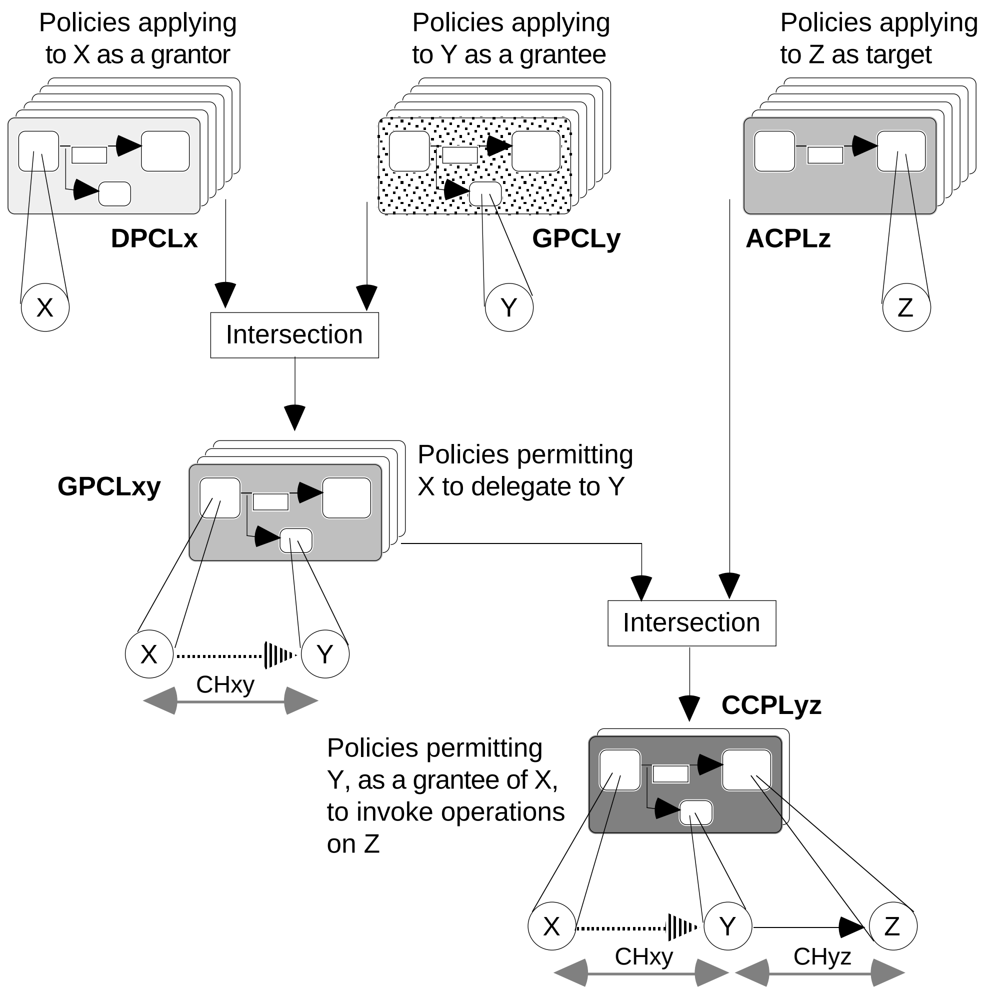

Policies applying to X as a grantor
Policies applying to Y as a grantee
Policies applying to Z as target
DPCLx
GPCLy
ACPLz
X
Y
Z
Intersection
GPCLxy
Policies permitting X to delegate to Y
X
Y
CHxy
Intersection
CCPLyz
Policies permitting Y, as a grantee of X, to invoke operations on Z
X
Y
Z
CHxy
CHyz

Figure 6.12 Construction of the CCPL for One-Hop Delegation

Note that if the subject ACA sends a incorrect PCL, that is, a list of policies that do not apply to the subject, the target AA would not be able to verify the requested domain membership and thus access would not be granted. It is the responsibility of the subject ACA to send the correct PCL in order to allow the construction of the appropriate RCPL.

6.4.3 One-Hop Delegation

We assume an one-step delegation scenario where the grantor delegates all its access rights to a grantor. Delegation of access rights takes place in the framework of a secure channel between the grantor and the grantee. The CHID of the channel can be used as a reference to the information associated with the delegation. The construction process of the RCPL that enables the grantor to make use of the delegated access rights is shown in Figure 6.13.

The grantor ACA maintains the DPLC of *X* which specifies the policies enabling *X* to delegate access rights. This list is given to the grantee ACA which compares it with the GPCL of the grantee in order to determine the policies enabling *Y* to act as grantee of *X*. The intersection of these two lists gives the GPCL for the channel between *X* and *Y* (*CHxy*). This list (which is denoted as *GPCLxy* in Figure 6.12) is given back to the grantor ACA which determines the membership certificates that have to be collected in order to support the policies in the *GPCLxy*. The list of these certificates is called the *Delegation Membership Certificate List* (DMCL) and is constructed by the subject AA. The subject AA also signs a delegation token (*DTxy*) which certifies that *X* has delegated its access rights to *Y*. The structure and use of the DT is described in Chapter 7. In this simple scenario *X* delegates all the access rights it is permitted to delegate to *Y* and the DT does not specify any special restriction on the use of these rights.

The signed DT and the *DMCLx* are given to the grantee AA and this completes the delegation phase. When *Y*, acting as a grantee of *X*, establishes a channel between itself and a remote object *Z*, the *GPCLxy* is given to the AA of *Z*. This determines the CCPL for this channel which contains the policies permitting *Y*, acting on behalf of *X*, to invoke operations on *Z*. This list is the intersection of *GPCLxy* and *ACPLz* and is denoted as *CCPLyz* in Figure 6.13. The rationale behind the construction of this list is illustrated in Figure 6.12. In order to validate this list, the ACA of *Z* has to ensure that:

- *X* has delegated *Y* (1)
- *X* is in the subject scopes of the policies in *CCPLyz* (2)
- *Y* is in the grantee scopes of the policies in *CCPLyz* (3)

The DT held by *AAy* provides sufficient grounds to prove (1). Also, the *DMCLxy* contains all the necessary certificates to prove (2). In order to prove (3), the grantee ACA determines the *GMCLyz,* which is the list of certificates proving that *Y* is in the grantee scopes of the policies in the *CCPLyz*. These certificates are collected by the AA of *Y* and are sent to *AAz* along with the *DTxy* and *DMCLxy*. If the verification of the DT and the certificates is successful, the ACA validates the *RCPLyz* which is eventually given to the RMF in the address space of *Z*.

The execution of the delegation process does not exclude the execution of the process described in 6.4.3, that is the process to determine the policies permitting *X* to invoke operations on the channel *CHxy*. In fact, this process takes place since *X* has to invoke operations on *Y*. Typically, these operations trigger further operations for which *Y* needs the delegated access rights.

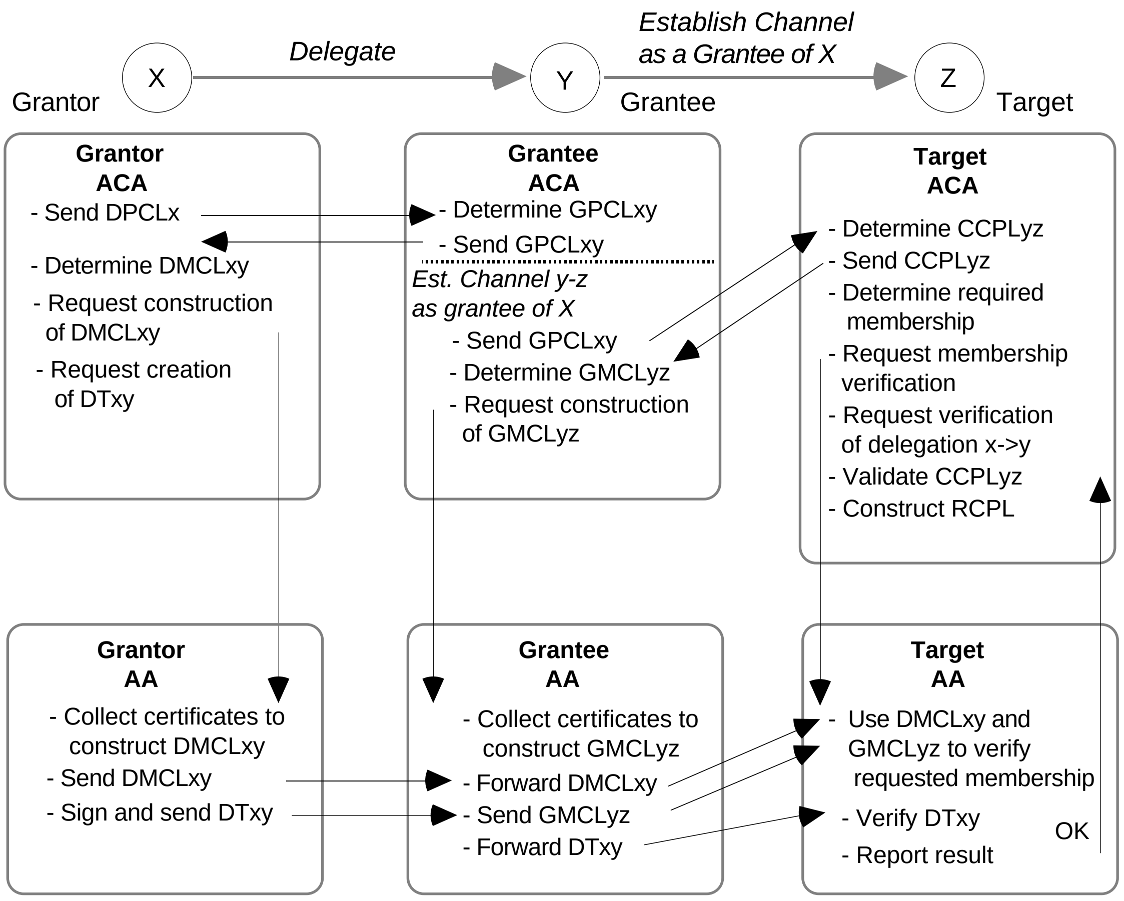

Delegate
Establish Channel as a Grantee of X
X
Y
Z
Grantor
Grantee
Target
Grantor ACA
- Send DPCLx
- Determine DMCLxy
- Request construction of DMCLxy
- Request creation of DTxy
Grantee ACA
- Determine GPCLxy
- Send GPCLxy
Est. Channel y-z as grantee of X
- Send GPCLxy
- Determine GMCLyz
- Request construction of GMCLyz
Target ACA
- Determine CCPLyz
- Send CCPLyz
- Determine required membership
- Request membership verification
- Request verification of delegation x->y
- Validate CCPLyz
- Construct RCPL
Grantor AA
- Collect certificates to construct DMCLxy
- Send DMCLxy
- Sign and send DTxy
Grantee AA
- Collect certificates to construct GMCLyz
- Forward DMCLxy
- Send GMCLyz
- Forward DTxy
Target AA
- Use DMCLxy and GMCLyz to verify requested membership
- Verify DTxy
- Report result
OK

Figure 6.13 Construction of the RCPL When Delegation Takes Place

6.4.4 Cascaded Delegation

The process described in 6.4.3 can be extended to cope with cascaded delegations. Figure 6.14 depicts the basic message exchange in the ACA and AA layers in a *(n-1)*-hop delegation. The grantee ACA of *Xk*, where *k >2*, receives the list $GPCLx_{k-2,k-1}$ from its preceding grantor ACA and constructs the list $GPCLx_{k-1,k}$ which is the intersection of $GPCLx_{k-2,k-1}$ and $GPCLx_k$ (denoted as $GPCLx_{k-2,k-1} \lozenge GPCLx_k$). $GPCLx_{k-1,k}$ specifies the policies enabling *X1* (the initial grantor) to delegate to *X2, ...Xk-1*. $GPCLx_k$ specifies the policies applying to *Xk* as a grantee. Thus, the intersection of these two lists gives the policies enabling *X1* to delegate to *X2, ...Xk*. The last grantee ACA constructs the list *GPCLx*$_{n-1,n}$ and forwards it to the ACA of the endpoint which determines the CCPL of the channel *CHn,z*. This list specifies the policies enabling *Xn* to invoke operations on *Z* when *Xn* acts on behalf of *Xn-1*, which acts on behalf of *Xn-2,....* which acts on behalf of *X1*.

In the AA layer, each grantor AA signs a delegation token which is passed to the next grantee along with all the preceding delegation tokens. The last grantee forwards all these delegation tokens to the endpoint AA which verifies all the delegation hops. In addition, the AA of *Xk* constructs the certificate list *GMCLx*$_{k,k+1}$ to support the grantee pseudo-capability list *GPCLx*$_{k,k+1}$. All these lists along with the *DMCLx*$_{1,2}$ are eventually forwarded to the endpoint AA in order to validate the CCPL of the channel *CHn,z*.

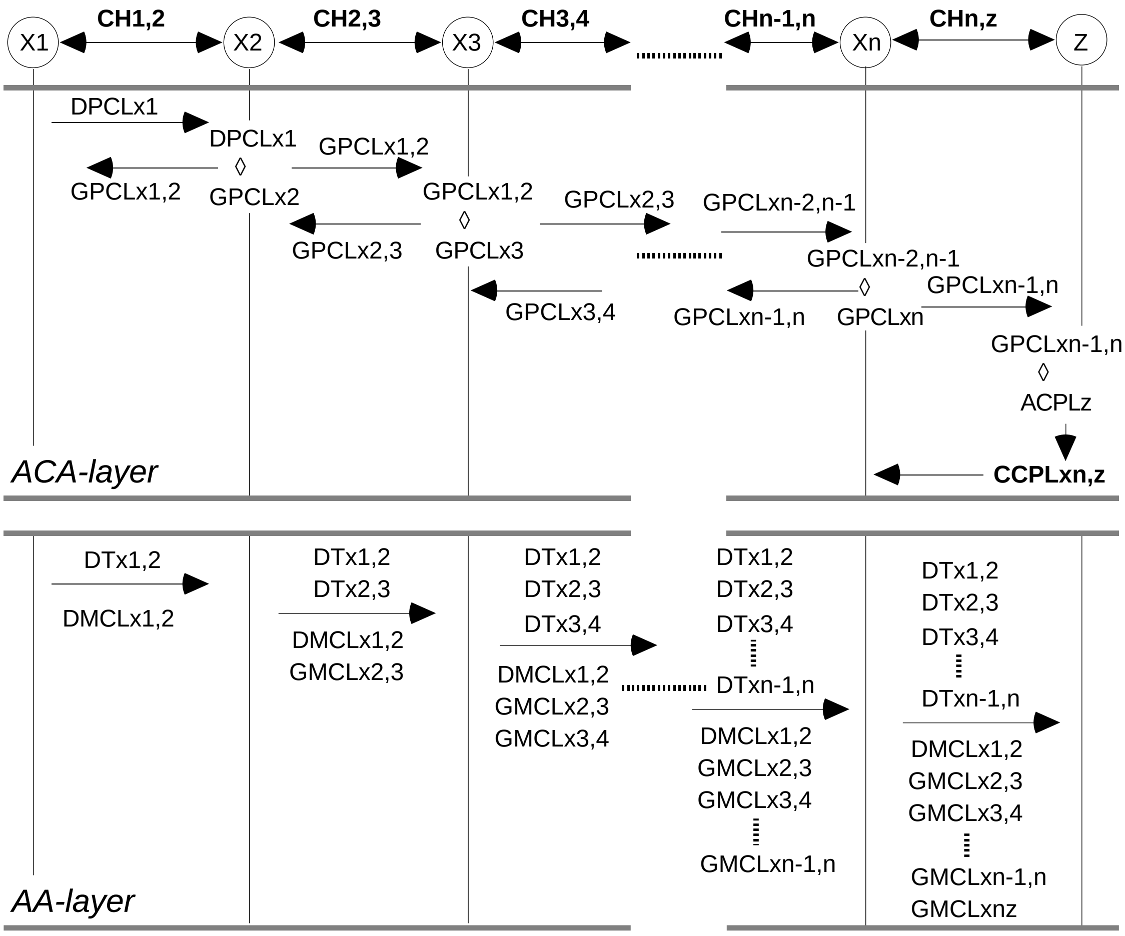


Figure 6.14 Cascaded Delegation

It is possible that an intermediate grantee acts as an endpoint for the preceding delegations. For example *X2* may have to invoke an operation on *X3* using the rights it has been given by *X1*. In this case, the ACA of *X3* has to construct and validate the CCPL for the channel *CH2,3*.

6.5 Restricted Access Rights

In the previous section we assumed, for the sake of comprehension, that the subject uses all its access rights and that the grantor delegates all the rights it is permitted to delegate. In practice, however, it may be the case that the subject or the grantor restricts the access rights used or delegated on an channel.

It is important that the system provides a convenient method for restricting rights. Typically, the restricted rights are defined in the application layer so the restriction method should be flexible and hide as much of the detail concerning channel establishment as possible. This section discusses a method which is based on the reduction of the domain membership of the subject or grantor. We first give a typical example in order to show how the membership reduction can be used as a practical way to restrict access rights.

6.5.1 An Example

Let us assume that the domain structure given in Figure 6.15 is a part of the domain structure used to manage computer resources at a college. The user Alice is represented within the system by her URD denoted as *Alice_URD*. This is a member of the domains *System_Administrator (SA)* and *Lecturers* to reflect the fact that Alice is a lecturer and she is also in charge of system administration. The domain *Academic_Staff* is a parent domain of *Lecturers* to reflect that lectures are members of the academic staff. Finally, the domain *Users* is a parent domain of both *SA* and *Academic_Staff*.

We also assume that Alice uses an adapter object *X* to interact with the system. This object is a member of her URD. For this object to be member of the URD, user authentication is required. Object *X* inherits all the access rights defined for its parent domains. In fact, these rights are subject to propagation limitations but we assume, in this example, that all subjects' scopes are defined by simple DSEs of the form **Domain*. For example, the DSE **Users* demarcates the subject scopes of the policies defining access rights for all users. The set of these policies is denoted as *PS1* in Figure 6.15.

There are a number of policies *(PS5)* defining access rights for members of the domain *Alice_URD*. These access rights enable Alice to manipulate her own files, read her email, etc. It may be the case that Alice can foresee that these rights are enough for the task she wants to accomplish, for instance reading her email. It would beneficial for Alice to be able to establish secure channels using just these rights, especially if these rights have to be delegated to a remote object (see also the discussion about delegation in Chapter 4).

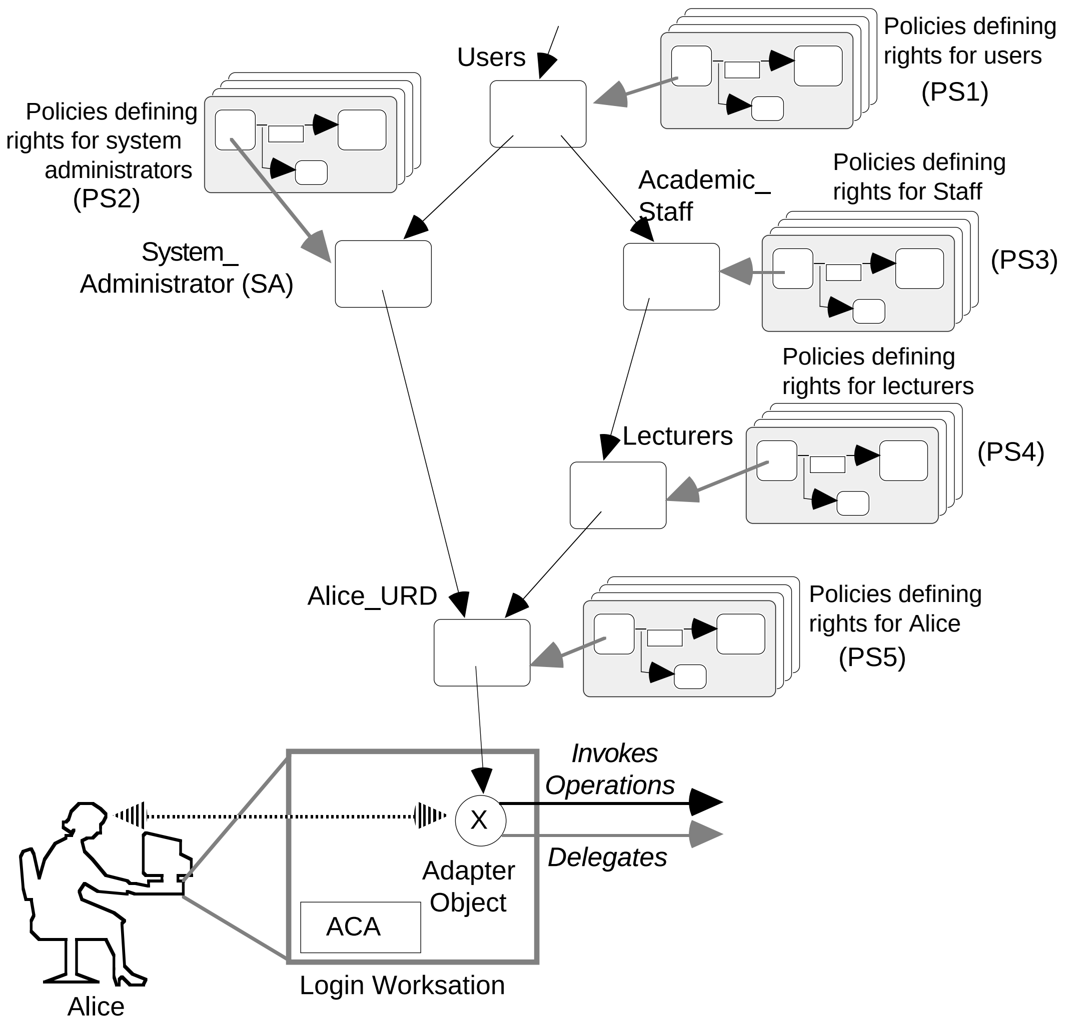

Users
Policies defining rights for users
(PS1)
Policies defining rights for system administrators
(PS2)
Academic_ Staff
Policies defining rights for Staff
(PS3)
System_ Administrator (SA)
Policies defining rights for lecturers
(PS4)
Lecturers
Alice_URD
Policies defining rights for Alice
(PS5)
Invokes Operations
X
Delegates
Adapter Object
ACA
Login Worksation
Alice

Figure 6.15 Example for Restriction of Access Rights

According to the discussion in Section 6.4.1, the ACA of *X* receives pseudo-capabilities and delegation pseudo-capabilities that correspond to the policies in the sets *PS1-PS5*. These policies define the full access right set that *X* can use. But if the ACA of *X* uses as PCL the set of pseudo-capabilities that correspond to *PS5* only (this pseudo-capability set is a subset of *PCLx*), the resulting channel will enable *X* to invoke only the operations permitted by policies in *PS5*. Thus, the access rights of *X* are restricted to those defined by *PS5*.

The ACA maintains the propagation paths of the PCFs and therefore it can recognise the pseudo-capabilities representing policies defined for the domain *Alice_URD*. In other words, the ACA can identify those pseudo-capabilities that have been propagated from the domain *Alice_URD*.

Similarly, the ACA can determine the policies defined for the domain *Users, Lecturers* or any other domain in Figure 6.15. For instance, the PCFs associated with the policies in *PS4*, have recorded the propagation path *[Lecturers, Alice_URD, X]*. Note that the restriction of the access rights in this way may considerably reduce the number of certificates that have to be collected and verified as well as the size of the CCPL. The same technique can be applied on the DPCL in order to restrict the rights that a grantor delegates on a channel.

Note that as we "travel" down the domain structure, the access rights defined for lower subdomains have greater significance or power. For instance, the policies defined for *Academic_Staff* give more rights for accessing certain resources than those defined for *Users*. Further, members of the academic staff inherit the rights defined for all users.

In general, it useful to select the rights defined for a parent domain and those defined for its ancestor domains. The following section presents an expression syntax that enables the application object to specify the access rights it wants to use or delegate in terms of selected domain membership.

6.5.2 Pseudo-Capability Select Expressions

We use pseudo-capability select expressions in order to specify the access rights a subject wants to use on a channel or the access rights a grantor wants to delegate. A *Pseudo-Capability Select Expression* (PCSE) is applied on a PCL or a DPCL and returns a subset of it. This subset is denoted as $PCLx_{/PCSE}$ where *PCSE* is the select expression applied on the complete PCL of the object *X*. The PCSE syntax is given by the following notation:

```
PCSE ::= ALL |
         SELF |
         domain |
         Δ domain |
         PCSE + PCSE
```

where:

- ALL selects the complete pseudo-capability list
- SELF selects the pseudo-capabilities that have been defined for the object rather than been inherited (i.e. using the operator { })
- `domain` selects the pseudo-capabilities defined for the parent domain `domain`
- Δ `domain` selects the pseudo-capabilities defined for the parent domain `domain` and for all its ancestors
- + is the set union operator

Note that *PCLx* is equal to $PCLx/_{ALL}$ and $PCLx/_{PSCE1}$ + $PCLx/_{PSCE2}$ is equal to $PCLx/_{PSCE1\ +\ PSCE2}$. In the example illustrated in Figure 6.15, if *PCLx* is the list of all pseudo-capabilities held by the ACA on behalf of *X*, the restricted PCL $PCLx/_{Users\ +\ Alice\_URD}$ contains the pseudo-capabilities corresponding to the policies in *PS1* and *PS5*. The restricted PCL $PCLx/_{\Delta Lecturers}$ contains the pseudo-capabilities corresponding to the policies in *PS1, PS3* and *PS4*. Similarly, *X* can restrict its DPCL. For example, if it wants to delegate the rights specified by the extended access rules in *PS1* and *PS5,* it instructs the ACA to construct the restricted DPCL $DPCLx/_{Users\ +\ Alice\_URD}$.

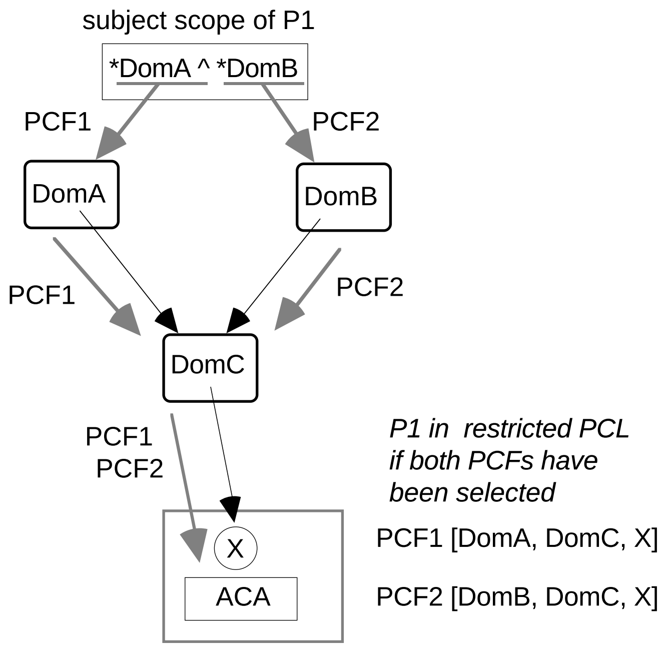


Figure 6.16 Example for Pseudo-Capability Select Expressions

Note that the ACA can construct a restricted PCL (or DPCL) of the form $PCLx/_{\Delta domain}$ by choosing the pseudo-capabilities of which the PCFs have been propagated through *domain*. For example. the propagation path of the PCFs corresponding to policies in *PS3*, is *[Academic_Staff, <u>Lecturers</u>, Alice_URD, X]*. Therefore, the restricted PCL *PCLx/ΔLecturers* contains, amongst others, the pseudo-capabilities corresponding to the policies in *PS3*.

It can be shown that the PCSEs can also be used when the subject scopes of policies are demarcated by more complex DSE such as **N DomA ^ *DomB*. We give an example to show how this can be done when intersection operators (^) are used in the subject DSEs. In Figure 6.16, the ACA of *X* receives two PCFs for policy *P1*. This implies that *X* is in the subject scope of *P1* and therefore the OID of *P1* is inserted in the *PCLx*. If we assume that the PCSE "*DomA*" is used, the restricted PCL *PCLx/DomA* would not contain *P1*. This is because this PCSE does not match the propagation path of *PCF2* and therefore this PCF is excluded. Similarly, the restricted PCL *PCLx/DomB* does not contain *P1*. On the other hand, the restricted PCL *PCLx/DomC* contains *P1* as the propagation paths of both *PCF1* and *PCF2* include *DomC*.

6.5.3 Restriction of Rights by a Grantee

A grantee can also restrict the access rights it uses to access the endpoint. Similarly, an intermediate grantee can restrict the delegated access rights it forwards to the next grantee. As shown in Figure 6.14, these rights are defined by the policies that are in both the GPCL received from the preceding grantor and the GPCL of the grantor (i.e. $GPCL_{k-2,\ k-1}$ ◊ $GPCL_k$ where X_k is the intermediate grantor). The ACA of X_k can reduce this list by applying a PCSE to the grantee scopes of the policies in $GPCL_k$ or to the subject scope of the policies in $GPCL_{k-2,k-1}$. For the latter to be achieved, the ACA should know the propagation paths of the PETs received by the initial grantor for the subject scope of the policies in $GPCL_{k-2,k-1}$. These paths are given by the initial grantor along with the DPCL and are forwarded by the preceding grantees along with the GPCLs.

6.6 Access Control API

This section discusses an Application Programming Interface (API) that allows the application layer to access information related to secure channels and the policies specifying access privileges for the object, as well as to control channel establishment and the delegation of access rights.

6.6.1 The Interface

We first discuss a simplified API enabling an object acting as a subject to establish a secure channel with a target object. We assume that the OID of the latter is known to the subject which establishes a secure channel by invoking:

```
establish_channel(target, pcse)
```

where *target* stands for the OID of the target and *pcse* stands for the PCSE that restricts the access rights the subject can use on this channel. If *pcse* is "*ALL*", the subject ACA will send the complete PCL to the target ACA and therefore the object will be able to use all the rights it possesses This call, if successful, returns the channel identifier *(chid)* of the established channel and the object can now invoke operations on that channel.

The subject can also delegate access rights to the target associated with the established channel by calling:

```
delegate(chid, pcse_grantor)
```

where *pcse_grantor* is a PCSE restricting the rights to be delegated. So, this first simple interface is:

```
CHID establish_channel(OID target,
                       PCSE subject_restriction);
delegate (CHID chid, PCSE grantor_restriction);
```

This interface, however, does not enable an object to establish a channel as a grantee or to forward access rights it has been delegated. An improved version is:

```
CHID establish_channel(OID target,
                       PCSE subject_restriction,
                       CHID grantor_chid,
                       PCSE grantor_restriction,
                       PCSE grantee_restriction);
delegate (CHID chid, PCSE grantor_restriction,
                       CHID p_grantor_chid,
                       PCSE p_grantor_restriction,
                       PCSE grantee_restriction);
```

The CHID *grantor_chid* refers to the channel through which the object has been delegated access rights. In the *establish_channel* call, the caller declares that it intends to use these rights as a grantee in addition to its own access rights. In the *delegate* call, the caller restricts its own rights to be delegated using the PCSE grantor_restriction. It can also forward the rights that have been delegated through CHID *p_grantor*. The parameters *p_grantor_restriction* and *grantee_restriction* are used to reduce these access rights along the lines discussed in Section 6.5.3.

At this point it is apparent that a subject should be able to have access to certain channel information as well as to the policies defining its own access rights. For these reasons, two additional calls have been included in the interface:

```
PolicySet get_policies();
ChInfo get_channel_info(CHID chid);
```

The *get_policies()* call enables an object to get the policies applying to it as subject, grantor and grantee. This information is accompanied by the propagation paths of the related PCFs so that the application layer can determine the PCSE it has to use in order to reduce its access rights. The second call enables the object to get access control information related to a given channel. This information includes:

- The OIDs of the preceding grantors if the object has been delegated access rights through *chid*.
- The policies defining the access rights it has been delegated and the propagation paths of the related PCFs.
- The policies defining the access rights of the subject. This information can be used by the target in order to make access control decisions that cannot be made by the RMF.

We can now give the complete version of the access control API. This has been enhanced to allow the subject to determine the security level the secure channel should provide (parameter *type*). This is further discussed in Chapter 7. Also, an object may discard an established channel by calling *discard_channel(chid)*.

```
CHID establish_channel(ChType type, OID target,
                   PCSE subject_restriction,
                   CHID grantor_chid,
                   PCSE grantor_restriction,
                   PCSE grantee_restriction);
delegate (CHID chid, PCSE grantor_restriction,
                   CHID p_grantor_chid,
                   PCSE p_grantor_restriction,
                   PCSE grantee_restriction);
discard_channel(CHID chid);
PolicySet get_policies();
ChInfo get_channel_info(CHID chid);
```

6.6.2 An Example

We use the example discussed in Section 4.3.3 to illustrate the use of the API to control the delegation of access rights. Consider the delegation scenario depicted in Figure 4.10. Bob's adapter object establishes a secure channel with the *DBMD_1* and subsequently delegates all its privileges to it:

```
chid_dbms_1 = establish_channel (Authenticity, DMBS_1,
                                  ALL, NIL, NIL, NIL)
delegate (chid_dbms_1, ALL, NIL, NIL, NIL)
```

The first call establishes a channel with *DBMS_1* that provides message authenticity (discussed in Chapter 7). The adapter object uses all its access rights to establish this channel. The second call delegates to *DBMS_1* all the access rights possessed by the adapter object. The *DBMS_1*, acting as a grantee of the adapter object, can now establish a secure channel with *File_B* :

```
chid_File_B = establish_channel(Authenticity, File_B, ALL,
                                  chid_dbms_1, ALL, ALL)
```

DBMS_1 uses all its access rights and the rights that it has been delegated through *chid_dbms_1*. Furthermore, *DBMS_1* can delegate these access rights to *Printer_2*:

```
chid_Printer_2 = establish_channel(Authenticity,
                  Printer_2, ALL, chid_dbms_1, ALL, ALL)
delegate (chid_Printer_2, ALL, chid_dbms_1, ALL, ALL)
```

Printer_2 can use all the delegated rights to establish a channel with *File_B*:

```
chid_File_B = establish_channel(Authenticity, File_B, ALL,
                             chid_Printer_2, ALL, ALL)
```

Consider now the delegation scenario in Figure 6.17. Alice's adapter object *(B)* inherits access rights from both the domains *Users* and *Trusted_Users*. *B* can delegate all these rights to *DBMS_1*:

```
chid_dbms_1 = establish_channel (Authenticity, DMBS_1,
                                  ALL, NIL, NIL, NIL)
delegate (chid_dbms_1, ALL, NIL, NIL, NIL)
```

DBMS_1 uses these rights to establish a channel with *File_A*:

```
chid_File_A = establish_channel(Authenticity, File_A, ALL,
                                  chid_dbms_1, ALL, ALL)
```

Furthermore *DBMS_1* can establish a channel with *File_B* using restricted access rights:

```
chid_File_B = establish_channel(Authenticity, File_B, ALL,
                                  chid_dbms_1, Users, ALL)
```

Note that the rights specified for domain *Users* (*AR1-4* in Table 4.1) enable *DBMS_1* to read *File_B*. Moreover, *DBMS_1* can forward these restricted access rights to *Printer_1*:

```
chid_Printer_1 = establish_channel(Authenticity,
                     Printer_1, ALL, chid_dbms_1, ALL, ALL)
delegate (chid_Printer_1, ALL, chid_dbms_1, Users, ALL)
```

Printer_1 can use these rights to read *File_B*:

```
chid_File_B = establish_channel(Authenticity, File_B, ALL,
                             chid_Printer_1, ALL, ALL)
```

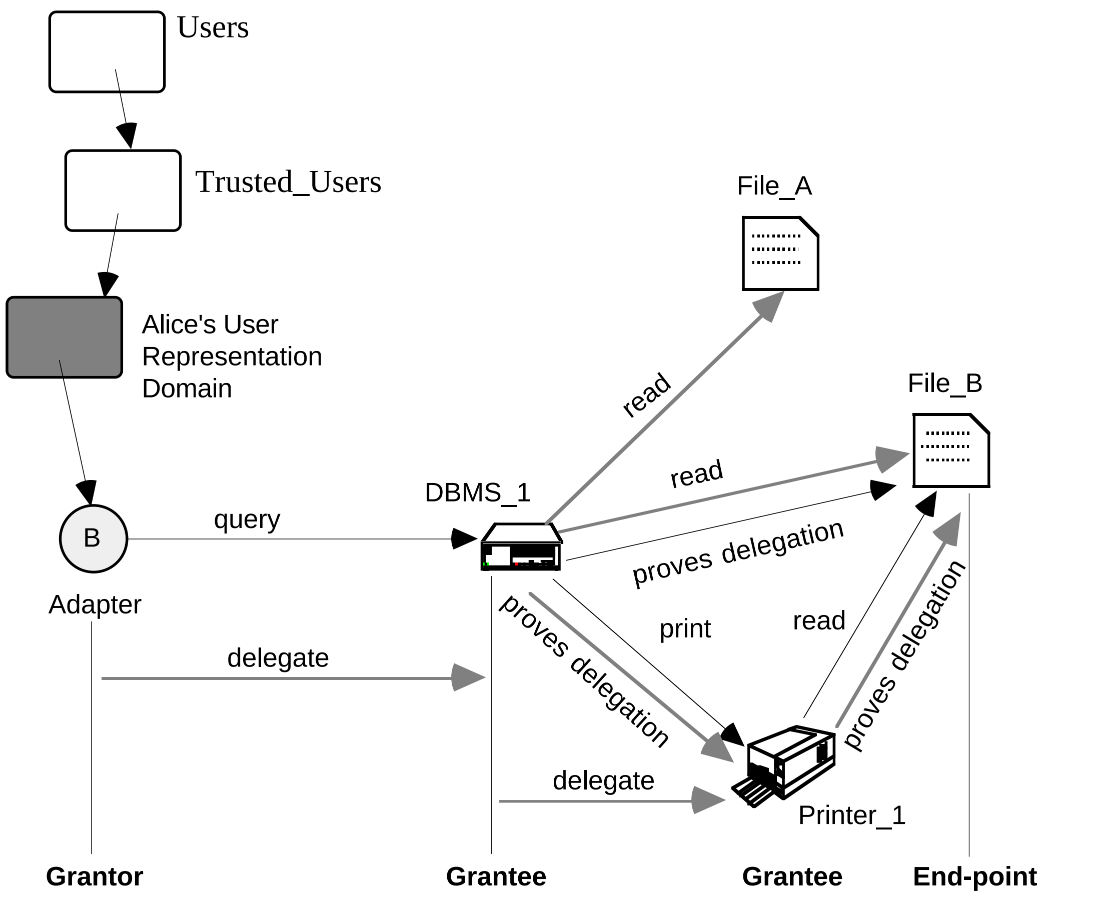


Figure 6.17 Cascaded Delegation Controlled by the Rules in Table 4.1

Printer_1, however, cannot read *File_A* because it has not been delegated the access rights specified for the members of the domain *Trusted_Users*. Thus an attempt to establish a channel with *File_A*:

```
chid_File_A = establish_channel(Authenticity, File_A, ALL,
                             chid_Printer_1, ALL, ALL)
```

will fail because there are no access rules permitting *Printer_1* to access that file (the CCPL that is determined by the ACA of the file server is empty).

6.7 Chapter Summary

This chapter has discussed the policy dissemination and access control mechanism. Since the domain structure changes, the sets of objects defined by the policy scopes are not static. The dissemination mechanism employs the domain service to propagate Policy scope Evaluation Tokens (PETs) that are given to the ACAs to determine which policies apply to objects on their hosts. The policies themselves are given by the policy servers to the ACAs that have to enforce them. Since this mechanism may be interrupted by communication or server failures, there is a need to employ a mechanism that enables administrators to monitor the policy dissemination and take the necessary counter measures should a problem arise.

The access control mechanism supports cascaded delegation of access rights and makes use of the notion of a pseudo-capability to minimise the policy space that has to be searched by target ACAs as well as the number of membership certificates that have to be collected by the subject. The set of policies permitting a subject to access a target is determined by the target ACA during the secure channel establishment process. This is given to the reference monitor in the address space of the target server to control the invocation of operations on the interface of the target.

The certificates stating the domain membership of the subject are collected by the subject AA and are forwarded to the target AA which accomplishes the necessary membership and delegation verification to support the access control decisions made by the ACA.

A subject or a grantor can restrict the access rights it uses or delegates on a channel by using pseudo-capability select expressions. These reduce the access rights of the subject or grantor by selecting policies that have been propagated from particular parent domains.

An API enables the application level to control the restriction of access rights and get information associated with a channel. Thus, security aware objects can make further application dependent access control decisions and control the delegation of their access rights.

Chapter 7

Authentication System

This chapter discusses the authentication system supporting the access control mechanism described in Chapter 6. The authentication system is based on symmetric cryptography to reduce the cryptographic overhead and supports domain membership verification as well delegation of access rights.

7.1 Introduction

As discussed in Chapter 5, the authentication task is undertaken by the Authentication Agents (AAs) which are trusted to serve all the objects on their hosts. The authentication operations are transparent to the ACAs and to the cryptographic facilities in the application servers.

This chapter shows how the authentication between two AAs can be achieved. Two authenticated AAs can provide the cryptographic facilities of the application servers on their hosts with keys to be used for secure communication on an established channel.

Since the access control decisions are heavily based on the domain membership of the subject and grantees, it is important that the system provides an efficient mechanism to verify domain membership. Note that the subject and grantee scopes of a policy are specified in terms of Domain Scope Expressions (DSEs) and that the parent domain hierarchy of an object may be complex in large distributed systems. In Chapter 6, we showed how an ACA can determine the set of membership certificates required to prove that an object is in the subject or grantee scopes of the policies applying to a particular secure channel. We have also shown how a grantor ACA can determine the minimal membership certificate set required to support delegation of access rights.

This chapter shows how these certificates can be collected by the subject, grantor and grantee AAs and verified by the AA of the end-point. The system adopts the *push* method, that is, the necessary membership certificates are “pushed” by the subject, grantor and grantee AAs to the AA of the end-point. This method exhibits a number of advantages compared to the *pull* method according to which the end-point collects the necessary certificates. Specifically, the push method reduces the task that has to be undertaken by the end-point. This is of importance when a host supports a large number of target objects that are accessed by multiple subjects. Another advantage is that a subject or grantee AA can cache membership certificates and use them to establish channels with multiple targets.

The AAs also support delegation of access rights by signing and verifying delegation tokens (DTs). This chapter describes a protocol that enables the end-point to verify a delegation chain of arbitrary length.

Finally, it is crucial that the cryptographic overhead is minimal as the target AA has to verify a number of certificates and delegation tokens that have been signed by multiple domain servers and grantors. The authentication system is based on symmetric cryptography in order to reduce the cryptographic overhead. The next section compares the use of asymmetric and symmetric cryptography with regard to the requirements posed by the access control mechanism described in Chapter 6.

7.2 Asymmetric vs. Symmetric Cryptography

7.2.1 Asymmetric cryptography

We first consider the use of an authentication system based on asymmetric cryptography. A typical example of an authentication system based on asymmetric cryptography is the SPX (Tardo and Alagappan 1991).

In this section we assume that an asymmetric authentication system is used to support the access control mechanism described in Chapter 6. In this case, the users and the AA representing their hosts possess public/private key pairs. The public keys of these principals are certified by the certification authority (CA). As it has been discussed in Chapter 2, an off-line certification authority can be used to certify the public keys of registered principals. The signed certificates can be retrieved from a public database which has to be trusted to promptly remove the certificates that have been revoked.

Domain servers can issue membership certificates which are signed with the private keys of their AAs. These certificates can be verified by another AA that can retrieve the corresponding public key. Similarly, a target AA can verify delegation tokens using the public keys of the grantor AAs. Session keys between remote AAs can also be established as all AAs possess a public/private key pair.

Use of asymmetric cryptography, on the other hand, introduces a high overhead in terms of the cpu-time required to sign and verify statements. As discussed in Chapter 2, symmetric cryptography is faster than asymmetric by a factor of 100 when they are both implemented in software. The access control system discussed in Chapter 6 requires the verification of a large number of membership certificates which could result in poor performance when no special hardware is used. For instance, in a typical scenario where a grantee requests the establishment of a secure channel, the target AA has to verify:

- membership certificates proving the required membership of the grantor
- membership certificates proving the required membership of the grantee
- the delegation token proving the delegation
- the public-key certificates of the principals that have signed these membership certificates
- the public-key certificate of the grantor AA
- the public-key certificate of the grantee AA

In such a scenario it is estimated that the target AA has to verify a number of certificates of the order of 10 but the exact number depends on the parent domain hierarchies of the grantor and grantee.

In addition to this, the grantor and grantee AAs have to sign delegation tokens while the AAs supporting domain servers have to sign domain membership certificates. It is worth noting that these membership certificates should have a short expiration time because many dynamic objects are expected to have a short life cycle and thus domains should frequently issue new certificates for their members (c.f. membership certificates signed by an off-line CA in the Taos OS). This also introduces a cryptographic overhead on the hosts of the domain servers which can significantly reduce the system performance.

Therefore, use of asymmetric cryptography gives rise to problems with respect to system performance. On the other hand, it provides a highly secure authentication system as the CAs can be physically secure as they can operate off-line.

7.2.2 Symmetric cryptography

Alternatively, symmetric cryptography can be used to provide an authentication service. Typical examples of systems that use symmetric cryptography are Kerberos (Neuman and Ts'o 1994) and the Kryptoknight (Molva et al. 1992). Two registered AAs can establish a shared key by executing an authentication protocol which involves the authentication server. In order to increase the availability and performance of the authentication service, replicated authentication servers may be used.

There is a problem however, with respect to the verification of the domain membership and delegation. This is because a verifier does not normally share keys with the principals that certify the required membership or delegate access rights. This problem can be tackled by utilising the authentication server as a translator or verifier of the encrypted tokens proving membership or delegation on the assumption that the authentication server shares keys with all the principals in the system.

Another problem with the use of symmetric cryptography is that it is difficult to replicate the security database in order to increase the availability and performance of the authentication service.

7.3 Authentication Servers with Minimal State

As discussed in Chapter 2, symmetric cryptography can simulate asymmetric cryptography by employing trusted relays that "translate" encrypted messages. This is of great importance in our system as it enables the use of symmetric cryptography to sign delegation tokens and membership certificates without knowing who will verify them. Furthermore, relays that maintain minimal state can be employed to facilitate replication. We show how servers with minimal state can be used to provide an authentication service (AS) that supports shared key establishment, as well as verification of domain membership and delegation of access rights.

7.3.1 Private-Key Certificates

(Davis and Swick 1990) proposes an authentication scheme that utilises trusted servers with minimal state. As shown in Figure 7.1, the principals' keys are published under the cryptographic protection of a *master key* *Km* which is known only to the replicated servers. A *Private-Key Certificate* (PKC) contains the encryption key and identity of a registered principal and is encrypted under the master key. A server can retrieve the necessary principals' keys by decrypting the appropriate private-key certificates which can be provided by any principal. Thus, the master key is the only state information held by the servers which has to be kept secure.

The main advantage of this scheme is that the servers can easily be replicated without giving rise to state consistency problems. A higher degree of replication also permits improved performance and availability of the authentication service. Furthermore, tamper proof machines can be used as trusted servers which increases the security of the system.

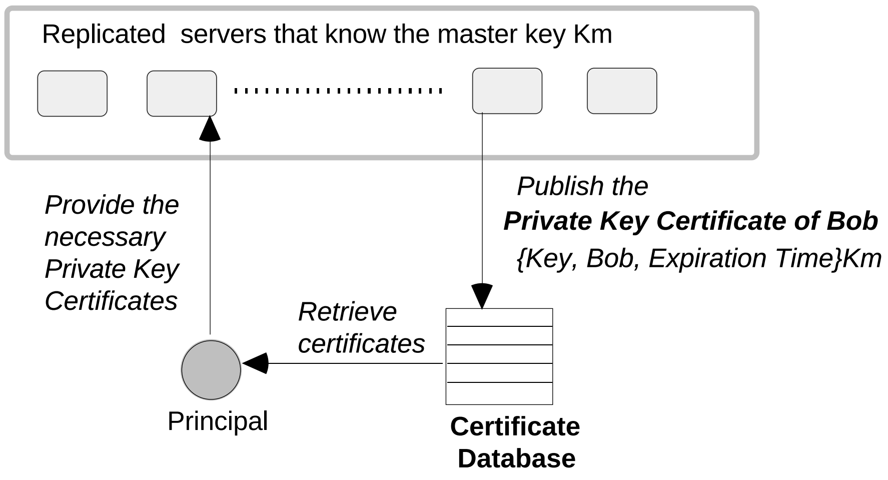

Replicated servers that know the master key Km
Provide the necessary Private Key Certificates
Publish the
Private Key Certificate of Bob
{Key, Bob, Expiration Time}Km
Retrieve certificates
Principal
Certificate Database

Figure 7.1 Using Private Key Certificates to Disperse the Security Database

7.3.2 Basic Aspects and Notation

In this Chapter, a message M encrypted with the key K in a way that preserves message secrecy and integrity is represented as $\{M\}_K$. If a hash function such as MD5 is used, the notation $\{M\}_K$ represents the message and its hash value that have been encrypted under the key K, that is *encrypt((M, H(M), K)*.

The expression $A \Rightarrow B$ denotes that A *speaks for* B where A and B are principals. The relation *speaks for* is defined as in (Lampson et al. 1992), that is A *speaks for* B if the fact that principal A says something means that we can believe that principal B says the same thing. If keys are considered to be principals, the statement $K \Rightarrow B$ means that if we receive a message M encrypted under the key K, we can believe that M has been said by B.

The notation $A \xleftrightarrow{K} B$ is used as in (Burrows et al. 1990), to denote that the shared key K can be used by the principals A and B to communicate with secrecy and integrity. That is, K is known only by A, B and principals that are trusted not to use or reveal it. If A believes that $A \xleftrightarrow{Kab} B$ is true, it derives that $Kab \Rightarrow B$. Similarly, B derives $Kab \Rightarrow B$ if it believes that $A \xleftrightarrow{Kab} B$ is true.

We utilise a number of replicated servers that provide an *Authentication Service* (AS). These servers know the same master key *Km* which is never revealed to other principals. Each server has a unique identifier. We assume that there is a unique realm and all principals' certificates are signed with the same master key.

Each private-key certificate encrypted by a server contains the unique ID of the server and the time it was generated which together form the *unique identifier* of the certificate (*CID*). In addition, each certificate contains an *expiration time* which indicates the time when the certificate becomes invalid.

The wire representation of a PKC that certifies the key *Kx* of a registered authentication agent *AAx* is:

$$(AAx,\ Te,\ \{CID,\ Te,\ AAx,\ Kx\}_{Km})$$

where *Te* is the expiration time. The OID of the AA and the expiration time that are transmitted in plaintext are used in order to identify certificates. This does not ensure, however, that one is using the right certificate. A substituted certificate can only be detected by the AS.

In the interests of comprehension, we denote the private-key certificate of the authentication agent *AAx* as:

$$\{CID, Te, AAx \xleftrightarrow{Kx} AS\}_{Km}$$

The registered authentication agent *AAx* receives and stores a PKC along with its key *Kx*. Because the AS cannot authenticate an unregistered AA, the security administrator has to be involved in the registration process which also involves out-of-band communication. A registration protocol is given in (Davis and Swick 1990).

Once an AA receives its first key, it can establish further keys with the AS. This is important as a key should be used for encrypting a limited amount of traffic.

7.3.3 Establishing Shared Keys Between Authentication Agents

We assume that the communication security between two address spaces on the same host is provided by the system software/hardware which has to be trusted anyway. It is necessary, however, to employ cryptographic and authentication techniques to provide secure communication between objects on different hosts. In this section we discuss the establishment of a shared key between two registered AAs.

The registered authentication agent *AAx* of the host *X* shares a key *Kx* with the AS (see Figure 7.2). This key is certified by a private-key certificate of the form:

$$\{CID_x, Te_x, AAx \xleftrightarrow{Kx} AS\}_{Km} \qquad \text{(PKCx)}$$

Similarly, the key of another registered authentication agent *AAy* is certified by the private-key certificate:

$$\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km} \qquad \text{(PKCy)}$$

These two AAs can execute a protocol to establish a shared key *Kxy*. We use a protocol that is based on a protocol presented in (Davis and Swick 1990) and is similar to the one employed by Kerberos V5 (Neuman and Ts'o 1994). The main difference is that the server involved in the protocol is provided with the private-key certificates of the two AAs.

Message 1: AAy -> AAx: $\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 2: AAx -> AS: $\{CID_x, Te_x, AAx \xleftrightarrow{Kx} AS\}_{Km}$,
$\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 3: AS -> AAx: $\{Ts, Te_{xy}, AAy, Kxy\}_{Kx}$,
$\{Ts, Te_{xy}, AAx, Kxy\}_{Ky}$

Message 4: AAx -> AAy: $\{Ts, Te_{xy}, AAx, Kxy\}_{Ky}$, $\{AAx, Tx\}_{Kxy}$

Message 5: AAy -> AAx: $\{Tx\}_{Kxy}$

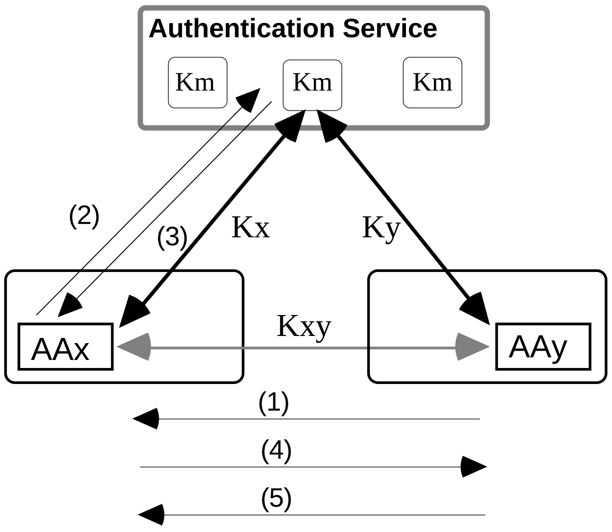


Figure 7.2 Shared secret key establishment between authentication agents

In the first message, *AAx* gets the private-key certificate *PKCy* of *AAy*. In message 2, *AAx* sends that certificate along with its own certificate *PKCy*. At this point the authentication server can decrypt these two certificates, as it knows the master key *Km*, and retrieve the keys of *AAx* and *AAy*. If these two certificates have not expired, we can state that:

$$AS\ \textbf{believes}\ (AAx \xleftrightarrow{Kx} AS) \text{ and } AS\ \textbf{believes}\ (AAy \xleftrightarrow{Ky} AS)$$

The rest of the protocol can now be analysed using BAN logic. (Burrows et al. 1990) analyse a similar protocol based on the one employed by Kerberos V4 (Miller et al. 1987). If we assume that the clocks of the principals are synchronised, upon completion of the protocol we derive that:

$$AAx \textbf{ believes } (AAx \xleftrightarrow{Kxy} AAy)$$

$$AAx \textbf{ believes } (AAy \textbf{ believes } (AAx \xleftrightarrow{Kxy} AAy))$$

and

$$AAy \textbf{ believes } (AAx \xleftrightarrow{Kxy} AAy)$$

$$AAy \textbf{ believes } (AAx \textbf{ believes } (AAx \xleftrightarrow{Kxy} AAy))$$

The shared key *Kxy* can then be used for secure communication between *AAx* and *AAy*. If, for instance, *AAx* sends a time stamped message *M* encrypted with *Kxy*, i.e. $\{Ts, M\}_{Kxy}$, *AAy* derives that *AAx* ***says*** *(Ts, M)*. This is because AAy believes that $Kxy \Rightarrow AAx$. The timestamp *Ts* guarantees the freshness of the message (i.e. the message is not a replay of an old message). In addition, as no untrusted principal knows the key *Kxy,* we also achieve message secrecy. We assume that the AS, which knows the key *Kxy,* never uses or reveals this key to untrusted principals.

Once a shared key has been established between two AAs, further session keys can be established without the use of the AS. These keys can either be used for secure communication between the AAs or for secure communication between other objects on the two nodes.

7.3.4 Translation of Messages

The authentication service can also be used to "translate" encrypted messages that are exchanged between registered principals. In particular, it is used to "translate" statements such as membership certificates or delegation tokens encrypted with the keys of the authentication agents.

Figure 7.3 illustrates two possible ways to translate a message *M* that has been encrypted by *AAx* with its secret key *Kx*. According to the first method depicted in Figure 7.3(a), the encrypted message is sent by *AAx* to an authentication server along with the PKCs that contain the keys of *AAx* and the receiver *AAy*. The server retrieves the two keys *Kx* and *Ky*, decrypts *M* and believes that *AAx* ***says*** *M* as *Kx* speaks for *AAx*. It then encrypts the statement *AAx* ***says*** *M* with *Ky* and returns it to *AAx*. This subsequently forwards the translated message to *AAy* which decrypts it and establishes that *AAx* ***says*** *M*. The complete message sequence is as follows:

Message 1: AAx -> AS: $\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$,

$\{CID_X, Te_X, AAx \xleftrightarrow{Kx} AS\}_{Km}, \{M\}_{Kx}$

Message 2: AS -> AAx: $\{AAx\ \textbf{says}\ M\}_{Ky}$

Message 3: AAx -> AAy: $\{AAx\ \textbf{says}\ M\}_{Ky}$

Alternatively, the encrypted message *M* is sent to the receiver *AAy* which requests an authentication server to translate it. This method is illustrated in Figure 7.3(b) and the complete message sequence is:

Message 1: AAx -> AAy: $\{M\}_{Kx}, \{CID_X, Te_X, AAx \xleftrightarrow{Kx} AS\}_{Km}$

Message 2: AAy -> AS: $\{M\}_{Kx}, \{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km},$

$\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 3: AS -> AAy: $\{AAx\ \textbf{says}\ M\}_{Ky}$

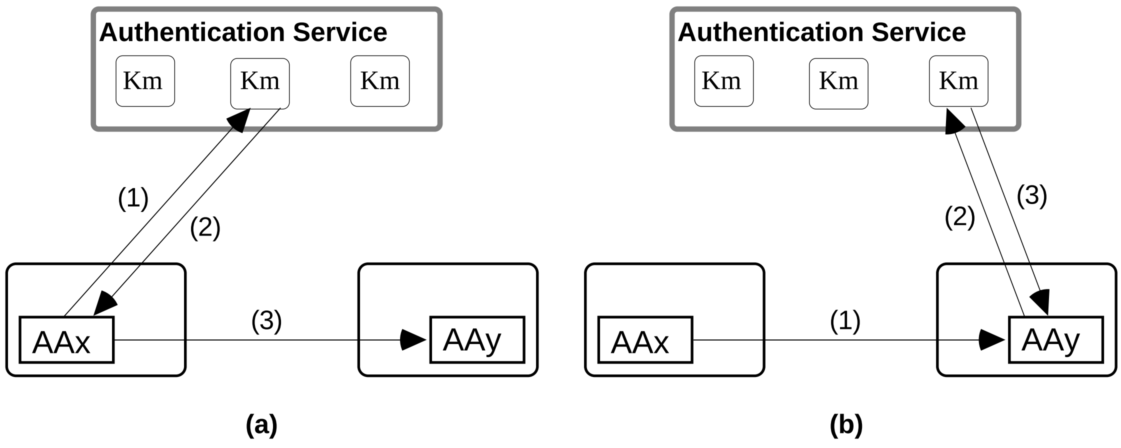


Figure 7.3 Message Translation by the Authentication Service

The second method is preferable when the sender does not know the identity of the receiver. This is the case when an AA signs membership certificates or delegation tokens that have to be forwarded to multiple verifiers.

These protocols do not provide message secrecy. Anyone can request translation of an encrypted message. Secrecy, however, can be achieved by including the identity of the receiver in the original encrypted message. The authentication server will only re-encrypt that message with the secret key of the AA whose identity is mentioned in the encrypted message. Further, a timestamp can be added in order to enable the receiver to verify the freshness of the message.

7.3.5 Certificate Revocation

The use of the certificate expiration time provides a simple means of revoking certificates. In some cases, however, it is important to provide a more drastic revocation mechanism. Specifically, a drastic revocation mechanism is required when the secret key of an AA has been compromised and thus the use of the corresponding certificate before its expiration time may enable the intruder to masquerade as that AA.

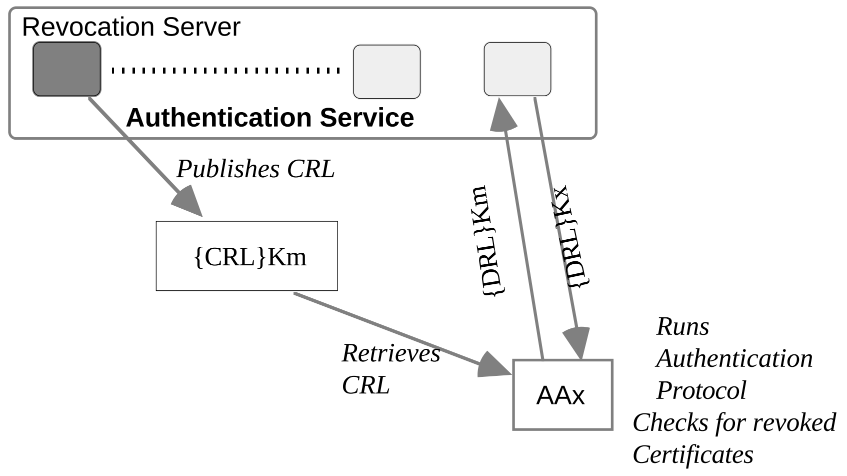


Figure 7.4 Translation of a CRL

A server involved in an authentication protocol or message translation does not know whether the certificates it uses have been revoked or not. We can, however, move the task of rejecting revoked certificates from the authentication servers to the AAs. The AAs base their beliefs on statements issued by the AS. The correctness of these statements depends on a number of private-key certificates which are processed by the AS. The AS can therefore associate each statement with the certificates on which it is based. The AA that receives a statement along with the identifiers of the certificates it is based on, can check whether these certificates are still valid provided that it has access to a fresh Certificate *Revocation List* (CRL). A CRL contains the identifiers of the certificates that have been revoked but have not expired.

CRLs should be created by a trusted server and used in a way that preserves their integrity. In order to achieve this, one server, referred to as the *Revocation Server*, is dedicated to the publication of the CRL. A single server is adequate as we do not expect a high revocation rate and one of the other servers can take over the revocation function should the first one fail.

The revocation server periodically publishes a CRL containing the CIDs and the expiration times of the certificates that have been revoked but have not expired, as well as a timestamp. The CRL is encrypted with the master key *Km* which is known to the revocation server. An AA can request translation of the latest CRL as shown in Figure 7.4.

The authentication protocol discussed in Section 7.3.3 has to be changed slightly in order to cope with revoked certificates. The statements that the AS generates should contain a *Certificate Dependence List* (CDL). This CDL contains the CIDs of the certificates on which the statement is based. In the protocol given below, the CDLs in the statements that are addressed to *AAx* and to *AAy* are *(CIDy)* and *(CIDx)* respectively. The two authentication agents should check these CIDs against the current version of the CRL in order to reject the statement if it is based on a revoked certificate:

Message 1: AAy -> AAx: $\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 2: AAx -> AS: $\{CID_x, Te_x, AAx \xleftrightarrow{Kx} AS\}_{Km}$, $\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 3: AS -> AAx: $\{CDLy, Ts, Te_{xy}, AAy, Kxy\}_{Kx}$, $\{CDLx, Ts, Te_{xy}, AAx, Kxy\}_{Ky}$

Message 4: AAx -> AAy: $\{CDLx, Ts, Te_{xy}, AAx, Kxy\}_{Ky}$, $\{AAx, Tx\}_{Kxy}$

Message 5: AAy -> AAx: $\{Tx\}_{Kxy}$

For example, in message 4, *AAy* receives the message

$$\{CDLx, Ts, Te_{xy}, AAx, Kxy\}_{Ky}$$

which indicates that the key *Kxy* can be used for secure communication with *AAx*. This statement has been encrypted with the key of *AAy* (*Ky*) and *AAy* believes that the statement has been made by the AS as only an authentication server can retrieve *Ky* from the corresponding private-key certificate. If the certificate *CIDx* mentioned in the *CDLx* is not in the current CRL, *AAy* believes that it can use the key *Kxy* for secure communication with *AAx* and replies with message 5 in order to complete the protocol. On the other hand if *CIDx* is in the current CRL, *AAy* cannot trust the key *Kxy* to speak for *AAx* and aborts the protocol.

A similar method can be used for the message translation protocols. For instance, the second message translation protocol presented in Section 7.3.4 can be enhanced as follows:

Message 1: AAx -> AAy: $\{M\}_{Kx}, \{CID_x, Te_x, AAx \xleftrightarrow{Kx} AS\}_{Km}$

Message 2: AAy -> AS: $\{M\}_{Kx}, \{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$, $\{CID_y, Te_y, AAy \xleftrightarrow{Ky} AS\}_{Km}$

Message 3: AS -> AAy: $\{CDL, AAx\ \textbf{says}\ M\}_{Ky}$

The CDL now contains the certificate identifier *CIDx* and the receiver checks the current CRL to determine whether the corresponding certificate has been revoked. If not, it establishes *AAx* ***says*** *M*.

The revocation server publishes a new version of the CRL when it receives a new revocation request. In addition, it publishes a new version of the CRL periodically (even if it is empty or identical to the previous version), for instance every hour. In this way, denial of service attacks can be detected and counter measures can be taken according to the security policy applied to the system.

In order to construct the new version of the CRL, the revocation server obtains and decrypts the latest published version, removes the CIDs of the revoked certificates that have expired and adds the CID of the newly revoked certificates (if any). The revocation server maintains the time the latest CRL was published, so it can ensure that the new CRL is always based on its previous version.

7.4. Secure Channels

As was mentioned in Chapter 5, the AAs deal with the authentication and cryptographic aspects of the secure channel establishment between application objects. That is, they authenticate remote AAs and provide the CFs in the address space of the application objects with keys to be used for secure communication.

7.4.1 Communication Security

The level of security a secure channel provides depends on the application. For many applications the provision of data integrity is sufficient while provision of secrecy may not be required for efficiency or legal reasons. In general, secure channels should be able to provide:

- *Integrity (I)*. The operation name and the arguments the target receives are those sent by the subject.
- *Integrity (II)*. The reply received by a subject is the one sent by the target.
- *No Replay (I)*. The target receives the invocation issued by the subject at most once.
- *No Replay (II)*. The subject receives the reply sent by the target at most once.
- *Secrecy(I)*. The identities of the principals that can obtain the operation name and the arguments of an invocation are known.
- *Secrecy(II)*. The identities of the principals that can obtain the reply data of an invocation are known.

The application level should be able to control the level of security required for a particular secure channel. We assume that the system can provide four different levels of communication security:

- *No Authentication*. When the channel is established, no authentication takes place. There is no guarantee for message integrity or secrecy.
- *Authentication Only*. The objects using the channel are authenticated but no cryptosystem is used to provide message integrity or secrecy
- *Message Authenticity*. The peer objects are authenticated and a cryptosystem is used to provide *Integrity I-II* and *No Replay I, II*.
- *Message Authenticity and Secrecy*. Message authenticity and *secrecy I-II* is provided by the channel.

The application level can specify the security level when it requests the establishment of a secure channel. If no security level is specified by the application, we assume that the system provides message authenticity as this should be sufficient for most commercial applications.

7.4.2 Channel Keys

Authentication agents negotiate cryptographic keys and cryptosystems to be used for secure channels between objects on their hosts. A single key could be used for all secure channels between two hosts provided that this secret key is not revealed to the servers using the channel and it is only used for encryption of a limited amount of traffic. Note that application servers, in general, are trusted to different extents even if they are on the same host. Thus, we have to prevent an application server from interfering with a channel which is normally used by another application server on the same host.

In our system, the channel keys are given to the CFs in the application address spaces, and thus we cannot use the same key for all the channels between two hosts.

This, however, do not exclude the use of the same key for multiple channels between objects in two given address spaces. It is not necessary, however, to use the same cryptosystem and key for the invocation and reply data. Two different cryptographic methods and keys providing different levels of security can be used for the invocation and reply data.

In the interests of simplicity, we assume that the same cryptosystem and key is used for the invocation and reply data and thus channels provide the same security level for both communication directions. On the other hand, a number of channels between objects on two given servers may use the same key if the required security level for these channels is the same.

Channel keys, however, can only be used for a limited amount of traffic and should be renewed frequently enough to prevent possible intruders gathering a large amount of data encrypted with the same key. The AAs that agree on the channel keys should also agree on an expiration time for each key and continuously provide the CFs with fresh keys for the period that a channel is valid.

Once two authentication agents have established a shared key *Kxy* by running the protocol described in Section 7.3.5, they can negotiate secret keys and cryptosystems to be used by a channel between objects on their hosts. A channel key can be computed by applying a hash function on two values generated by the two AAs. These two values should be transmitted under the cryptographic protection of the shared key *Kxy*.

This method, however, does not provide *perfect forward secrecy* (PFS). If PFS is required, a protocol that employs the Diffie-Hellman key exchange (Diffie and Hellman 1976) can be used. If the key exchange protocol does not provide PFS and an attacker manages to break the key *Kxy*, she will learn all the past and future channel keys provided that she records the messages that are exchanged between the AAs and are encrypted under the key *Kxy*. A detailed discussion on this issue is given in (Krawczyk 1996).

7.4.3 Setting up a Secure Channel

Figure 7.5 gives an illustration of the secure channel establishment process that is undertaken by the authentication agents of *ObjX* (subject) and *ObjY* (target). In the first phase the subject requests the establishment of a secure channel with a particular target object. The subject can define the security level required as well as other access-control parameters which are handled by the ACA and are discussed in Chapter 6.

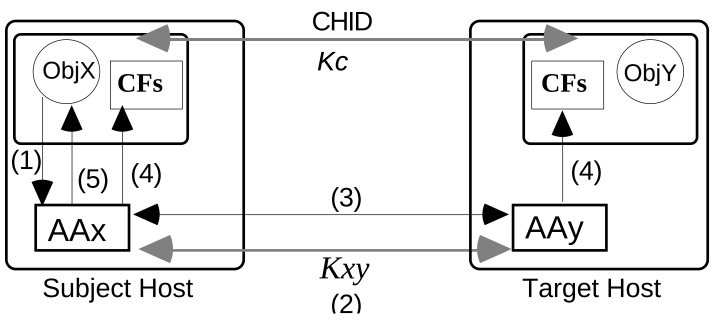


Figure 7.5 Authentication Agents Setting Up a Secure Channel

In phase (2), the two AAs establish a shared key *Kxy* by running the authentication protocol in Section 7.3.5, unless they have already done this in the framework of a previous channel establishment. In phase (3), the two AAs negotiate a cryptosystem to be used for the new channel, a channel key and a CHID. If the subject object requests a channel that does not provide message authenticity and secrecy, the AAs have to negotiate just a CHID as no cryptosystem has be employed to protect the messages transmitted on the channel.

In phase (4), the CFs in the address spaces of the two objects are given the CHID, the first version of the channel key *Kc* and a reference for the selected cryptosystem. Finally, in phase (5), the subject is given the CHID of the established channel.

7.4.4 Cryptographic Facilities

Once the CFs of the subject and target servers have been given the key and other cryptographic details of an established channel, the subject can invoke operations and receive replies that are protected according to the chosen security level of the channel.

Figure 7.6 illustrates the operation of the CFs when both message authenticity and secrecy is provided by an established channel. When the invocation data (which contain the identifier of the channel) have been marshalled, the cryptographic mechanism chosen for this channel is applied in order to provide data secrecy and integrity. The structure Cryptotable maps CHIDs into keys and cryptosystems and keeps track of the serial numbers. A serial number is added to the invocation data in order to allow the target CF to detect possible replays.

In Figure 7.6, the result of the encryption is denoted as:

$\{Marshalled\ data,\ SN\}K^{v}{}_{c}$

where $K^{v}{}_{c}$ is the current key of the channel. Note that there is a variety of cryptographic systems that can be used in order to achieve data integrity and secrecy. Here, we assume that the message and a hash value of it are encrypted under the key $K^{v}{}_{c}$. For example, DES and MD5 is a possible combination. The *serial number* (SN) contains a *direction bit* and its maximum value should be large enough to avoid the repetition of a number while the same key version is still in use. Each time the key changes, the serial number starts counting from zero. The direction bit is used to identify reflected messages.

The result of the encryption is sent to the target host along with a key identifier (*keyid*). The *keyid* is sent in plaintext and determines the key (and the cryptosystem) that the target CFs has to use in order to decrypt the message. The *keyid* consists of the CHID and the *key version* that is currently in use. Note that during the lifetime of a channel, many keys may be used in turn (key rollover). The CHID alone determines the cryptosystem that has been used to encrypt the message.

The target CF checks the integrity of the message that has been decrypted with the key specified by the *keyid*. If the check is successful, the CF believes that the message has been sent by the subject of the channel specified by the CHID sent in plaintext, and checks the SN to determine whether the message is a replay or a reflected one. Subsequently, it checks the CHID carried inside the message and if this matches the CHID that identified the decryption key, the marshalled data are forwarded to the unmarshalling routines.

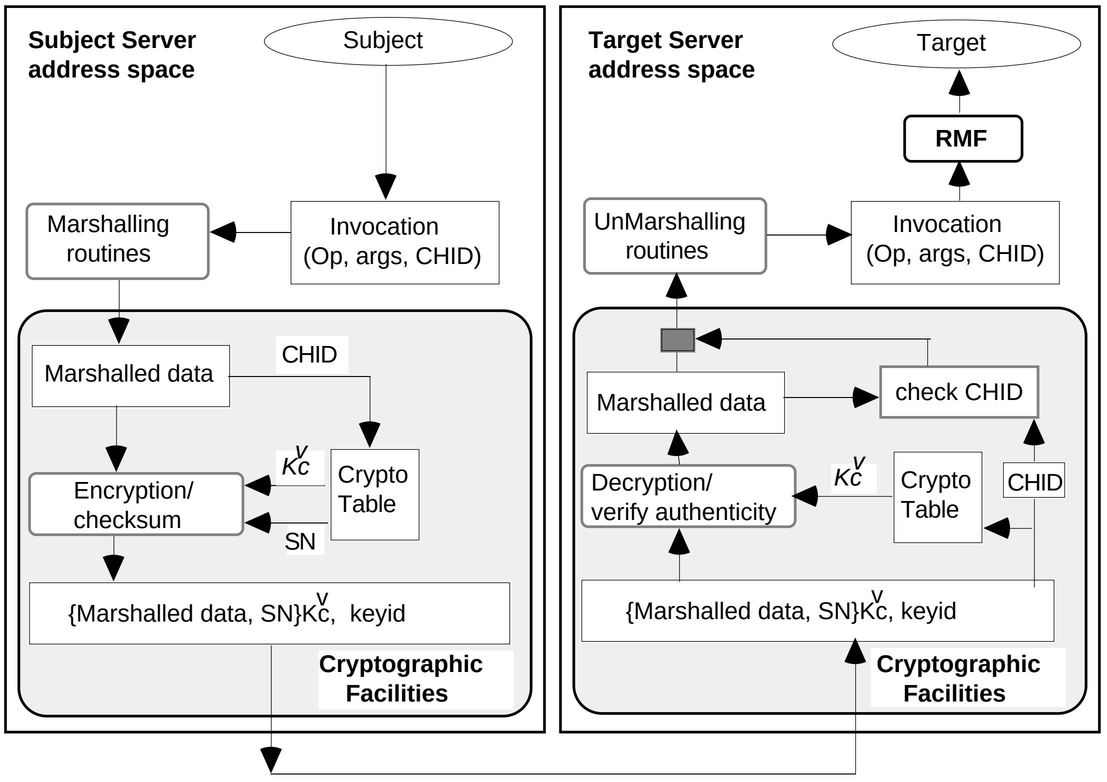


Figure 7.6 Invocation on a Channel Providing Authenticity and Secrecy

The unmarshalled invocation data are given to the RMF. This believes that the operation has been issued by the subject of the channel identified by the CHID in the invocation data. If the operation permitted by an access control policy in the RCPL of the specified channel, the operation is forwarded to the target object. Otherwise, the RMF raises a "*Unauthorised Access*" exception which is sent back to the subject.

When a reply has to be sent back to the subject, the CFs of the subject and target swap roles. That is, the target CFs encrypt the reply and the subject CFs decrypt it and check its authenticity.

7.5. Verification of Domain Membership

Membership verification is achieved by utilising the AS to translate membership certificates issued by the domain service. The AA of the claimant "pushes" the necessary certificates to the AA that acts as verifier. Membership certificates can be cached by the AA of the claimant and presented to multiple verifiers.

7.5.1 An Example

In the example illustrated in Figure 7.7, we assume that there is a policy *P1*, of which the subject scope is defined by the DSE **DomA*. This policy permits *ObjX* to invoke operations on *ObjY* but the ACA of *ObjY* needs evidence that *ObjX* is indeed a member of *DomA*.

Following the discussion in Chapter 6, the subject ACA determines the membership certificates required to prove that *ObjX* is in the subject scope of *P1*. In our example, these certificates are: *[ObjX < DomB]* and *[DomB < DomA]*. These two certificates form the capability certificate set (CCS) that has to be constructed by the subject AA. As *DomA* and *DomB* are OIDs, the subject AA can find out the host names of the domain servers maintaining these two domains. This implies that the AAs of these hosts can also be contacted by the subject AA. As illustrated in Figure 7.7, *DomA* is maintained by the domain server *DSa* while *DomB* is maintained by *DSb*. The AAs of these servers are *AAa* and *AAb* respectively.

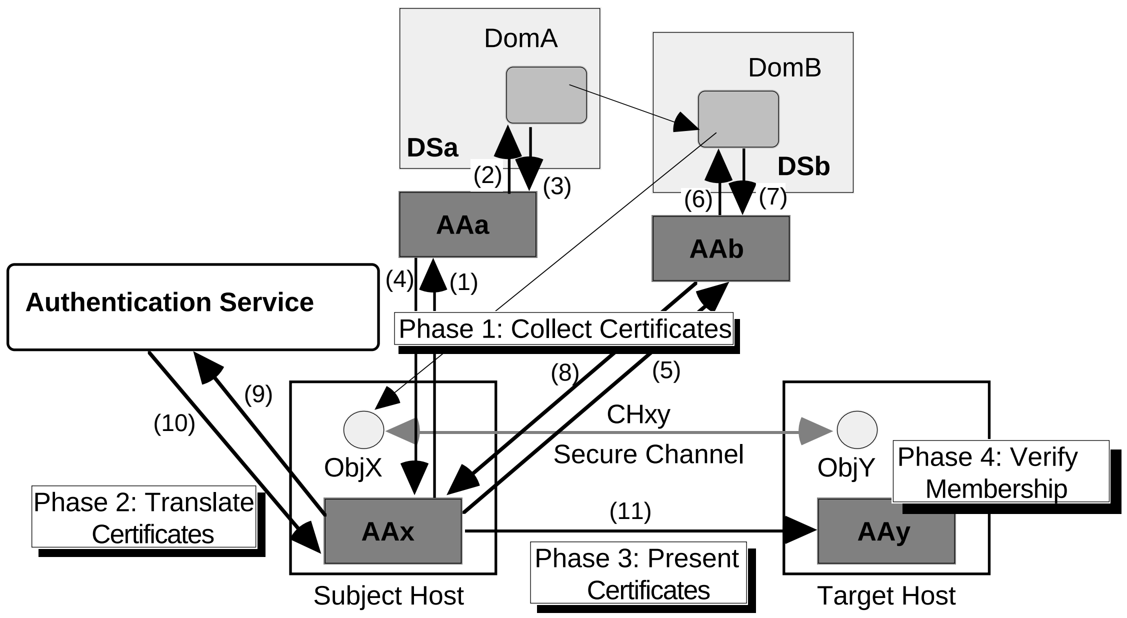
The domain membership verification process consists of four phases. In the first phase, *AAx* contacts the AAs of the domain servers and requests the necessary certificates. These certificates are encrypted under the keys of the AAs of the domain servers. So, at the end of the first phase, *AAx* possesses the membership certificates:

$$\{(Me1, DomB < DomA)\}_{Ka}$$
$$\{(Me2, ObjX < DomB)\}_{Kb}$$

where *Ka* and *Kb* are the keys of *AAa* and *AAb* respectively. These certificates, however, cannot be verified by *AAy* as it does not know the keys *Ka* and *Kb*. For this reason, the certificates are translated in the second phase by the AS along the lines described in Section 7.3.4. In this example, the translation is requested by the subject AA. Alternatively, the translation could be requested by the target AA.

Upon completion of the translation, the subject AA possesses the following "translated" certificates:

$$\{AAa\ \mathbf{says}\ (Me1, DomB < DomA)\}_{Ky}$$
$$\{AAb\ \mathbf{says}\ (Me2, ObjX < DomB)\}_{Ky}$$



Message 1. AAx -> AAa:	*Request certificate [DomB< DomA]*
Message 2. AAa -> DomA:	*Certify membership[DomB < DomA]*
Message 3. DomA -> AAa:	*(Me1, DomB < DomA)*
Message 4. AAa -> AAx:	$\{(Me1, DomB < DomA)\}_{Ka},$ $\{Te1, AAa \xleftrightarrow{Ka} AS\}Km$
Message 5. AAx -> AAb:	*Request certificate [ObjX< DomB]*
Message 6. AAb -> DomB:	*Certify membership[ObjX< DomB]*
Message 7. DomB -> AAb:	*(Me2, ObjX< DomB)*
Message 8. AAb -> AAx:	$\{(Me2, ObjX < DomB)\}_{Kb},$ $\{Te1, AAb \xleftrightarrow{Kb} AS\}Km$
Message 9. AAx -> AS:	$\{(Me1, DomB < DomA)\}_{Ka},$ $\{Te1, AAa \xleftrightarrow{Ka} AS\}Km,$ $\{(Me2, ObjX < DomB)\}_{Kb},$ $\{Te1, AAb \xleftrightarrow{Kb} AS\}Km,$ $\{Te1, AAy \xleftrightarrow{Ky} AS\}Km$
Message 10. AS -> AAx:	$\{AAa\ \textbf{says}\ (Me1, DomB < DomA)\}_{Ky}$ $\{AAb\ \textbf{says}\ (Me2, ObjX < DomB)\}_{Ky}$
Message 11. AAx -> AAy:	$\{AAa\ \textbf{says}\ (Me1, DomB < DomA)\}_{Ky}$ $\{AAb\ \textbf{says}\ (Me2, ObjX < DomB)\}_{Ky}$

Figure 7.7 Example of Domain Membership Verification

In the third phase, these certificates are given to the target AA which can now verify the requested membership (phase 4). Note that *AAy* can ensure that *AAa* is the authentication agent on the host of *DomA* and thus it trusts *AAa* to speak on behalf of

DomA. Similarly, *AAy* trusts *AAb* to speak on behalf of *DomB*. Consequently, *AAy* establishes that $(DomB < DomA)$ and $(ObjX < DomB)$.

The membership certificates contain expiration times. The period for which a certificate remains valid depends on the enforced security policy. In general, long certificate lifetimes increase the benefits of caching but decrease the security level in that it takes longer for a membership change to be realised by the verifier. The lifetime of the membership statements can be adjusted to comply with the security and efficiency requirements of the system. Furthermore, domains that certify the membership of dynamic objects with a short life cycle, can issue membership certificates with a shorter expiration time.

In this example, we have omitted the CDL that is normally inserted in the translated certificates in order to deal with PKC revocation. As discussed in Section 7.3.5, the verifier has to check the CDL against the current CRL to ensure that the translated membership certificates do not depend on revoked PKCs.

7.5.2 Domain Membership and Secure Channels

Verification of domain membership is accomplished during the establishment of a secure channel. In the example mentioned in Section 7.5.1, the verified membership *[ObjX<<A]* is associated with the channel *CHxy* between *ObjX* (subject) and *ObjY* (target). The verified membership of the subject is used by the target ACA to ensure that the policies in the CCPL of *CHxy* apply to the subject. The policies that are supported by the verified membership are inserted in the RCPL which is given to the RMF of the target.

The target AA maintains a list of certificates that support the RCPL of an active channel. When a certificate approaches its expiration time while the channel is still in use, membership re-verification is required by the target AA. If the re-verification fails, the target ACA is informed and the policies that depend on the expired membership are removed from the RCPL. The new RCPL is given to the RMF of the target object.

This implies that if the subject ceases to be a member of a domain, it can still use the rights inherited from that domain until the corresponding membership certificate expires.

7.6 Delegation of Access Rights

There is a need for a trustworthy mechanism that enables the security components of the end-point to verify that a delegation of access rights has indeed taken place. In this section we assume that the grantor, the grantee and the end-point are on different nodes. Delegation can also take place among objects that reside on the same host, but this can be handled by the local AA and there is no need to use the AS.

7.6.1 Delegation Protocols

This section describes the protocol employed to verify delegation. This protocol involves the AAs of the grantor, the grantees and the end-point as well as the AS. Figure 7.8 illustrates the protocol sequence for one step delegation.

The protocol is based on the use of *delegation tokens* (DTs). Each delegation step is represented by a DT of the form:

<DID, PDT, Tde, DAR, Grantor, Grantee>

where:

- *DID* is the Identifier of the token which is created by the AA of the grantor using its host name plus a time stamp. So, a *DID* is globally unique.
- *PDT* is the DID of the Preceding Delegation Token and it is used when cascaded delegation takes place.
- *Tde* is the expiration time of the delegation. We assume the clocks of the AAs are loosely synchronised.
- *DAR* specifies the delegated rights.
- *Grantor* is the OID of the grantor.
- *Grantee* is the OID of the grantee.

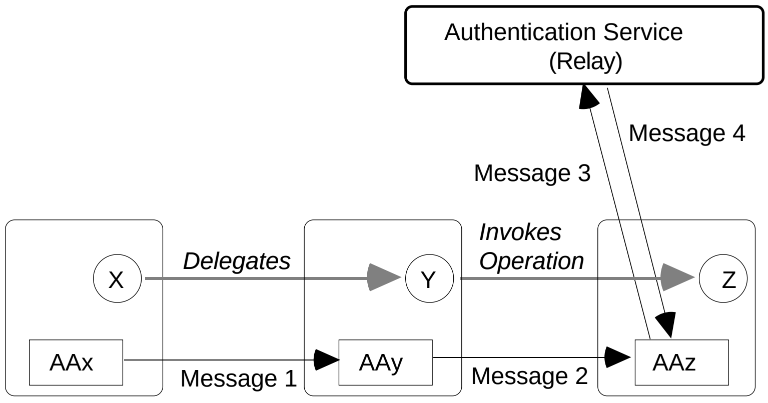
A DT is created by the grantor AA and it is encrypted under its key to enable the end-point to verify its integrity. As the end-point does not know the key of the grantor AA, a DT has to be "translated" by the AS. Note also that, in general, the grantor does not know the end-points that will verify a DT.

In Figure 7.8. the grantor AA encrypts the delegation token with its key and passes it to the grantee AA. Subsequently, the grantee AA passes it along with the PKC of the grantor to the end-point AA as proof that a delegation has taken place. The DT is passed to the AS for translation along with the PKCs of the grantor and end-point AAs (message 3). Alternatively, the translation of the DT can be requested by the grantee AA. In message 4, the end-point AA receives the DT encrypted under its key and establishes that *X* has delegated *Y*. Note that the end-point AA has to ensure that the AA that has signed the DT is trusted to speak for object *X*, that is, the host name in the OID of *X* matches that in the OID of *AAx*.

In addition, the grantor AA sends to the grantee AA the same DT encrypted with the shared key that has been established between the grantor and the grantee AAs (Figure 7.8, message 1). The grantee AA can decrypt this token and verify its integrity. Alternatively, the DT can be sent in plaintext along with a MAC to ensure its integrity. In this way the grantor is aware of the correct delegation details. This is important because a corrupted DT in plaintext may mislead the grantee which may forward the wrong access rights to a subsequent grantee. Another way to achieve this is to translate the encrypted

DT *{DT}*$_{Kx}$ into the key of *AAy*. This, however, involves additional communication cost as well as encryption/decryption that has to be performed by the AS.

Finally, the end-point AA can ensure that the translation of the DT is not based on a revoked PKC by checking the current CRL along the lines described in Section 7.3.5.



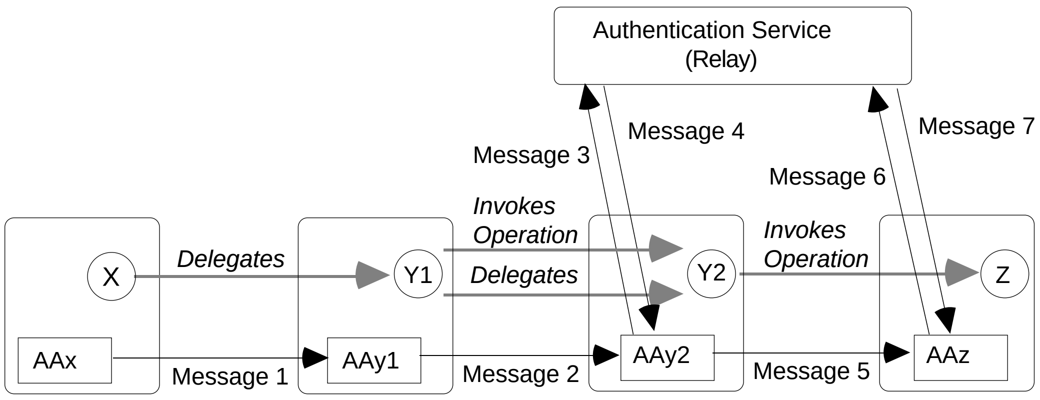
Message 1. AAx -> AAy: *{DT}Kxy, {DT}Kx, PKCx*
(delegate-send DT)
Message 2. AAy -> AAz: *DT, {DT}Kx, PKCx*
(claim delegation-forward DT)
Message 3. AAz -> AS: *{DT}Kx, PKCx, PKCz*
(request translation of DT)
Message 4. AS -> AAz: *{CDL, AAx* ***says*** *DT}Kz*
(receive translated DT)

$$\mathbf{DT} = \langle DID, DAR, Tde, X, Y\rangle,\ \mathbf{PKCx} = \{Te1, AAx \xleftrightarrow{Kx} AS\}Km$$

$$\mathbf{PKCz} = \{Te2, AAz \xleftrightarrow{Kz} AS\}Km$$

Figure 7.8 One Step Delegation Protocol

In the case of cascaded delegation, the AA of each intermediate grantor encrypts a DT which is sent to the next grantee along with all the delegation tokens that refer to the preceding delegation steps. In addition, all the delegation tokens are sent encrypted with the key shared with the next grantee. Figure 7.9 illustrates the protocol sequence in a two-step cascaded delegation.



Message 1. AAx -> AAy1: *{DT1}Kxy1, {DT1}Kx, PKCx*
(delegate-send DT1)
Message 2. AAy1 -> AAy2: *{DT1, DT2}Ky1y2, {DT1}Kx, {DT2}Ky1, PKCx, PKCy1*
(claim delegation-forward DT1, delegate-send DT2)
Message 3. AAy2 -> AS: *{DT1}Kx, PKCx, PKCy2*
(request translation of DT1)
Message 4. AS -> AAy2: *{DT1}Ky2*
(receive translated DT1)
Message 5. AAy2 -> AAz: *DT1, DT2, {DT1}Kx, {DT2}Ky1, PKCx, PKCy1*
(claim cascaded delegation-forward DT1, DT2)
Message 6. AAz -> AS: *{DT1}Kx, {DT2}Ky1, PKCx, PKCy1, PKCz*
(request translation of DT1 and DT2)
Message 7. AS -> AAz: *{CDLx, AAx* ***says*** *DT1}Kz, {CDLy, AAy* ***says*** *DT2}Kz*
(*receive translated DT1 and DT2)*

DT1 *= <DID1, -, Tde1, X, Y1>, DT2 = <DID2, DID1, Tde1, Y1, Y2>*

$$\textbf{\textit{PKCx}} = \{Te1, AAx \xleftrightarrow{Kx} AS\}Km,$$

$$\textbf{\textit{PKCy1}} = \{Te2, AAy1 \xleftrightarrow{Ky1} AS\}Km$$

$$\textbf{\textit{PKCy2}} = \{Te3, AAy2 \xleftrightarrow{Ky2} AS\}Km,$$

$$\textbf{\textit{PKCz}} = \{Te4, AAz \xleftrightarrow{Kz} AS\}Km$$

Figure 7.9 Two Step Delegation Protocol

The PDT field contains the DID of the preceding DT and is used to prevent misuse of delegation tokens in cascaded delegations. If this field is omitted, it is possible that a grantee can claim a false delegation chain (Varadharajan et al. 1991). For instance, consider the scenario illustrated in figure 7.10. *X1* delegates to *Y1* (token *DT1*) which subsequently forwards the delegated rights to *Y2 (DT1, DT3)*. In addition, *X2* delegates to *Y1 (DT2)*. If *DT3* does not specify the token that represents the preceding delegation step (i.e. *DT1*) *Y2* may claim that it has been delegated by *X2* via *Y1* by forwarding *DT2* and *DT3* to *Z* (end-point). Note that *DT2* can easily be obtained by *Y2* using wiretapping.

Another possible attack is illustrated in figure 7.11. Initially *X1* delegates to Y1 using *DT1* which is identified by *DID1*. Subsequently, *X2* delegates to *Y1* using *DT2* which is also identified by *DID1*. If *Y1* does not notice the repeated delegation identifier, it can forward to *Y2* the access rights delegated by *X2* using the delegation token *DT3* which mentions *DID1* in the PDT field. Though *Y1* uses *DID1* to refer to *DT2*, *Y2* can forward *DT1* to the end point (*Z*) in order to claim the delegation chain *X1->Y1->Y2*. This attack is avoided if the grantee checks whether the host name in the DID matches the host name of the AA that encrypted the token. We assume here that a grantor AA is trusted not to repeat the DIDs it generates.

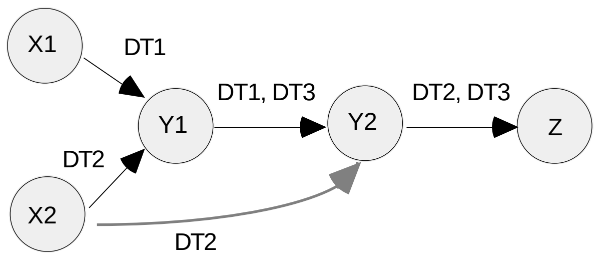
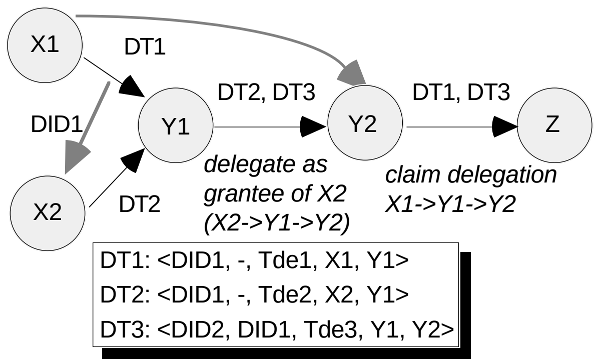


Figure 7.10 Misuse of Delegation Token if the PDT Field is Omitted


X1
DT1
DT2, DT3
DT1, DT3
DID1
Y1
Y2
Z
delegate as grantee of X2 (X2->Y1->Y2)
claim delegation X1->Y1->Y2
X2
DT2

DT1: <DID1, -, Tde1, X1, Y1>
DT2: <DID1, -, Tde2, X2, Y1>
DT3: <DID2, DID1, Tde3, Y1, Y2>

Figure 7.11 Attack on the Delegation Protocol Using Repeated DIDs

The expiration time of the delegation tokens provides a simple revocation mechanism. However, a more drastic revocation mechanism, based on the idea of the revocation server, is described in the next section.

7.6.2 Revocation of Delegation Tokens

The revocation server is employed to periodically publish a *Delegation Revocation List* (DRL) which contains the DIDs of the DTs that have been revoked but have not expired. An end-point AA can obtain the latest DRL, translate it, and check whether the presented DTs have been revoked.

Typically, a grantor revokes a DT if it discovers that a grantee misuses it or if there is no reason for the represented delegation to continue to be valid. The grantor AA requests the revocation server to add the DID of the revoked DT in the DRL by sending a message encrypted under its secret key. This is illustrated in Figure 7.12. Note that the grantor AA has also to pass its PKC to enable the revocation server to retrieve its key. For a revocation request to be valid, the host name of the DID should match the host name of the grantor AA requesting the revocation.

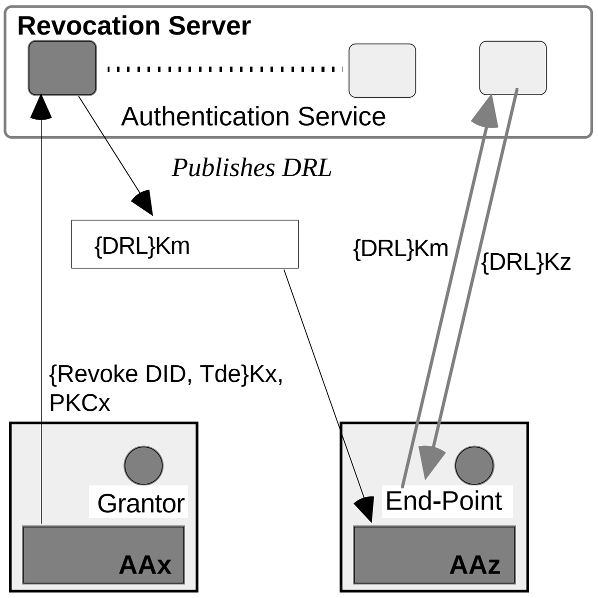


Figure 7.12 Revocation of a Delegation Token

7.6.3 Restricted Access Rights

We have shown, in Chapter 6, how a grantor can restrict the access rights it delegates by using PCSEs. These access rights are represented by the DPCL specified by the grantor ACA. This list is given to the grantor AA which inserts it in the DAR field of the corresponding DT, so that the end-point can ensure that the grantor has indeed delegated these access rights.

The access rights that an intermediate grantor forwards to a subsequent grantee are determined by the GPCL. Similarly, this list is inserted in the DAR field of the DT representing the cascaded delegation.

In addition, the DAR field may specify further restrictions on the delegated access rights, such as maximum number of invocations, time restrictions, etc. These restrictions are interpreted by the ACA and RMF of the end-point.

7.6.4 Discussion and Related Work

Following the discussion in Chapter 4 about delegation, we can show that our delegation scheme achieves the six goals set out in (Sollins 1988):

- *Unforgeability*. Verification of each delegation step is achieved by checking the corresponding DT. Since a DT is signed with the key of the grantor AA and is translated into the key of the verifier by the AS, the end-point AA can ensure that all delegation steps in a claimed delegation sequence have taken place.
- *Accountability*. Each delegation step contains the identities of the grantor and grantee. Also, in the case of cascaded delegation, each DT is chained to the one representing the preceding delegation step. So, the end-point can track the route of cascaded invocations and delegations.
- *Discretionary Restriction*. The initial grantor as well as each intermediate, can restrict the access rights it delegates. The application level can specify the access rights it delegates by using PCSEs. In addition, it is possible that further restrictions on the access rights can be imposed. These are included in the corresponding DTs so that the end-point knows the exact delegated rights the final grantee in a delegation chain can use.
- *Modularity*. Each grantor can delegate access rights without knowing how the grantee will use these rights. In addition a grantor does not have to know the identities of the end-points that will verify a signed DT.
- *Independence*. The end-point can verify a delegation chain by requesting translation of the corresponding DTs and it does not have to contact the grantors.
- *Combining of Identity*. The end-point can identify all the principals involved in a delegation sequence and permit access only if there are extended access rules permiting that sequence.

It is also important that our system supports delegation of DTs. A grantor can revoke a delegation before the corresponding DT expires. This is of crucial importance when a grantee misuses the delegated access rights. It is interesting to compare this approach with the revocation scheme employed in the Distributed Systems Security Architecture (DSSA) and is described in (Gasser and McDermott 1990).

The DSSA delegation mechanism is based on the use of asymmetric cryptography. The initial grantor is always a user that possess a smart card capable of storing a private key and performing asymmetric cryptographic operations. A user uses its smart card to delegate to a workstation which can further delegate the user's access rights to another workstation, etc. The smart card uses the private component of the key associated with the user and certified by a CA, to sign a *delegation certificate* to delegate to a workstation *WS1*. The delegation certificate associates the identity of the workstation with a public key *K1* that has been generated by the delegated workstation. This key is referred to as the *delegation key*. The workstation can use the delegated access rights if it can prove that it knows the private component of *K1*. In addition, *WS1* can further delegate the user's access rights by using the private component of its delegation key to sign a delegation certificate that associates the identity of the grantee workstation *WS2* with its public delegation key *K2*. Again, *WS2* has to prove that it knows the private component of *K2* in order to use the delegated access rights.

When a delegation chain is no longer needed, the workstations that have been delegated by the user erase the private components of their delegation keys. This protects against misuse of the delegated access rights by a workstation that is compromised after its private delegation key has been destroyed. Note that even if a compromised workstation can present a delegation certificate that mentions its identity, it would be able to prove knowledge of the associated private delegation key.

This implies that a user can revoke a delegation if the delegated workstations are sufficient trustworthy to destroy their private delegation keys. A user cannot revoke a delegation if a delegated workstation fails to destroy its delegation key.

Our revocation scheme is based on the assumption that the revocation server is trusted to include the DID of a revoked delegation token in the DRL. It is not based on the faithfulness of the grantees. A delegation chain can be revoked even if one or more grantees are not trusted to revoke the access rights they have been delegated.

Another disadvantage of the DSSA delegation scheme is that a workstation has to generate a private/public key pair for each delegation. Since generation of these keys is computationally intensive, this may impose a considerable overhead if a large number of delegations take place. This overhead, though, may be alleviated if these keys are generated in advance of any actual need to use them.

Other delegation schemes that are based on symmetric cryptography are described in (Sollins 1988) and (Varadharajan et al. 1991). (Sollins 1988) proposes a delegation mechanism based on *nested* tokens named *passports*. Each delegation step is represented by a passport signed by the grantor. The grantor signs the passport with a key that it shares with a trusted authentication server. If a passport represents cascaded delegation, it includes the preceding passport. The end-point cannot directly verify the presented passports as it does not know the keys of the grantors. The verification of the passports is accomplished by the authentication server which shares a key with the end-point.

A similar protocol is described in (Varadharajan et al. 1991), but the authors state that the nesting of the tokens can be avoided if a delegation token includes the unique identifier of the predecessor token. This is similar to the approach that has been adopted in our delegation scheme. Our approach, though, is different in that trusted servers with minimal state are used, so that we can easily replicate them and achieve higher availability and performance.

7.7 Chapter Summary

We have shown how an authentication system that is based on symmetric cryptography and employs trusted servers with minimal state can be used to provide authentication as well as verification of domain membership and delegation of access rights. The servers disperse their security database by publishing Private-Key Certificates. Replicated servers can be used in order to increase the availability and performance of the authentication service. The use of symmetric cryptography reduces the overhead when membership certificates and delegation tokens are issued or verified.

Authentication Agents on each host provide the cryptographic facilities of the application objects with keys to be used for secure communication. In addition they verify the domain membership of subjects and grantees in order to enable the target ACA to determine the policies that allow the accessor to invoke operations on a secure channel. Domain membership verification is based on encrypted membership certificates that are issued by the domain service. These are translated by the authentication service into the keys of the AAs that act as verifiers.

The system employs the “push” method in order to provide the AA of the end-point with the necessary membership certificates. This reduces the task that has to be undertaken by the security components of the end-point. In addition, it enables the AA of a subject or grantee to cache membership certificates and use them to establish secure channels with multiple targets.

Each delegation step is certified by a delegation token encrypted by the AA of the grantor. The delegation tokens are also translated by the AS into the keys of the end-point AAs. Since a grantor may restrict the access rights it delegates, a delegation token specifies the delegated access rights and it may also impose special restrictions on their use. The information contained in a delegation token is used by the end-point ACA to determine the access control policies applying to a grantee that requests establishment of a secure channel along the lines discussed in Chapter 6.

The system also supports revocation of Private-Key Certificates and delegation tokens. This is achieved by utilising a revocation server that periodically publishes encrypted revocation lists. These can be translated by the authentication service into the keys of the AAs which can ensure that the execution of an authentication protocol or the verification of a domain membership is not based on revoked certificates. Similarly, an end-point AA can ensure that a delegation token has not been revoked.

Chapter 8

Conclusions

The work presented in this thesis has been motivated by the need to enforce access control policy specified in terms of domains in distributed systems. The thesis has built on the notions of domains and access control policies which have been developed and used in the Esprit-funded SysMan project. In this chapter we review the aims discussed in Chapter 1 to examine how each has been achieved. The main aspects of the proposed system are criticised and directions for further work are discussed.

8.1 Review of Aims and Accomplishments

The thesis has presented the basic design of a security architecture that supports domain-based access control. The fact that the policy scopes are specified in terms of domains facilitates the policy specification from the administrator's point of view. Instead of specifying policies for individual objects, the administrator does this for groups of objects. This, however, complicates the access control and authentication mechanisms. We have shown how these two mechanisms can co-operate with the policy and domain services to provide policy dissemination and access control in a distributed environment.

The integration of cryptographic and authentication mechanisms into the proposed architecture protects against malicious actions which could lead to communication compromise. In particular, the system protects against attacks that could result in the enforcement of the wrong access control policy, the invocation of operations by unauthorised principals and optionally can protect against unauthorised access to information transmitted over physically unprotected networks.

The Orbix™ environment has been used as a distributed platform to implement the policy dissemination mechanism described in Chapter 6 and the Appendix. This has been integrated with an existing domain service and a policy service which enables administrators to create, edit, activate and retract policies using a graphical interface. A simple Access Control Agent has also been implemented on Orbix™. This is capable of determining the access control policies applying to the objects on its host as well as determining the policies applying to an established secure channel. This ACA has been used along with the policy dissemination mechanism, the domain service and the policy service to demonstrate the specification, dissemination and enforcement of access control policies for an example bank application in the framework of the SysMan project. This simple ACA, however, does not cope with delegation of access rights and the authentication service to support access control decisions was not implemented.

8.1.1 Delegation of Access Rights

The notion of an access control policy as defined in the framework of the SysMan project does not cope with delegation of access rights. There was a need to extend it in order to enable the administrators to control the delegation of rights when cascaded invocations take place. The notion of an *extended access policy* allows a security administrator to clearly specify, using domains, to whom rights can be delegated. This scheme does not cater for the order of delegation hops, c.f. (Abadi et al. 1993), but we have argued that this is acceptable for most applications and simplifies the policy specification from the administrator's point of view.

We have shown how extended access control policies can be distributed to the relevant Access Control Agents (ACAs). The access control decision making when delegation takes place is supported by the use of pseudo-capabilities which reduces the computational effort required by the target ACA. The authentication system supports verification and revocation of delegation.

Our specification of delegation does not cater for constraints such as maximum delegation hops or maximum delegation period. It may also be beneficial that the administrator can impose restrictions on the values of some operation arguments. For instance, the grantee delegated by an account owner cannot debit more than 100 pounds. There is a need to identify the types of delegation constraints needed in practical applications and work out an appropriate specification language.

It has been shown that the grantor can delegate restricted access rights. These are specified in terms of a subset of the domains of which the grantor is a member. Though this is quite flexible, it is interesting to examine how this could be combined with the notion of a role. Following the discussion in Chapter 4 about roles and delegation, it is convenient for many applications to use roles to refer to a set of access rights. Research on the notion of a role in a domain framework is currently being carried out (Lupu and Sloman 1996; Lupu et al. 1995) and it has been shown that roles can be supported without changing the main aspects of the architecture (Yialelis et al. 1996).

8.1.2 Policy Distribution

There is a need to disseminate policies to the access control components. We have extended the policy propagation mechanism described in (Twidle 1993). Our mechanism copes with policy scopes that are defined in terms of set operations and with domain structures that change dynamically. We have shown how the Access Control Agents can determine that a change in the domain structure affects the applicability of policies to the objects on their hosts. We have identified the need to provide a monitoring mechanism that reports policy dissemination problems which may occur due to communication failures. This mechanism enables the security administrator to be aware of the policy enforcement status and take the necessary counter measures should a problem arises.

It is worth trying to investigate possible improvements such as storage of active policies within system components other than the Access Control Agents. This may be beneficial for Access Control Agents that are not physically close to the policy server and thus communication overhead can be avoided when a policy is being fetched from a local policy repository rather than the remote policy server.

8.1.3 Access Control Enforcement

The access control mechanism is based on the ACL paradigm and is supported by pseudo-capabilities used as hints. This mechanism can also cope with cascaded delegation. We have shown how the pseudo-capabilities can be used as hints to reduce the policy space that has to be searched by the target access control facilities when a secure channel is established. This is important when a host supports a large number of targets that accept invocations from multiple subjects which have a large parent domain structure.

A high degree of access control transparency is provided to security unaware applications in that the security system can control access to operations supported by the target’s interface. However, the notion of a secure channel is visible to the application level. Should the need arise, an object can retrieve access control, domain membership and authentication information related to a channel, in order to make further application dependent access control decisions.

8.1.4 Authentication

We have used an authentication scheme that is based on servers with minimal state and symmetric cryptography. This supports identity authentication, verification of domain membership and delegation. The use of Private-Key Certificates facilitates the server replication (Davis and Swick 1990). Replicated servers can be used to increase the performance and availability of the authentication service. Furthermore, servers that maintain minimal state can be implemented as simple tamper-proof machines. These machines can also utilise special hardware to perform the cryptographic operations.

We do not use asymmetric cryptography to avoid the cryptographic overhead which would dramatically affect the efficiency of the system, particularly when the domain service has to issue a large number of membership certificates and there are servers that establish a large number of secure channels with multiple remote subjects.

On the other hand, the security of the system depends on the master key. If the master key is compromised, the security of the whole system is compromised. So the security of the system is weaker that in a system based on asymmetric cryptography. To make it more difficult for an intruder to steal the master key, this should be used for a limited period of time. This requires a scheme that enables the replicated servers to agree on new versions of the master key periodically.

The system also provides revocation of private key certificates which is required when the key of a host is stolen or a host is compromised. The revocation mechanism is vulnerable to denial of service attacks but these can be detected by issuing the revocation list periodically. We can increase the resistance of the system to these attacks by utilising redundant revocation servers to issue the list. This redundancy will not affect the performance of the system as we expect a low revocation rate.

Authentication agents are trusted to represent objects on their hosts in terms of authentication, so our scheme reduces the cryptographic and authentication overhead compared to the system which is described in (Deng et al. 1995) and generates a public/private key pair for each individual object.

There is a need to extend the authentication scheme so that inter-realm authentication is supported. (Davis and Swick 1990) discusses possible solutions for inter-realm authentication using relays. There are many details that have to be worked out in order to provide verification of domain membership and delegation over multiple realms.

Furthermore, there is a need to reconsider the naming of objects. Note that in a single realm, there is one well-known authentication authority that is trusted to authenticate all objects. If we assume multiple realms, the OID of an object should indicate the authentication authority that is trusted for authentication of that object.

8.1.5 Security Architecture

The security architecture integrates the policy, domain, access control and authentication services. Distributed security is provided by security agents employed on each host and security facilities that are linked as a library to the application servers. The use of the security agents reduces the security components and information that have to be replicated amongst application servers. A secure channel represents cryptographic, authentication and access control information associated with a subject-target pair.

An interesting issue is the integration of the system with existing standards such as the Common Object Request Broker Architecture (CORBA) defined by the Object Management Group (OMG). In particular, it is interesting to examine how the security agents could be implemented inside the ORB. (OMG 1995) discusses the provision of security services at the ORB level and the use of interfaces that enable security aware objects to handle security-related information and deal with a wide range of security facilities such as access control, authentication, auditing, non-repudiation and delegation. The proposed model does not depend on specific access control and authentication mechanisms. It is sufficiently general to support a variety of security mechanisms. It is possible that this model could be adopted in order to implement our architecture and provide the necessary APIs without changing the main aspects of the authentication and access control mechanisms described in this thesis.

8.2 Performance Issues

The provision of security affects the performance of the system. In the proposed system the cryptographic overhead is reduced by using symmetric cryptography and utilising authentication agents on each host to avoid unnecessary authentication operations. The design of the system has been influenced by a number of assumptions concerning its usage on real applications which we outline below. We give some numerical estimates of the system performance in typical cases and compare these with the estimated performance of a hypothetical system based on asymmetric cryptography. We also discuss how the adoption of the push method can decrease the channel establishment cost.

8.2.1 Usage Assumptions

One major assumption is that the object population and the domain structure change at a high rate. In a distributed object system we expect that many dynamic objects with a short life cycle are created continuously. This is also true for domain objects at the lower levels of the domain structure. Many tasks may require the creation of domains in order to group objects associated with these tasks. On the other hand, we expect that the higher levels of the domain structure would be relatively static.

The dynamic nature of the domain structure and the object population implies that a large number of new certificates have to be created by the domain service. This is for two reasons. The domain service has to create new certificates to reflect the creation of new membership. What is more, the certificates stating the membership of object that are expected to have a short life cycle should have a short expiration time.

The use of asymmetric cryptography to sign membership certificates would impose a significant overhead on the hosts of the domain servers. Asymmetric cryptography can be 100 to 1000 times slower than symmetric cryptography when they are both done in software. Other systems that support group membership verification, for example the Taos OS (Wobber et al. 1993), utilise an off-line CA that uses asymmetric cryptography to sign membership certificates. In these systems, however, the group structure is expected to change at a low rate and the certificates have long expiration times.

Another assumption is that a large number of secure channels between objects have to be established. Some of them would have a long life cycle. However, the dynamic nature of the object population implies that a considerable number of channels would have a short life cycle. It is important that the establishment cost of these channels is as small as possible. Also, it is expected that some servers, for instance file servers, will have to establish a large number of channels with multiple subjects. It is important that the channel establishment introduces a minimal processing overhead on the hosts of these servers.

Finally, it is expected that in many delegation cases the delegated rights are used for a small number of invocations. We also expect that delegation takes place often enough to consider the delegation-related delays as an important performance factor of the system.

8.2.2 Numerical Estimates

It is expected that cryptographic operations will increase the latency of invocations on a secure channel. In order to give some numerical estimates, we assume the use of a DES implementation capable of encrypting/decrypting 640KBytes/sec. The actual performance depends on the efficiency of the code and power of the processor. For example, a DECchip 21064a processor at 225MHz can achieve 990 Kbytes/sec (Doorn et al. 1996). A 80486 at 33MHz can achieve about 320KBytes/sec (Schneier 1994). Furthermore, we assume a MD5 implementation running at 10MBytes/sec (an implementation on the DECchip 21064a at 225MHz runs at 15MBytes/sec (Doorn et al. 1996)).

First we assume that the MD5 and a 64-bit key is used as it is depicted in Figure 2.6 on a channel that provides message authenticity but not secrecy (see Section 7.4.1). Upon an invocation, the MD5 has to be applied to the operation name, the arguments, the

CHID, the SN and the key (see Section 7.4.4 and Figure 7.6). If we assume a total size of 150bytes, we expect a delay of about 15μsecs for MD5 operations on the subject side plus 15μsecs on the target side. We should expect a similar delay for the reply which gives a round-trip delay of 60μsecs. We have measured an average round-trip remote invocation latency of 7 msecs on Orbix™ 2[1] . This is the case when the binding of the client to the server has been done. The binding delay is of the order of hundreds of msecs (we have measured delays that are more than 600 msecs).

In this example, the use of the MD5 hash function introduces a delay of about 1%. Although there are other ORB implementations that are faster, for example (Crane and Dulay 1997), and the invocation latency also depends on network delays, this example indicates that the use of the MD5 introduces an acceptable delay and can be used in applications that require data integrity.

If we assume that the DES is used along with the MD5 as it is depicted in Figure 2.7 on a channel providing both authenticity and secrecy, the DES algorithm has to be applied to about 200Bytes (invocation data + 128-bit digest) for a similar invocation. In this case we expect a round-trip delay due to DES and MD5 of about 1.3msec which reflects an increase of about 20% on the average invocation delay (see Table 8.1). Thus, the provision of secrecy does increase considerably the invocation latency.

	Insecure	Authenticity	Secrecy
Estimated round-trip invocation latency	7 msecs	7.06 msecs	8.3 msecs
% increase	-	0.85%	18.57%

Table 8.1 Estimated Invocation Latency on Secure Channels

In these examples we have assumed that the secure channel has already been established. It is interesting to estimate the average channel establishment cost but this depends on the number of membership certificates that have to be verified. We give an example that represents a typical case where five membership certificates are required.

We assume that a secure channel has to be established between two objects residing on two hosts that have already exchange a key, that is their Authentication Agents have executed the authentication protocol discussed in Section 7.3.3. We also assume that these Authentication Agents have already established a number of shared keys that can be used for secure channels between objects on their hosts and that the subject AA has collected the necessary membership certificates. In this case a secure channel can be set up with two invocations between the address spaces of the two host managers. In the

[1]Running on Solaris 2.4. The client machine was an IPX while the server machine was a Sparc 20.

first invocation, the target ACA is given the PCL associated with the subject object and the reply contains the CCPL (see Figure 6.11). The target AA is given the necessary membership certificates in the second invocation (see Figure 7.7). These have to be verified by the target AA. We expect that the average size of a membership certificate is 300 bytes (two OIDs + certificate ID + expiration time + Digest). The size of the OID in the current implementation is not constant but the average OID size is about 60 bytes. Thus, the verification of a certificate takes about 0.5msec. Each certificate also has to be translated by the AS. Each translation requires the decryption of two PKCs, an encryption and a decryption of the certificate by the relay as well as a remote invocation. The expected size of a PKC is 250 bytes (OID + CID + expiration time + Digest). Thus a certificate translation takes about 9 msecs (7 msecs for invocation, 1 msec for decryption of two PKCs and 1 msec for decryption and decryption of the certificate). If *n* certificates can be translated in a single invocation where *n* is reasonable small ($n < 10$), the total cost is about ($8 + n$) msecs if we assume that the invocation latency is not seriously affected by the size of the arguments and the reply data. Thus, if we assume that *five* membership certificates have to be verified. The estimated cost for the establishment of the channel is:

- 14 msec for two invocations
- 13 msecs for the translation of the certificates
- 2.5 msecs for the verification by the target AA.

This gives a total of about 30 msecs. This time may be less if the target AA has already verified some certificates as a result of another channel establishment. It is also possible that the authentication servers use special hardware to perform cryptographic operations which makes the translation of the certificates faster. It is important however that only a small fraction (verification and invocations) of this time reflects the required processing effort on the target site.

The cost of the channel establishment is large compared with the estimated invocation latency on a secure channel. In some cases the channel is used for a large number of invocations and thus the establishment cost is amortised over the invocations. In many cases, however, a channel is used for a small number of invocations and it is important that the delay of the first invocation is reasonable. In these cases, the secure channel establishment cost turns out to be high. On the other hand, this cost is small compared with the cost of binding a client to a server (hundreds of msecs).

Delegation also introduces delays. The grantor has to encrypt a DT twice, the grantee has to decrypt the DT and the end-point has to verify the DT and the membership certificates proving the required membership of the grantor and grantees. If we assume that the size of a DT is about 300bytes, the encryption or decryption of a DT takes about 0.5msec. The delay for translation of a DT is 9 msecs.

The grantor has to encrypt a DT with its secret key and the key of the channel on which the delegation takes place. This requires 1 msec. In addition the grantee has to decrypt the DT. Thus, the delegation on a channel takes about 1.5 msecs plus the time required to collect the required certificates. The verification of a delegation takes 9 msecs for the translation of the DT plus 0.5 msec for the decryption of the delegation DT by the end-point. The end-point also has to verify a number of certificates which takes about (8 + 1.5*n*) msecs for *n* certificates.

In these examples we have not included the time required to collect the membership certificates. We assumed that these have been collected and cached by the subject, grantor and grantee ACAs. In order to obtain a membership certificate, a remote invocation is required. This costs about 7 msecs, but a single invocation may be used to obtain multiple certificates from the same domain server.

	Symmetric	**Asymmetric**	**Asymmetric/ Symmetric**
Generate Certificate (Domain Server)	0.5 msecs	300 msecs	600
Translate 5 Certificates	13 msecs	-	-
Verify 5 Certificate (target)	2.5 msecs	25 msecs	10
2 Invocations between Agents	14 msecs	14 msecs	1
Total Channel Establishment Time	29.5 msecs	39 msecs	1.32 (32% increase)
Processing Time on Target Host (crypto operations)	2.5 msecs	25 msecs	10

Table 8.2 Numerical Estimates using Symmetric and Asymmetric Cryptosystems

In order to compare the estimated performance of our system with a hypothetical system based on asymmetric cryptography, we assume that a small public exponent is used so that the verification is faster than the signing. This scheme is usually adopted in practical systems on the assumptions that a certificate is generated once while it has to be verified by multiple verifiers, and that the processing power of the server that signs a certificate is higher than the power of the average verifier. According to a table in (Lampson et al. 1992) that compares the speeds of RSA and DES, if the public exponent equals 3 and a 500 bit modulus is used, signing is about 800 times slower than DES encryption/decryption while verification is about 12 times slower than DES. We assume that we can use a asymmetric cryptosystem capable of verifying 64Kbytes/sec and signing 1Kbyte/sec.

On this assumption, the verification of a membership certificate takes approximately 300 bytes * 1/64KBytes/sec = 5 msecs and there is no need to use relays to translate certificates. In the channel establishment example where five certificates have to be verified, the cost for establishing the channel using asymmetric cryptography is 14 msec for two invocations plus 25 msecs for the verification of five certificates. This gives a total of 39 msecs which indicates a 30% increase on the channel establishment cost compared with symmetric cryptography (30 msecs). In addition, the cryptographic operations performed on the target host require 25 msecs while in our system the cryptographic operations require only 0.5 msecs * 5 = 2.5 msecs (see also Table 8.2). Thus, the use of asymmetric cryptography would impose a significant processing overhead on the target hosts that establish channels with multiple subjects.

	Symmetric	**Asymmetric**	**Asymmetric/ Symmetric**
Create DT (grantor)	0.5 msec	300 msecs	600
Encrypt DT (with the channel key)	0.5 msec	0.5 msec	1
Processing Time on Grantor host	1 msec	300.5 msecs	300
Decrypt DT (grantee)	0.5 msec	0.5 msec	1
Translate 5 Certificates	13 msecs	-	-
Verify 5 Certificates (end-point)	2.5 msecs	25 msecs	10
Translate DT	9 msecs	-	-
Verify DT (grantor)	0.5 msec	5 msecs	10
Translation + Verification Time	25 msecs	30 msecs	1.2 (20% increase)
Processing Time on End-point host	3 msecs	30 msecs	10

Table 8.3 Delegation-related Estimates using Symmetric and Asymmetric Cryptosystems

Also, the signing of a membership certificate requires 300bytes * 1/1Kbyte/sec = 300 msecs. Thus, a domain server can sign approximately 3.3 certificates per second while in our system a domain server can create 2,000 certificates per second (see also Table 8.2).

On the assumption that a large number of domains and dynamic objects are created, the use of asymmetric cryptography introduces a great processing overhead on the domain servers which can seriously affect the performance of the system.

Table 8.3 gives some numerical estimates for the delegation process using symmetric and asymmetric cryptography. If we assume that the end-point has to verify five membership certificates to ensure that the grantee is permitted to act on behalf of a grantor, we expect a 20% increase on the total verification time. In addition, the processing time required on the end-point host increases by a factor of 10 while the processing time on the grantor host increases by a factor of 300. This implies that, when asymmetric cryptography is used, delegation turns out to be very expensive, especially for the grantor host.

8.2.3 Pull vs. Push

The system uses the push method in order to provide the necessary membership certificates to the target machine. In addition, the push method is used to distribute the activated access control policies to the relevant hosts. This section discusses how the push method reduces the channel establishment delay and compares it with the pull method.

The push method enables the AA of the subject to cache membership certificates that can be forwarded to multiple target AAs. Because it is likely that a certificate can be used to establish channels with multiple target objects, this caching is expected to reduce the communication cost required to collect the necessary certificates and therefore reduce the channel establishment cost.

In addition, the target AA can cache “translated” certificates and use them in order to establish channels with multiple subjects that have common parent domains.

The use of the pull method would enable the target to cache membership certificates but we expect this to be less efficient because the possibility of using a certificate cached by the target for establishing channels with multiple subjects is smaller than the possibility of using a certificate cached by a subject to establish channels with multiple targets. This is because, in the general case, different subjects have different parent domain hierarchies.

Another advantage of the push method is that it enables subjects to forward only the certificates that are required to access a given target. Thus a target can only see the membership certificates that the subject has decided to forward.

On the other hand, the pull method enables the verifier (target) to get the certificates directly from the domain service and thus eliminate the possibility of using certificates that certify a revoked membership. This, however, requires that the verifier does not use cached certificates which increases the average channel establishment delay.

The activated access control policies are also pushed to the relevant components. So, an ACA is aware of the policies applying to the channel that is being established. In this way at least an invocation is avoided during the channel establishment process which would introduce a delay of the order of 7msecs. This delay will be higher if policy secrecy is required. This is because the policies have to be transmitted in ciphertext and the policy server has to decide whether the ACA that requires a policy is permitted to get it.

On the other hand, if the target ACA requests a policy when it has to use it in order to establish a channel, it eliminates the possibility of using a policy that has been retracted. In the proposed system, this is avoided by informing the ACAs holding a copy of a policy that it has been retracted (see Section 6.2.5).

8.3 Further Work

Several directions for further work flow directly from the analysis in section 8.1 but there are also aspects that have not been discussed at all in this thesis.

Such an aspect is the implementation of meta-policies controlling the specification of access control policies by the security administrators (see Section 3.2.6). These meta-policies have to be enforced by the policy servers. This is also related with the issue of delegation of authority (Moffett 1990).

We have not discussed the issue of auditing. This is an important security aspect as it enables security administrators to monitor the behaviour of individuals and take the necessary measures should they ascertain the misuse of access rights. It is possible that auditing can be undertaken by security agents employed on each host.

We have shown in Chapter 6 that the application layer can control the delegation of access rights. This provides a high degree of flexibility, but it is worth examining whether it is possible to make the delegation process transparent to the application level for some applications.

In particular, it may be beneficial to control delegation using obligation policies (Marriott and Sloman 1995; Marriott and Sloman 1996) which specify what access rights have to be delegated for certain tasks. Such obligation policies can be interpreted by the Access Control Agents (ACAs) of the grantors. This would allow the development of application objects that do not consider delegation details and do not have to make decisions with respect to the access rights they have to delegate. Note that delegation obligation policies can be defined by the managers being responsible for the grantors while extended access control polices are defined by the administrators being responsible for the target objects. In addition, the use of obligation policies has the advantage that they can be changed if there is a need to modify the behaviour of an application object in terms of delegation, which eliminates the need to modify the application code. Furthermore, it is worth examining if delegation obligation policies can be used in conjunction with roles.

Finally, another aspect is the specification and enforcement of access control policies by the authority which is responsible for the subjects. Our system enforces access control policies that control access to target objects and are enforced by security components that are trusted by the authority being responsible for the target. It is also possible that the behaviour of the subject has to be controlled with respect to the interests of the authority responsible for it.

Consider the following scenario: two companies *A* and *B* have established limited mutual trust in the framework of joint project *PrAB*. Each company however, has to specify policy in order to allow limited access to sensitive information. Note that information can flow either in the form of a reply to an invocation or in the form of an invocation request (for instance the *write* operation on a file). The first flow is controlled by the access control policy discussed in this thesis, for example, objects in *A* can access files in domain *B_PrAB* (maintained by *B*). This policy, however, does not control information flow according to the interests of *A* in that an object inside *A* can append a file in *PrAB* with information that should not be given to company *B*. An abstract policy to control this type of information flow could be: “objects in *A* cannot reveal pricing information to company *B*”. Such a policy is specified by company *A* and should be enforced by components that are trusted by authorities in company *A*, not in company *B*.

There are many aspects of this scheme that have to be worked out. For instance, what components should enforce this type of policy. It is possible that in some cases these policies can be enforced by gateways controlling outgoing invocations. It is also possible that one can use obligation policies to control the behaviour of the subjects. Another aspect is the resolving of possible conflicts among policies specified by different authorities. For example the above-mentioned companies may set different requirements with regard to the security protection of the exchanged messages.

8.4 Final Remarks

We have described an architecture that integrates policy dissemination, access control and authentication mechanisms. Although the system has been designed to cope with the enforcement of domain-based access control policies, it deals with issues that appear in more general distributed access control systems, such as enforcement of access control policy that is specified in terms of groups of principals, delegation of access rights and policy dissemination in large distributed systems.

Appendix

Evaluation of Policy Scopes

This Appendix describes the first phase of the policy dissemination mechanism outlined in Chapter 6. For each policy, the policy server propagates Policy scope Evaluation Tokens (PETs) which enable the ACAs to determine whether the policy applies to objects on their hosts. This mechanism deals with domain structures that change while a policy is active.

A.1 Propagation Control Flags (PCFs)

We use Domain Scope Expressions (DSEs) that have the following syntax:

```
DSExp ::= ANY |
          *object |
          *N object |
          @object |
          { object } |
          DSExp + DSExp |
          DSExp - DSExp |
          DSExp ^ DSExp |
          (DSExp)
```

A DSE contains a number of subexpressions in which an object operator (*, @, { }) is applied to an object. Each subexpression defines a set of objects. For example, the DSE **A + @B - {C}* contains the subexpressions: **A*, *@B*, *{C}*. The first one defines the set that contains all the direct an indirect members of domain *A*. The second defines the set that contains all the direct members of the domain *B*. The third one defines the set containing the object *C*. The subexpression "*ANY*" in a DSE can be replaced by "**Root_Domain*".

Each subexpression generates a *Propagation Control Flag* (PCF). The subexpression that generates a PCF is referred to as the *generator* of that PCF. A PCF contains five fields:

- *Position*
- *Path*
- *Control*
- *Depth*
- *Alive*

The Control, *Depth* and *Alive* fields depend on the object operator in the generator subexpression. Object operators generate flags as follows:

- *Operator *: Control: 0, Depth: 0, Alive: 1*
- *Operator *N: Control 1, Depth: N, Alive 1*
- *Operator @: Control 2, Depth: 1, Alive: 0*
- *Operator { }: Control 1, Depth : 0, Alive 1*

Each PCF is propagated down the domain structure until it arrives at all the ACAs of the objects in the set defined by its generator.

A.1.1 Controlling the Propagation Depth

Let us assume that the target scope of the access rule *AR1* is defined by the expression **3DomA*. This expression generates one PCF that is given to domain *DomA* which further propagates that flag to its members. This is repeated until the flag reaches the members of *DomA* down to the 3rd level. When the member of a domain is a non-domain object, the flag is propagated to the ACA that is responsible for that object. Otherwise it is propagated to the member (subdomain) itself. Figure A.1 illustrates the propagation of this flag to the ACAs of the objects that are in that target scope of *AR1*. Note that the ACA of *ObjZ* does not receive this flag because its distance from *DomA* is more that 3 levels.
The *Depth* field is used to control the propagation depth of the flag. Initially this field is set to 3 by the policy server. Each time the flag is propagated, the value of *Depth* is decreased by 1. The propagation stops when the *Depth* becomes 0. The value of the *Depth* field is illustrated in Figure A.1 for each propagation step.

If the generator defines an unlimited propagation (for example **DomA*), the policy is propagated down to all the direct and indirect members of the domain. In this case the *Depth* field is not used.

If the generator contains the operator @, for example *@DomA*, the propagation stops at the direct members of the domain. For this reason the *Depth* field is set to 1 by the policy server. Note that the flag generated by @ has its *Alive* field set to 0 to indicate that *DomA* is not in the set defined by its generator. This field is set to 1 when the flag is given to the direct members of *DomA*.
Finally, if the operator in the generator is *{ }*, the PCF is given to the object specified in the generator and no further propagation takes place.

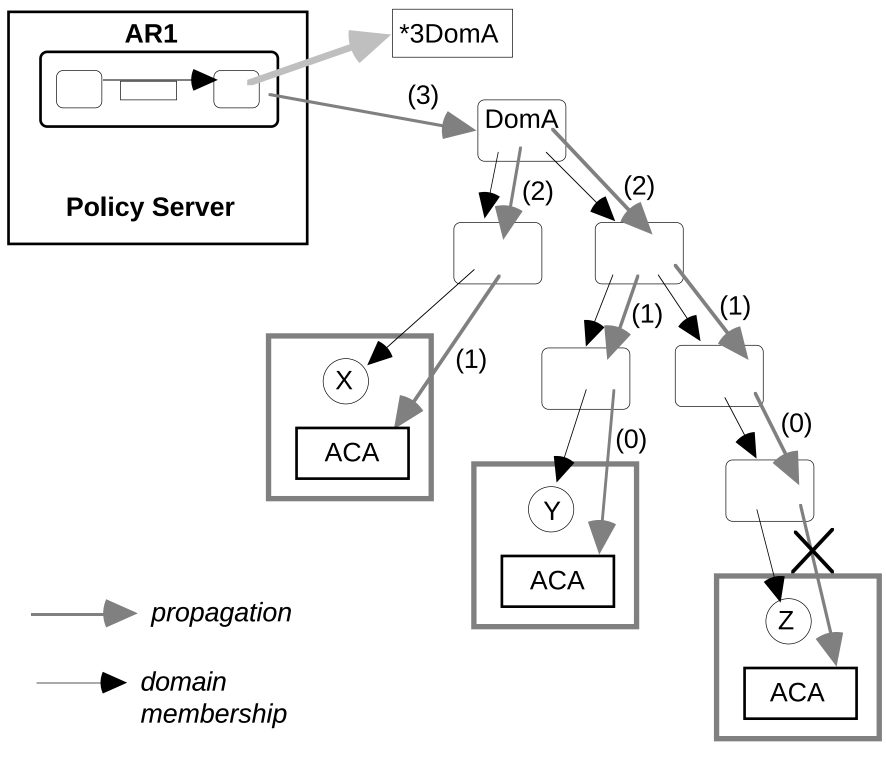


Figure A.1 Propagation of a PCF

A.2 Policy Scope Evaluation Tokens (PETs)

A PCF does not carry any information about the policy that has generated it. So, PCFs are always propagated inside *Policy scope Evaluation Tokens* (PETs). A PET contains the following fields:

- *Policy OID*: The OID of the policy.
- *Scope*: The scope of the policy (subject, grantee or target).

- *DSE*: The DSE defining the scope of the policy.
- *PCFList*: A list of PCFs.
- *Negative*: It is set to 1 when the PETs in the PCFList should be deleted.

The policy server creates a PET for each PCF. Each domain or ACA holds a PET that contains all the PCFs propagated by the same DSE. Furthermore, if a domain has to propagate multiple PCFs generated by the same DSE, it does this by propagating a single PET containing all these PCFs.

An ACA that holds a PET on behalf of an object, can determine whether the object is in the scope of the DSE by checking the PCFList. Consider for example the propagation shown in Figure A.2. The ACA of *C* receives two PCFs which are held in a PET that contains the DSE "**DomA* - **DomB*". The ACA determines that *C* is in both sets defined by the two generators and therefore is not in the scope of the DSE. On the other hand, the ACA of *B* receives only the PCF that corresponds to the first generator and determines that *B* is in the scope of the DSE.

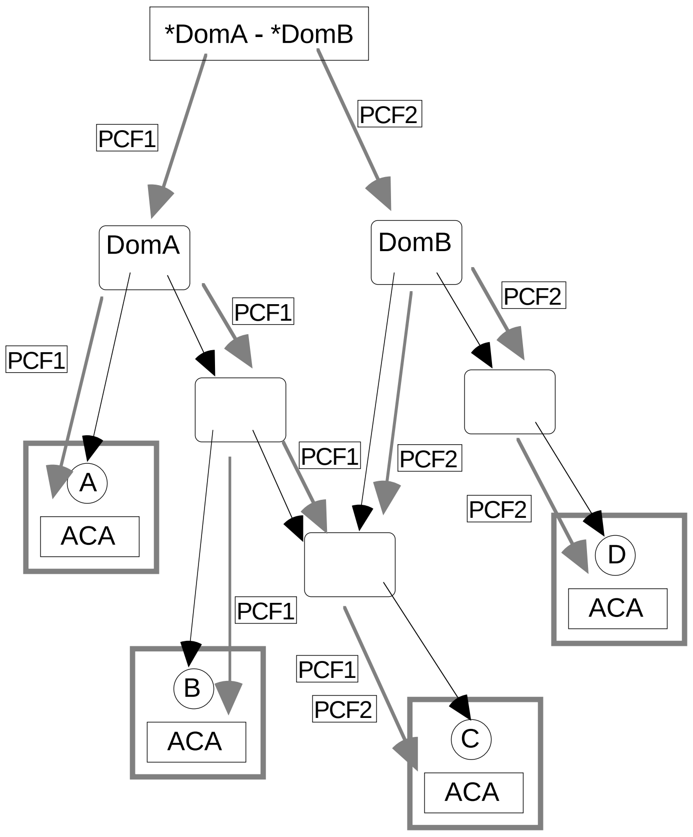


Figure A.2 Propagation of Two PCFs Generated by *DomA - *DomB

A.2.1 Skeleton Parser

We use a simple algorithm to parse a DSE. In fact, the algorithm parses an expression which is called skeleton. A skeleton is derived from the DSE in two steps. In the first step the postfix form of the DSE is created. In the second step, each generator in the postfix is replaced by the character “#”. For example, the postfix of the DSE “**DomA - (*DomB ^ *DomC)*” is “*DomA* DomB* DomC ^ -*” and the skeleton is “*###^-*”.

The *Position* field in a PCF indicates the position of the flag generator in the skeleton and is set by the policy server. The following algorithm determines whether a PET represents a policy that applies to the object on behalf of which the PET has been constructed.

```
int position := 0;
char token;
boolean X, Y;

while (not_empty(pet.skeleton)) {
     token := next_token(pet.skeleton); /* read skeleton */

     if (token == “#”) {
          position = position + 1; /* count the position */
          if (pet.PCFList.alive_PCF_exists(position))
               /* if an alive PCF exists for this position,
               the object is in the scope of the
               generator */
               mystack.push(TRUE)
          else
               /*  if  there  is  no  alive  PCF  for  this
position
               the object is not in the scope of the
               generator */
               mystack.push(FALSE);
     }

     else { /* if token is a set operator */
          Y = stack.pop();
          X = stack.pop();
          switch (token) {
          case :“+”
               if ((X == TRUE) or (Y == TRUE))
                    mystack.push(TRUE);
               else
                    mystack.push(FALSE);
               break;
          case :“-”
               if ((X == TRUE) and (Y == FALSE))
                    mystack.push(TRUE);
               else
                    mystack.push(FALSE);
               break;
          case : “^”
               if ((X == TRUE) and (Y == TRUE))
                    mystack.push(TRUE);
```

```
                else
                     mystack.push(FALSE);
     }
} /* end while; skeleton has been parsed */

X = mystack.pop();
if (X == TRUE)
     /* object in scope */
else
     /* object not in scope */
```

A.2.2 An Example

Figure A.3 illustrates an example of PET propagation. We assume that the target scope of a policy is defined by the DSE "**DomA - (*2DomB ^ @DomC)*" of which the postfix form is "*DomA * DomB 2* DomC @ ^ -*". This expression generates three PETs, namely *PETa*, *PETb* and *PETc* which contain the following PCFs (see also Figure A.4):

- *PETa: PCFa1: [Position: 1, Control: 0, Depth: 0, Alive: 1]*
- *PETb: PCFb1: [Position: 2, Control: 1, Depth: 2, Alive: 1]*
- *PETc: PCFc1: [Position: 3, Control: 2, Depth: 1, Alive: 0]*

In the first propagation wave, these PETs are given to *DomA*, *DomB* and *DomC* which store these PETs and generate new ones to be propagated to their members. *DomA* generates *PETa'* which is given to *DomD* and to the ACA of *X*. This PET contains *PCFa'1* which is identical to *PCFa1*. *DomB* generates *PETb'* which is given to the ACA of *X* and to *DomE*. This PET contains *PCFb'1* of which the *Depth* field is 1. Similarly, *DomC* generates *PETc'* which contains *PCFc'1: [Position: 3, Control: 2, Depth: 0, Alive: 1]*. Note that *DomC* is not in the set defined by *@DomC* and this fact is indicated by the *Alive* field in *PCFc1* which is 0. The members of *DomC*, however, receive *PETc'* that contains *PCFc'1* which is alive and therefore they are in the set defined by *@DomC.*

In the second propagation wave, the ACA of *X* receives three PETs: *PETa', PETb'* and *PETc'*. These are merged into one (since have been generated by the same policy object and for the same scope) which contains the three received PCFs. This PET is held by *X*'s ACA which can determine that *X* is in the target scope of the activated policy. In fact, *X* is in this scope as it is not in the set defined by *(*2DomB ^ @DomC).*

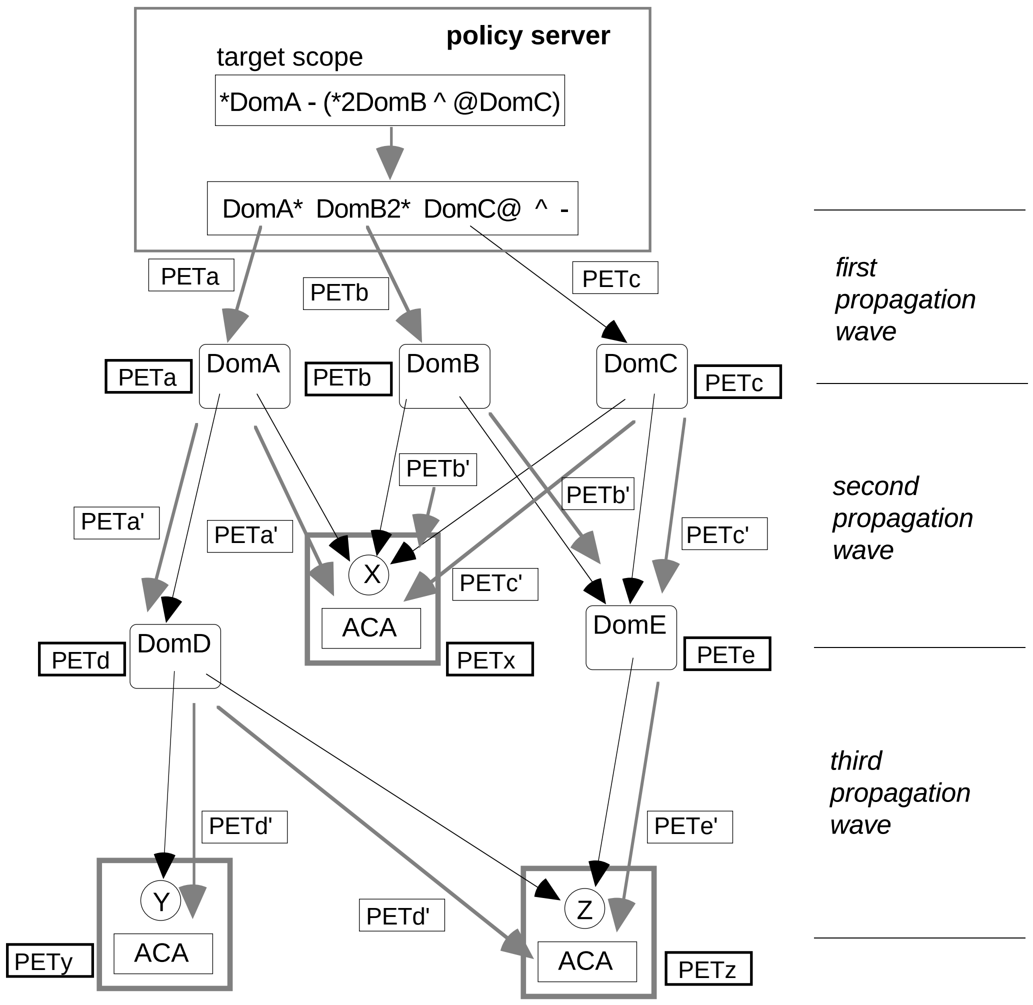


Figure A.3 Propagation of PETs by *DomA - (*2DomB ^ @DomC)

Similarly, *DomE* receives two PETs: *PETb'* and *PETc'*. These are merged into one named *PETe* which is held by *DomE*. This contains the flags *PCFb'1* and *PCFc'1* which indicate that *DomE* is not in the target scope of the policy.

DomD receives and stores *PETd'* which contains *PCFa1* and indicates the this domain is in the target scope of the propagated policy.

In the third propagation wave, *DomD* propagates *PETd'* to the ACA of *Y* which can determine that *Y* is in the target scope of the policy. This PET is also given to the ACA of *Z*. This also receives *PETe'* which contains *PCFe'1*: *[Position: 2, Control: 1, Depth: 0, Alive: 1]*. This PCF is based on *PCFb'1* but the field *Depth* has been reduced by 1. Note that the *PCFc'1* is not propagated by *DomE* because the *Depth* field is 0. Thus, the object *Z* is not in the set defined by *(*2DomB ^ @DomC)*. It is, however, in the scope defined by **DomA - (*DomB ^ @DomC)* as it receives *PCF'1* from *DomD*.

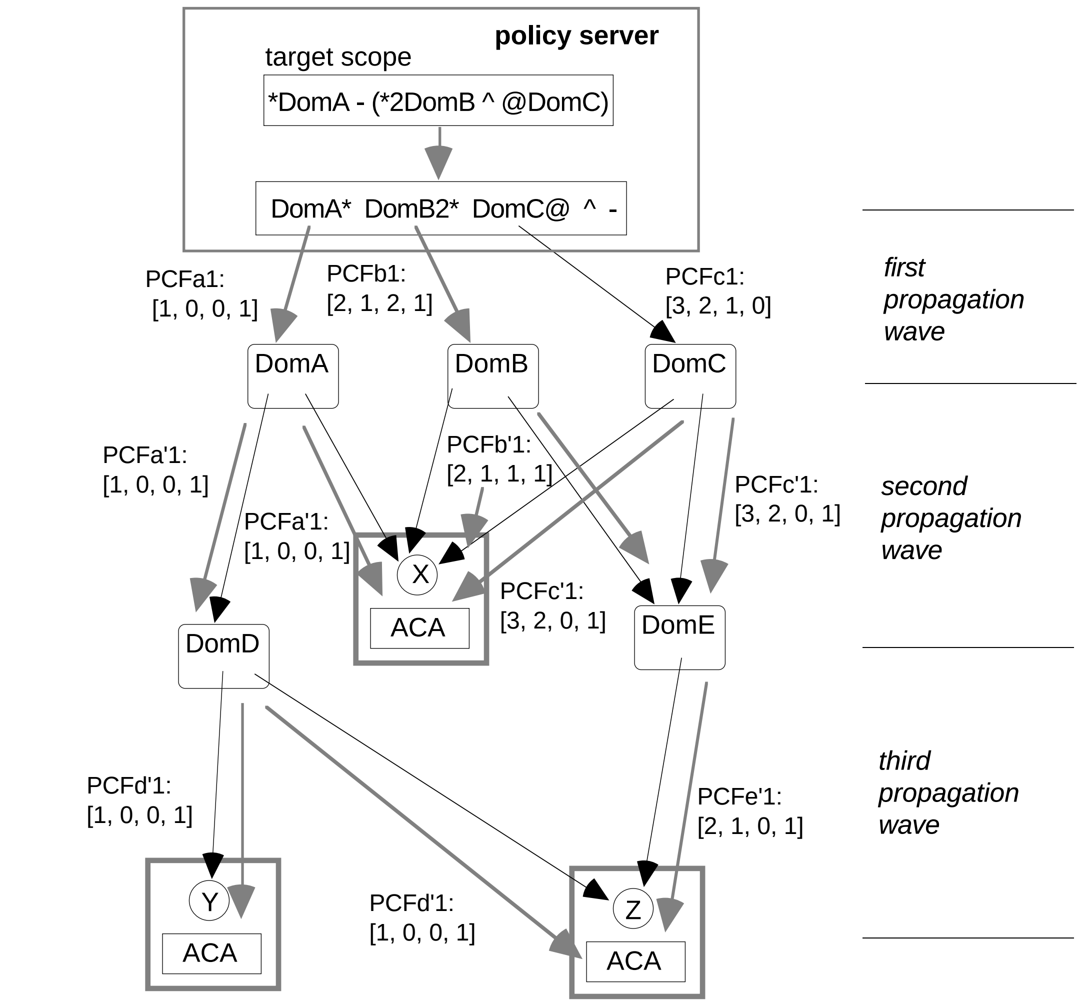


Figure A.4 Propagation of PCFs by *DomA - (*2DomB ^ @DomC)

Table A.1 illustrates the states of the skeleton parser given in Section A.2.1 for the skeleton of the DSE **DomA - (*2DomB ^ @DomC)* and the PET held by the ACA of *Z*. Note that this PET contains two alive PCFs which have been generated by **DomA* (Position 1) and **2DomB* (Position 2). The skeleton of **DomA - (*2DomB ^ @DomC)* is ###^- and the parser terminates with *X* = *True* which implies that *Z* is in the scope of **DomA - (*2DomB ^ @DomC)* .

A.3 Dynamic Domain Structure

This section shows how the propagation mechanism can cope with dynamic domain structures. It is possible that during the life cycle of a policy, new objects are inserted in domains that are affected by the policy. In addition, the set of policies applying to an object depends on its parent hierarchy. An ACA should be able to re-determine the set of policies applying to an object if its parent domain structure changes.

Token	**X**	**Y**	**Position**	**PCF Alive**	**Action**	**Stack**
#	-	-	1	True	Push(True)	True
#	-	-	2	True	Push(True)	True True
#	-	-	3	False	Push(False)	False True True
^	True	False	-	-	Push(False)	False True
-	True	False	-	-	Push(True)	True
	True					

Table A.1 Skeleton Parser States

A.3.1 Insertion of a Non-Domain Object

As discussed in Section A.2, each domain holds PETs representing policies that are active. When a non-domain object becomes member of a domain, that domain propagates the appropriate PETs to the ACA of the new object. The ACA uses this PETs to re-determine the policies applying to the object. For example, if the object *Z* in Figures A.3 and A.4 becomes a member of the domain *DomC*, this domain propagates a PET containing *PCFc’1*. This indicates that *Z* is now a member of the set defined by the generator *@DomC*. The ACA construct a new PET for *Z* which contains *PCFc’1* and uses the skeleton parser which terminates with *X = False* because the PCF in position 3 is alive (see also Table A.1). This implies that the corresponding policy ceases to apply to *Z*.

A.3.2 Removal of a Non-Domain Object

If an object is removed from a domain, its ACA removes the PCFs that have been received by the former parent domain. For this to be achieved, each PCF is associated with the domain that has propagated it. For instance, if *Z* ceases to be a member of

DomC, the ACA removes *PCFc'1* which has been propagated by *DomC* and subsequently runs the skeleton parser for the new PCFList containing only two PCFs (in positions 1 and 2 as in Section A.2.2) which terminates with *X*= *True*.

A.3.3 Insertion of a Subdomain

We now examine the case where a domain becomes a member of another domain. In this case, all the members of the new subdomain may be affected by this insertion. As is A.3.1, the parent domain propagates the appropriate PETs to its new member (subdomain). The subdomain continues the propagation to its members until the appropriate PETs arrive at all its direct and indirect members. For example, let us assume that *DomC* becomes a member of *DomA* in Figure A.5(a). *DomA* holds a PET containing *PCF1*. This PCF is propagated (inside a PET) down to *DomC*. Because *PCF1* has be generated by the subexpression **DomA*, it should be propagated down to all the members of *DomA*. So, *DomC* continues the propagation and eventually the ACAs of *B* and *C* as well as *DomD* receive and store a copy of *PCF1*.

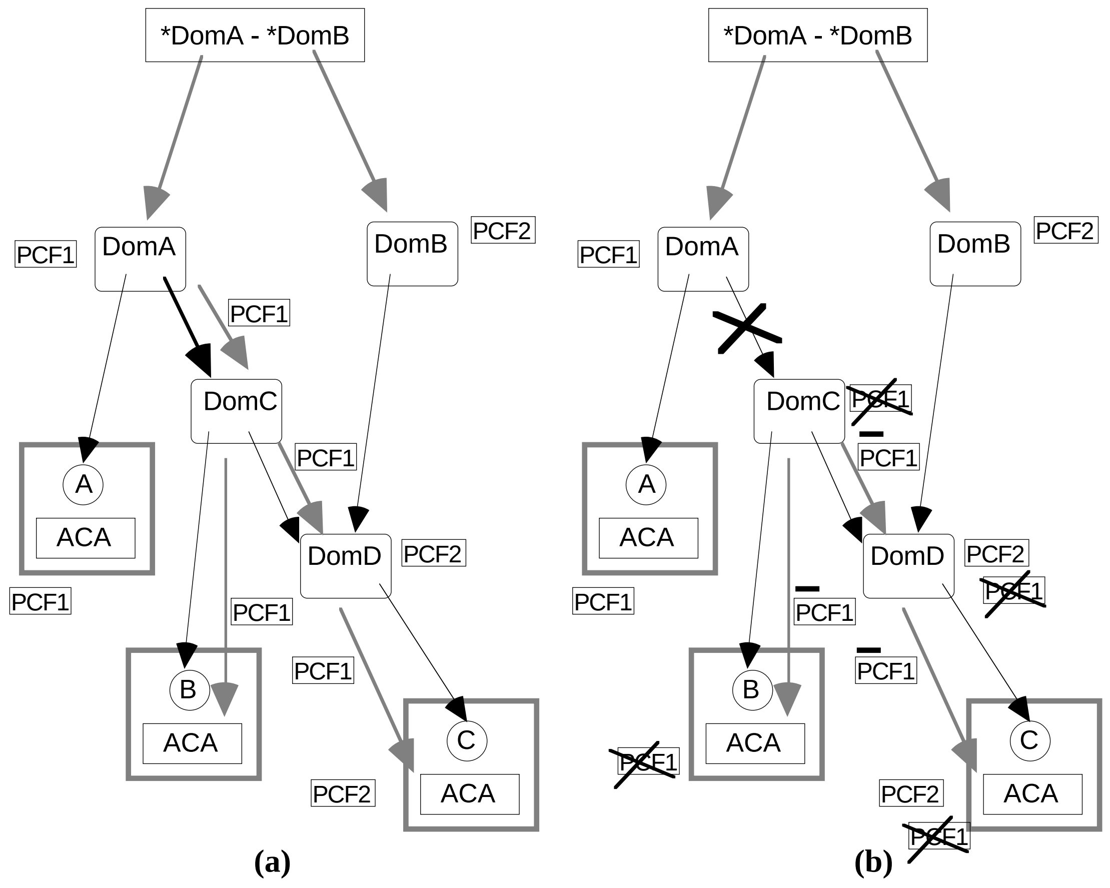


Figure A.5(a) Example for Insertion of a Subdomain
Figure A.5(b) Example for Removal of a Subdomain

A.3.4 Removal of a Subdomain

Finally, we discuss the case in which a subdomain is removed from one of its parent domains. In this case, the removed subdomain deletes the PCFs that have been propagated by its former parent. In addition, this domain constructs and propagates to its members a number of PETs that have their *Negative* field set to 1. A PET that has its Negative field set to 1 is referred to as a *negative* PET. A negative PET indicates that the PCFs it contains should be removed from the PETs maintained by the recipients. The propagation of negative PETs continues down to all the members of the removed subdomain.

For example, if the domain *DomC* in Figure A.5(b) ceases to be a member of *DomA*, it removes *PCF1* from the PET it holds and subsequently propagates to its members negative PETs containing this PCF. The propagation continues until the negative PET arrives at all the indirect members of *DomC*. This results in the removal of *PCF1* from all the PETs maintained by the members of *DomC*. Each time the PCFList of a PET changes, the skeleton parser is employed in order to re-determine the applicability of the corresponding policy.

A.4 Policy Retraction

When a policy is retracted, the policy server propagates negative PETs in order for all the PCFs held by the affected domains and ACAs to be removed. When the PCFList of a PET is empty, the PET itself is deleted.

Figure A.6 illustrates the propagation of PCFs inside negative PETs which results in the deletion of all the PCFs held by domains and ACAs which had previously received PETs because of the activation of a policy of which the target scope is defined by the DSE **DomA* - **DomB* .

A.5 Propagation in the Subject and Grantee Scopes

The same propagation mechanism is employed in order to propagate PCFs for the subject and grantee scopes of a policy. In these cases, however, a PCF records its *propagation path*. This enables the ACAs to determine the domain membership certificates required to prove that an object is in the subject or grantee scope of a policy.

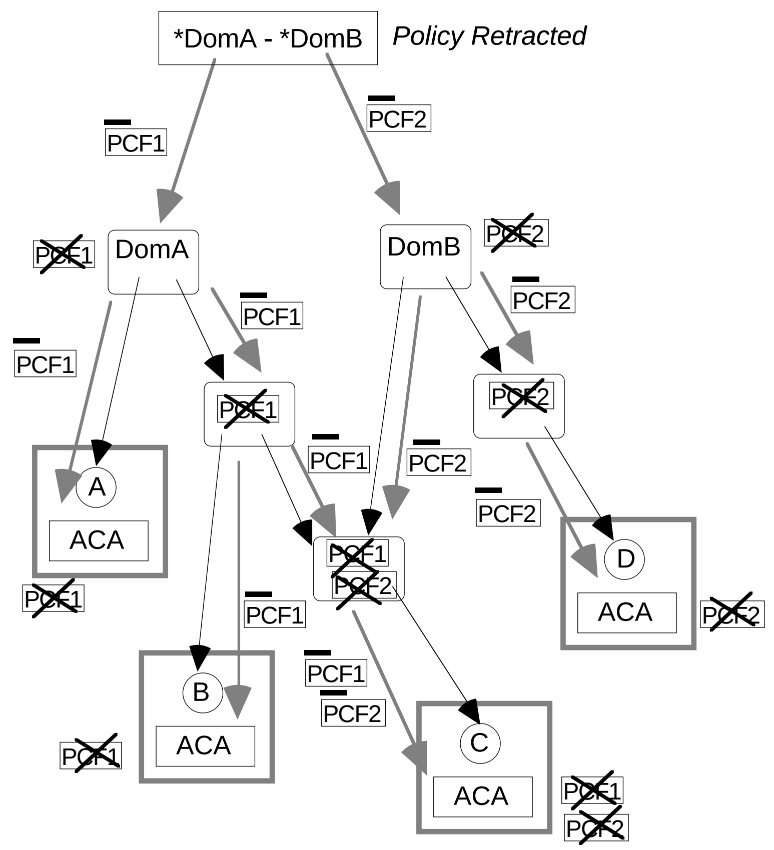


Figure A.6 Propagation of PCFs in Negative PETs for Policy Retraction

Note that an object may be a member of multiple domains that have a common ancestor. So, it is possible that a domain or an ACA receives multiple PCFs generated by the same subexpression. These PCFs, however, record different propagation paths.

Consider, for example the PCF propagation in Figure A.7(a). Domains *D* and *E* receive two copies of the PCF generated by the subject DSE **A*. These PCFs have recorded different propagation paths as it is illustrated in Figure A.7(a). In this case, there are two different certificate set that can be used to prove that *E* is in the scope of **A*:

$\{E < D, D < B, B < A\}$ and $\{E < D, D < C, C < A\}$

Note that if domain *D* ceases to be a member of *C* at some point, the second copy of *PCF1* is deleted from the PETs held by *D* and *E* (Figure A.7(b)). In this case there is only one certificate set that can be used to prove that *E* is in the scope of **A*.

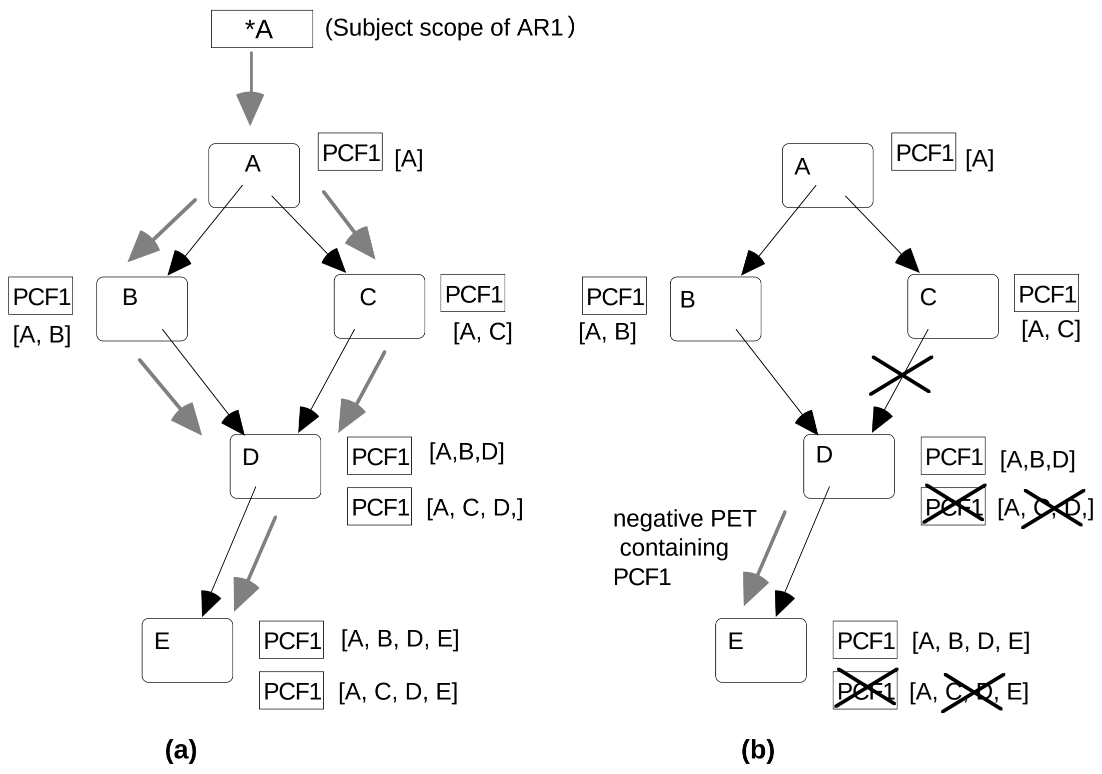


Figure A.7(a) Propagation of PCFs in the Subject Scope of *A
Figure A.7(b) Removal of a PCF

Bibliography

Abadi, M., Burrows, M., Kaufman, C., and Lampson, B. (1990). "Authentication and Delegation with Smart-cards." *DEC SRC 67*, Systems Research Center, Palo Alto, CA.

Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. (1991). "A Calculus for Access Control in Distributed Systems." *DEC SRC 70*, Digital Systems Research Center, Palo Alto CA.

Abadi, M., Burrows, M., Lampson, B., and Plotkin, G. (1993). "A Calculus for Access Control in Distributed Systems." *ACM Transactions on Programming Languages and Systems*, 15(4), 706-734.

Abadi, M., and Needham, R. (1994). "Prudent Engineering Practice for Cryptographic Protocols." *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 16-18, 1994, IEEE Computer Society Press, 122-136.

Abrams, M. D. (1993). "Renewed Understanding of Access Control Policies." *Proceedings of the 16th National Computer Security Conference*, Baltimore, Maryland, Sept. 20-23, 87-96.

Anderson, J. P. (1972). "Computer Security Technology Planning Study." *ESD-TR-73-51, vol 1 AD-758 206, ESD/AFSC Hanscom, AFB Bedford, Mass*.

Becker, K., Raabe, U., Sloman, M., and Twidle, K. (1993). "Domain and Policy Service Specification." *IDSM Deliverable D6, SysMan Deliverable MA2V2 S-SI-07-I-2-R*.

Bell, D. E., and LaPadula, L. J. (1974). "Secure Computer Systems." *ESD-TR-73-278 (Vol I-III)*, Mitre Corporation, Bedford, MA.

Bellovin, S. M., and Merritt, M. (1991). "Limitations of the Kerberos Protocol." *Proceedings of the USENIX*, 253-267.

Biham, E., and Shamir, A. (1990). “Differential Cryptanalysis of DES-like Cryptosystems.” *Proceedings of the Advances in Cryptology-CRYPTO ‘90*, Berlin: Springer-Verlag 1991, 2-21.

Birrell, A. D., Lampson, B. W., Needham, R. M., and Schroeder, M. D. (1986). “A Global Authentication Service without Global Trust.” *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, April 7-9, IEEE Computer Society Press, 223-230.

Burrows, M., Abadi, M., and Needham, R. (1990). “A Logic of Authentication.” *ACM Transactions on Computer Systems*, 8(1), 18-36.

Clark, D. D., and Wilson, D. R. (1987). “A Comparison of Commercial and Military Security Policies.” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 184-194.

Crane, S., and Dulay, N. (1997). “A Configurable Protocol Architecture for CORBA Environments.” *Accepted by The Third International Symposium on Autonomous Decentralised Systems*.

Davis, D., and Swick, R. (1990). “Network Security via Private-Key Certificates.” *ACM SIGOPS Operating Systems Review*, 24(4), 64-67.

Deng, R. H., Bhonsle, S. K., Wang, W., and Lazar, A. A. (1995). “Integrating Security in CORBA Based Object Architectures.” *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 8-10, 1995, IEEE Computer Society Press, 50-61.

Diffie, W., and Hellman, M. (1976). “New Directions in Cryptography.” *IEEE Transactionson Information Theory*, IT-22(6), 644-654.

Diffie, W., and Hellman, M. (1977). “Exhaustive Cryptanalysis of the NBS Data Encryption Standard.” *Computer*, 10(6), 74-84.

Doorn, L. v., Abadi, M., Burrows, M., and Wobber, E. (1996). “Secure Network Objects.” *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 6-8, 1996, IEEE Computer Society Press, 211-221.

Eberle, H. (1992). “A High-Speed DES Implementation for Network Applications.” *Proceedings of the CRYPTO 1992*, 1993, Springer-Verlag, 527-545.

Gasser, M., and McDermott, E. (1990). “An Architecture for Practical Delegation in a Distributed System.” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 20-30.

Gollman, D. (1996). “What do we mean by Entity Authentication?” *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 6-8, 1996, IEEE Computer Society Press, 46-54.

Gong, L. (1992). “A Security Risk of Depending on Synchronized Clocks.” *ACM SIGOPS Operating Systems Review*, 26(1), 49-53.

Gong, L. (1993). “Increasing Availability and Security of an Authentication Service.” *IEEE Journal on Selected Areas in Communications*, 11(5), 657-662.

Gong, L., Needham, R., and Yahalom, R. (1990). “Reasoning about Belief in Cryptographic Protocols.” *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 1990, IEEE Computer Society Press, 234-248.

IONA. (1993). “OrbixTM - A Technical Overview.” *PN: PR-TEC-7-5*, IONA Technologies Ltd. Dublin, Ireland.

ISO. (1989). “Security Framework III: Access Control Framework.” *ISO/SEC JTC1/SC21 N4206 (Draft)*, ISO.

Kaijser, P., Parker, T., and Pinkas, D. (1994). “SESAME: The solution to security for open distributed systems.” *Computer Communications, Butterworth-Heinemann*, 17(7), 501-518.

Kaufman, C., Perlman, R., and Speciner, M. (1995). *Network Security: Private Communication in a Public World*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey.

Kohl, J., and Neuman, C. (1993). “The Kerberos Network Authentication Service (V5).” *RFC1510*.

Krawczyk, H. (1996). “SKEME: A Versatile Secure Key Exchange Mechanism for Internet.” *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, California, IEEE Computer Society Press, 114-127.

Lai, X., and Massey, J. (1992). “On the Design and Security of Block Ciphers.” ETH Series in Information Processing, Hartung-Gorre Verlag.

Lampson, B., Abadi, M., Burrows, M., and Wobber, E. (1992). “Authentication in Distributed Systems: Theory and Practice.” *ACM Transactions on Computer Systems*, 10(4), 265-310.

Lampson, W. B. (1974). “Protection.” *ACM SIGOPS Operating Systems Review*, 8(1), 18-24.

Lampson, W. W. (1971). “Protection.” *Proceedings of the 5th Princeton Symposium Information Sciences and Systems*, 437-443.

Landwehr, C. E. (1981). “Formal Models for Computer Security.” *ACM Computing Surveys*, 13(3), 247-278.

Liebl, A. (1993). “Authentication in Distributed Systems: A Bibliography.” *ACM Operating Systems Review SIGOPS*, 27(4), 31-41.

Linn, J. (1990). “Practical Authentication for Distributed Computing.” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 31-40.

Linn, J. (1993). “Generic Security Service Application Program Interface.” *RFC 1508*, Network Working Group.

Lomas, T. M. A., Gong, L., Saltzer, J. H., and Needham, R. M. (1989). “Reducing Risks from Poorly Chosen Keys.” *SIGOS-ACM Operating Systems Review*, 23(5), 14-18.

Lupu, E. C., and Sloman, M. S. (1996). “Towards a Role Based Framework for Distributed Systems.” *To appear in: Journal of Network and Systems Management*, 4(3).

Lupu, E. C., Yialelis, N., Sloman, M. S., and Marriott, D. A. (1995). “A Policy Based Role Framework for Access Control.” *Proceedings of the 1st ACM Workshop on Role-Based Access Control*, Gaithersburg, Maryland, USA, ACM.

Marriott, D., and Sloman, M. (1995). “Management Policy Service for Distributed Systems.” *Research Report DoC 95/10*, Imperial College.

Marriott, D., and Sloman, M. (1996). “Management Policy Service for Distributed Systems.” *Proceedings of the The Third International Workshop on Services in Distributed and Networked Environments (SDNE’96)*, Macau, IEEE Computer Society Press, 2-9.

Meadows, C. (1992). “Applying Formal Methods to the Analysis of a Key Management Protocol.” *Journal of Computer Security*, 1(1), 5-35.

Miller, S. P., Neuman, B. C., Schiller, J. I., and Saltzer, J. H. (1987). “Kerberos Authentication and Authorization System.” *Technical Report Project Athena Technical Plan*, MIT.

Moffett, J. (1990). “Delegation of Authority Using Domain-Based Access Rules,” PhD Thesis, University of London, Imperial College, Dept of Computing.

Moffett, J., and Sloman, M. (1993). “User and Mechanism Views of Distributed System Management.” *IEE/IOP/BCS Distributed Systems Enginnering*, 1(1), 37-47.

Molva, R., Tsudik, G., Herreweghen, E. V., and Zatti, S. (1992). “KryptoKnight Authentication and Key Distribution System.” *Proceedings of the ESORICS-92*, Toulouse, France, 155-174.

NBS. (1980). “DES Modes of Operation.” *NBS FIPS PUB 81*, National Bureau of Standards, U.S. Department of Commerce.

NCSC. (1985). “Depertment of Defense Trusted Computer System Evaluation Criteria.” *DOD 5200.28-STD*, National Computer Security Center (NCSC).

Needham, R. M., and Schroeder, M. D. (1978). “Using Encryption for Authentication in Large Networks of Computers.” *Communications of the ACM*, 21(12), 993-999.

Needham, R. M., and Schroeder, M. D. (1987). “Authentication Revisited.” *Operating Systems Review*, 21(1), 7.

Neuman, B. C. (1991). “Proxy-Based Authorization and Accounting for Distributed Systems.” *91-02-01*, Department of Computer Science and Engineering, University of Washington, FR-35, Seattle, Washington.

Neuman, B. C. (1993). “Proxy-Based Authorization and Accounting for Distributed Systems.” *Proceedings of the 13th International Conference on Distributed Computing Systems*, 283-291.

Neuman, C., and Ts’o, T. (1994). “Kerberos: An Authentication Service for Computer Networks.” *IEEE Communications Magazine*, 32(9), 33-38.

NIST. (1992a). “The Digital Signature Standard, Proposal and Discussion.” *Comminications of the ACM*, 35(7), 36-54.

NIST. (1992b). “Secure Hash Standard.” *NIST FIPS PUB YY, DRAFT*, NIST.

NIST. (1993a). “Digital Signature Standard.” *NIST FIPS PUB XX*, NIST.

NIST. (1993b). “Secure Hash Stardard.” *NIST FIPS PUB 180*, NIST.

Olson, I. M., and Abrams, M. D. (1990). “Computer Access Control Policy Choices.” *Computers & Security*, 9(8), 699-714.

OMG. (1995). “CORBA Security Draft.” *Draft 0.4*, ORB.

Rivest, R. (1990). “The MD4 Message Digest Algorithm.” *Proceedings of the Advances in Cryptology-CRYPTO ‘90*, Berlin: Springer-Verlag, 1991, 303-311.

Rivest, R. (1992). “The MD5 Message-Digest Algorithm.” *RFC1321*, MIT Laboratory for Computer Science and RSA Data Security, Inc.

Rivest, R. L., Shamir, A., and Adleman, L. (1978). “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.” *Communications of the ACM*, 21(2), 120-126.

Rubin, A. D. (1993). “Formal Methods for the Analysis of Authentication Protocols.” *CITI Technical Report 93-7,* Center for Information Technology Integration, Dept. of Electrical Engineering and Computer Science, University of Michigan.

Schneier, B. (1994). *Applied Cryptography,* John Wiley & Sons, Inc.

Shand, M., and Vuillemin, J. (1993). “Fast Implementations of RSA Cryptography.” *Proceedings of the 11th IEEE Symposium on Computer Arithmetic*, Los Alamitos, CA, 1993, IEEE Computer Society Press, 252-259.

Simmons, G. J. (1994). “Cryptanalysis and Protocol Failures.” *Communications of the ACM*, 37(11), 56-65.

Sloman, M. (1994). “Policy Driven Management for Distributed Systems.” *Journal of Network and Systems Management*, 2(4), 333-361.

Sollins, K. R. (1988). “Cascaded Authentication.” *Proceedings of the IEEE Symposium on Research in Security and Privacy,* IEEE Computer Society Press, 156-163.

Stallings, W. (1995). *Network and Internetwork Security,* Prentice Hall Internatinal Editions, Englewood Cliffs, New Jersey.

Tardo, J. J., and Alagappan, K. (1991). “SPX: Global Authentication Using Public Key Certificates.” *Proceedings of the IEEE Symposium on Research in Security and Privacy,* IEEE Computer Society Press, 232-244.

Tuchman, W. (1979). “Hellman Presents No Shortcut Solutions to DES.” *IEEE Spectrum,* 16(7), 40-41.

Twidle, K. (1993). “Domain Services for Distributed Systems Management,” PhD Thesis, University of London, Imperial College, Dept of Computing.

Twidle, K. P., and Moffett, J. D. (1992). “Domino Reference Manual.” *A3/IC/4*, Imperial College of Science Technology and Medicine, Dept. of Computing.

Varadharajan, V., Allen, P., and Black, S. (1991). “An Analysis of the Proxy Problem in Distributed Systems.” *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 255-275.

Voydock, J., and Kent, S. (1983). “Security Mechanisms in High-Level Network Protocols.” *ACM Computing Surveys*, 15(2), pp135-171.

Wobber, E., Abadi, M., Burrows, M., and Lampson, B. (1993). “Authentication in the Taos Operating System.” *DEC SRC 117,* DEC, Systems Research Center, Palo Alto, California.

Wobber, E., Abadi, M., Burrows, M., and Lampson, B. (1994). “Authentication in the Taos Operating System.” *ACM Transactions on Computer Systems*, 12(1), 3-32.

Woo, T. Y. C., and Lam, S. S. (1992). “Authentication for Distributed Systems.” *IEEE Computer*, 25(1), 39-52.

Woo, T. Y. C., and Lam, S. S. (1994). “A Lesson on Authentication Protocol Design.” *ACM Operating Systems Review, SIGOP*, 28(3), 24-37.

Wray, J. (1994). “Public-Key Login for DCE 1.2.” *RFC 68.0*, OSF.

Yialelis, N., Lupu, E., and Sloman, M. (1996). “Role-Based Security for Distributed Object Systems.” *Proceedings of the IEEE Fifth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprices (WET-ICE’96),* Stanford, CA, June 19-21, IEEE Computer Society Press.

Yialelis, N., and Sloman, M. (1996). “A Security Framework Supporting Domain-Based Access Control in Distributed Systems.” *Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, Dan Diego, CA, IEEE Computer Society Press, 26-39.