A Security Framework Supporting Domain Based Access Control in Distributed Systems

Nicholas Yialelis Morris Sloman

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom
Email: {ny, mss}@doc.ic.ac.uk

Abstract

This paper describes a security framework for object-based distributed systems which is being developed in the CORBA-compliant Orbix™ environment. This framework allows the development of secure distributed applications on existing operating systems that do not support distributed security. The design aims at making the authentication and access control mechanisms transparent to the application level and supporting access control policies specified using the concept of the management domain. This concept has been developed as a means of specifying policies in terms of groups of objects. The description focuses on how the Access Control List paradigm is combined with pseudo capabilities which are used as hints to improve the time-efficiency of the access control decision mechanism. The protocols to support the (cascaded) delegation of access rights to agents acting on behalf of a grantor are explained. A brief description of the authentication mechanism is also given.

1 Introduction

Management of very large, inter-organisational distributed systems cannot be centralised; it must be distributed to reflect the distribution of the system being managed. In addition, these systems may include millions of entities. In order to make the management of these systems practically feasible, the concept of the *Management Domain* has been used within the framework of the Esprit funded SysMan Project. Management domains provide the means of partitioning management responsibility by *grouping* objects in order to specify policies including access control policies[1]. *Access control policies* specify what operations a group of *subject* objects is permitted to perform on a group of *target* objects. As the scope of these policies is specified in terms of management domains, access control decisions are mainly based on the authenticated domain membership of the subject.

A *Domain Service* is provided by a set of domain servers which are distributed for efficiency, availability, management and security reasons. However, this implies that verification of a subject's membership in domains cannot be based on a trusted centralised server. A verifier may receive membership certificates from multiple domain servers which are trusted to different extents (c.f. the notion of the Privilege Attribute Service-PAS in the SESAME architecture [2]). Access control policies are specified in terms of domain membership rather than individual identities so performance of domain membership verification is critical for the access control decision mechanism.

This paper describes a security framework for object-based distributed systems that is being developed using the Orbix™[3] implementation of the Common Object Request Broker (CORBA) distributed programming environment. The aim of the design is two-fold: (i) to provide a security platform for distributed applications that makes the access control and authentication mechanisms transparent to the application level, and (ii) to support the enforcement of access control policies that are specified using management domains. The design of the system supports user authentication, verification of object membership in management domains, secure channel establishment between remote objects, access control and cascaded delegation of access rights. The description given in this paper focuses on the access control and delegation mechanisms. A brief description of the authentication mechanism and the supporting domain and policy services is given.

The framework permits the development of large secure distributed applications on existing operating systems that do not support distributed security. 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 object-based distributed systems and in access control systems that make use of the notion of *group of principals* as a means of specifying access control policies.

The next section introduces the basic concepts of domains, the access control policy and the delegation of access rights. Section 3 sketches the security architecture and discusses the concept of the *secure channel* which is the cornerstone of the architecture. Section 4 explains how a secure channel is established between a subject and a target object and how it is employed to make access

™ Orbix is a registered Trademark of Iona Technologies Ltd, Dublin, Ireland.

control decisions. Section 5 presents the delegation mechanism and how it relates to the access control system. Section 6 summarises the presented architecture and discusses further work. A list of the abbreviations used in this paper is given in Appendix II.

2 Background

This section gives an overview of the basic concepts behind domain-based access control systems and discusses some implementation details with respect to the Orbix environment.

2.1 Objects

An object-based approach has been adopted to building domain-based access control systems. In this approach an *object* encapsulates a *state* and provides an *interface* on which operations on its state can be invoked by other objects. In the Orbix environment objects are maintained within *servers*. In the scope of this paper, we consider a server to be a Unix process (see figure 2.1).

An object has a unique name which is called *Object Identifier* (OID) which is the string representation of the *object reference* in the Orbix environment. It consists of

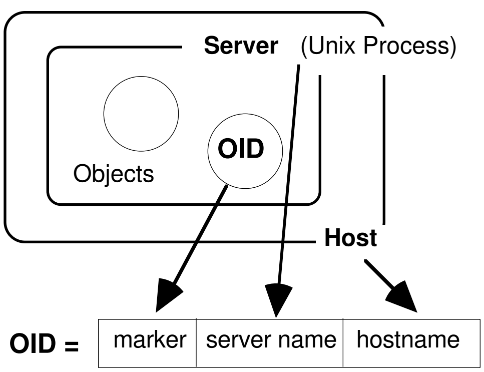


Figure 2.1: OID in the Orbix environment

the object's *marker* which is unique within its server, the *server name* and the *host name* of its server. An object can act as a server (accept invocations) and as a client (make invocations on other object interfaces).

2.2 Domain service

A *domain* is an object which maintains a list of references to objects that have been explicitly grouped together for the purposes of management [1]. 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*. A domain may be a member of another domain and is then said to be a *subdomain* of its parent. Members of a subdomain are *indirect* members of the parent domain. An object can be a direct or indirect member of multiple domains.

For example, in figure 2.2 DomA is the parent domain of DomC, DomC is a direct member of DomA, and DomD and DomF are indirect members of DomA. DomF is a member of DomC and DomE. Finally, ObjX (a non-domain object) is a direct member of DomE and indirect member of DomB.

A *Domain Server* is implemented as a Unix process that maintains domain objects within its address space and supports persistence by storing the information held by domains in a Unix file system. Domain creation is performed by invoking an operation on a Domain Server Object. A domain server is trusted to certify membership of objects in the domains it maintains. Not all domain servers are trusted to the same extent as they may be managed by principals that represent different interests.

2.3 Policy service

2.3.1 Access Control Policies: An *access control policy* is a relationship between a *subject scope* and a *target scope* in terms of the *operations* which subjects are permitted to perform on targets as well as *constraints* on the applicability of the policy (e.g. validity time for the policy) [1, 4]. The operations field specifies operation names along with the type of the objects on which they can be invoked. A graphical representation of an access control policy is given in figure 2.3. The subject and target scopes in a policy are specified using *scope*

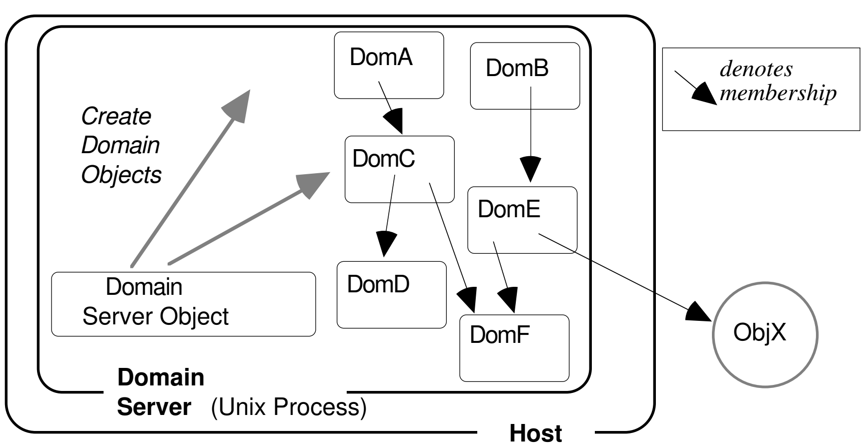


Figure 2.2: A Domain Server maintaining a simple domain structure

expressions. They are applied on objects and return a *set of objects*. Domains provide the means of first-order grouping while scope expressions are used to specify groups in terms of set intersection, difference and union. An example scope expression is *DomA ^ *DomB which defines the scope that contains the objects that are direct or indirect members of DomA *and* DomB (the operator ^ denotes set intersection). A simplified definition of scope expressions is given in Appendix I.

When a person logs into the system, one or more adapter objects are created on the login host. These objects represent the user in the system and their access privileges depend on their domain membership. It is, therefore, important to control to which domains a user is permitted to add new members.

We are also experimenting with the concept of a role as a set of policies applying to a particular manager position [5]. Using this concept, the user would be able to select the access rights an adapter object possesses by specifying the role or roles it is acting in. The use of roles, however, does not affect the access control enforcement mechanism described in this paper as the specification of the access rights is still based on the notion of policy.

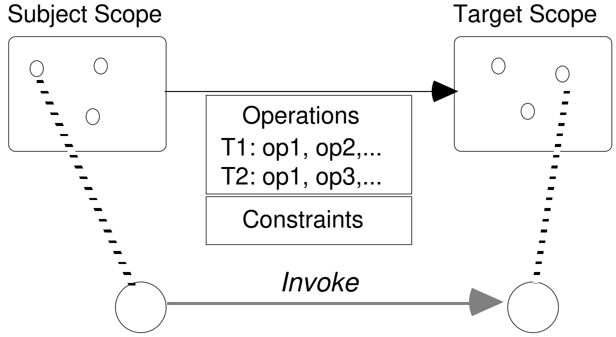


Figure 2.3: Graphical representation of an access control policy

2.3.2 Delegation and Extended Access Control Policies: The notion of the access control policy has been extended to deal with the *delegation of access rights* whereby an object authorises a possibly remote agent to act on its behalf to access a service or resource. Distribution means that the components involved may not be equally trusted. We describe a simple scenario in order to illustrate the issues and define some useful terms. Let us assume that a subject X invokes an operation Op1 on a target Y. This operation triggers the invocation of Op2 on Z (figure 2.4). X possesses the right to invoke Op2 on Z but Y is not trusted to possess this right permanently, so X temporarily delegates the necessary access rights to Y to enable it to invoke Op2 on Z. The access control facilities of Z allow the invocation of Op2 if they ensure that Y has been delegated the necessary access rights. We refer to X as the *grantor,* to Y as the *grantee* and to Z as the *end-point. Cascaded delegation* is also possible in the sense that a grantee may further delegate to a second grantee, the second to a third, etc.

In general, objects maintained within different servers are trusted to different extents. Therefore, it is necessary to control what access rights can be delegated to which objects. In addition, the possession of access rights does not imply the right to delegate these rights. For these reasons, there is a need for a policy scheme that enables the determination of:

i) The *Grantor,* i.e. who can delegate
ii) The *Grantee,* i.e. to whom rights can be delegated
iii) The *Delegatable rights,* i.e. what rights the grantor is permitted to delegate to the grantee.

An *extended access control policy*, (figure 2.5) is used to specify delegation policy. It determines that objects in the *Subject scope* can delegate to objects in the *Grantee scope* the right to perform operations, specified in the *Operations* field, on objects in the *Target scope*.

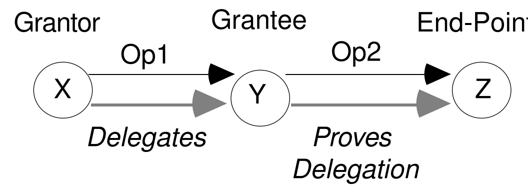


Figure 2.4: A simple delegation scenario

Cascaded delegation is allowed provided that all grantees are in the *Grantee scope* of the policy. This approach is weaker in terms of security than that adopted by the DSSA and described in [6] and [7] which specifies the order of delegation steps in the Access Control List (ACL). Our scheme does not impose constraints on the order of the delegation steps which can be difficult to determine in advance for some applications.

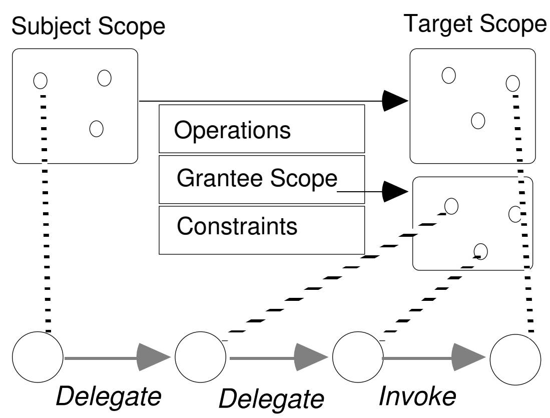


Figure 2.5: Graphical representation of an extended access control policy

2.3.3 Policy Servers: Access Control Policy Objects are maintained within *Policy Servers* (see figure 2.6) which collectively provide the policy service. A policy server object is the factory that creates policy objects. A user that has the necessary access privileges can create, edit, activate and delete policies using a *Policy Editor* [8]. When a policy is activated, *Policy scope Evaluation Tokens*

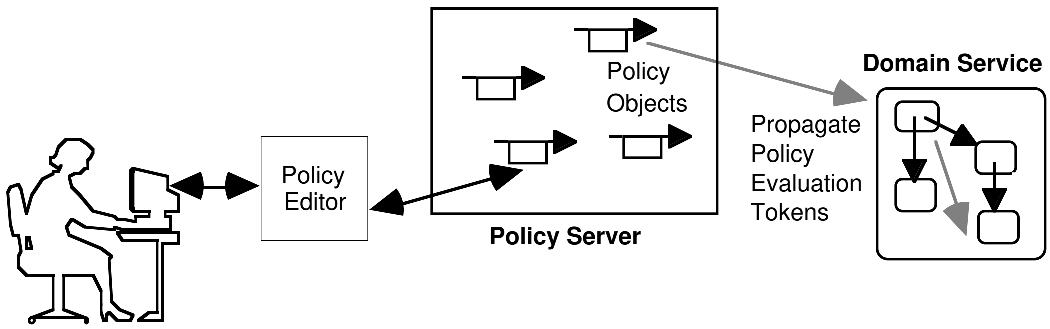


Figure 2.6: Policy Server maintaining Policy Objects

(PETs) are propagated down the domain structure by the domain service. These tokens contain the OID of the policy object and information related to the scope expressions of the policy. This information is used by the domain service and the access control service to determine which objects are in the subject, target and grantee scopes of the policy. This is done dynamically in the sense that the scopes of the activated policies are re-evaluated every time the domain structure changes.

3 Overview of the security architecture

This section gives an overview of the security architecture and briefly discusses the functionality of its main building blocks. In addition, it introduces the concept of the *secure channel* which is the cornerstone of the design. This concept is explained in more detail in section 4.

A Host Manager Server is present on all hosts (see figure 3.1). It supports the *Host Manager Object,* which is aware of all objects maintained within its host, an *Authentication Agent* (AA) object and an *Access Control Agent* (ACA) object which are described below.

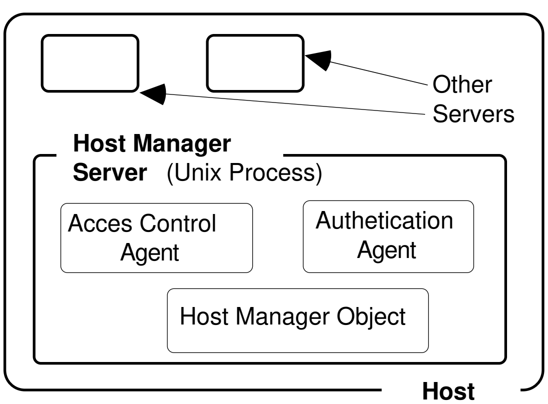


Figure 3.1: Host Manager Server

We take an approach to building the system in which distributed security is provided at the levels of the host manager server and the application server but is transparent to the application objects. This is because the code of the application objects (supported by the application servers) can be developed independently of the security components. A simple application programming interface (API) to the security components is provided. By transferring part of the security functionality into the host manager we minimise the size of the security components that must be replicated among the application servers. In addition, we minimise the number of principals (objects in this context) that have to be registered with the authentication service.

3.1 Security components and secure channels

There is a need to provide secure communication between subject and target objects. The type and degree of security required is dependent upon the application. For many applications, integrity would be sufficient. Other applications may in addition require secrecy. By the term *integrity* we mean that the receiver of a message knows the identities of the possible senders while by the term *secrecy* we mean that the sender of a message knows the possible receivers of that message. We can achieve both properties by encrypting the cleartext along with a suitable *checksum* [7, 9]. Integrity can be achieved by using a *digest* function (see for example [10]). Use of encryption may be limited by time-efficiency requirements or legislation.

The provision of secure communication between two objects requires the execution of an authentication protocol which results in the establishment of a shared secret key (for a formal discussion see [11]). An *Authentication Agent* (AA) on each node executes the authentication protocols on behalf of the application objects on the node. Subjects which request the provision of secure communication can still specify the type and degree of security they require. As shown in figure 3.2, the AA is an object in the address space of the Host Manager Server which is present on all nodes. This server also maintains the *Access Control Agent* (ACA) which holds copies of the access control policies applying to the objects on its node and determines whether a policy exists to permit a remote subject to access a target on its host.

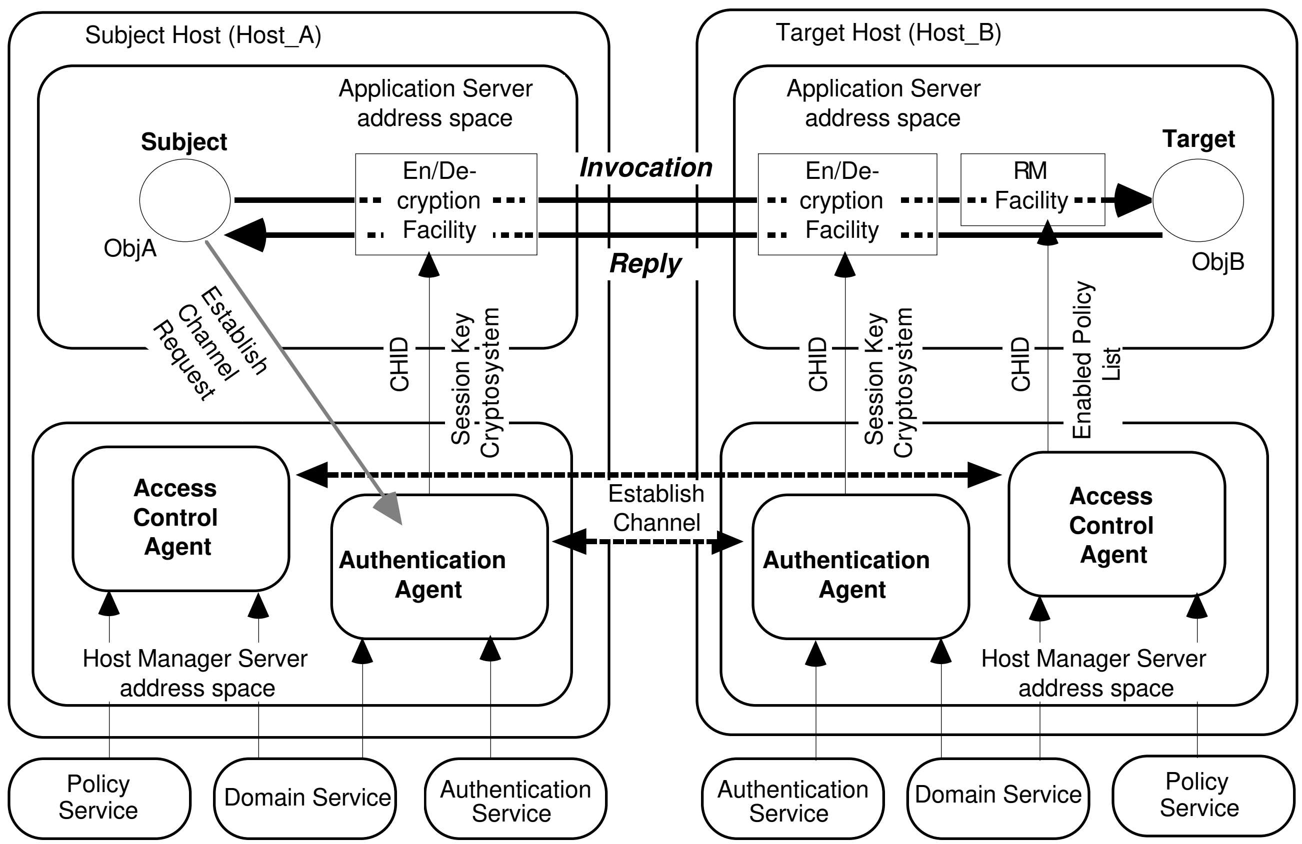


Figure 3.2: Basic Security Architecture

In order to integrate the authentication, cryptographic and access control information related to a subject-target pair, the concept of the *secure channel* has been adopted. This represents a secure communication link between a subject and a target object (i.e. authentication and cryptographic information) as well as access control and domain membership information related to the two objects that use the channel. Each secure channel has a *unique identifier* (CHID) which can be used as a reference to the information related to the channel. Both the AA and ACA are involved in the creation of secure channels on behalf of the application objects on their host. When a secure channel has been established, the target and subject can exchange messages which are encrypted and decrypted by the *En/Decryption facilities* (EDF) in their address space. Furthermore, the access control decision for each invocation is made by the *Reference Monitor* (RM) in the address space of the target. These decisions are based on the access control information related to the channel through which the invocation is made. The RM is provided with this information by the target ACA. The En/Decryption and RM facilities are linked as a library into each application server. The En/Decryption facility uses the transformer feature added for Orbix version 1.3.2. This feature allows access to the buffer carrying a request immediately before it is transmitted and immediately after it has been received.

3.2 A typical scenario

We give a brief description of how a secure channel is used by presenting a simple but typical scenario. Let us suppose that the object ObjA on Host_A intends to invoke operations on ObjB which resides on a remote host Host_B (figure 3.2). ObjB denotes the OID of that object and we assume that it is known to ObjA. We also assume there is an access control policy P1 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. The delegation mechanism is described in detail in section 5.

In figure 3.2, the subject requests the local AA to establish a secure channel between itself and the target ObjB, specifying the type of secure channel it requires. The subject AA and ACA co-operate with those of the target to establish the channel and give the *session key* and the cryptosystem type chosen by the AAs along with the identifier (CHID) of the established channel to the local encryption/decryption facilities. The session key can then be used to exchange messages between the two application objects.

Domain 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 has to be verified to enable operation invocations on the target (ObjB). Furthermore, the subject

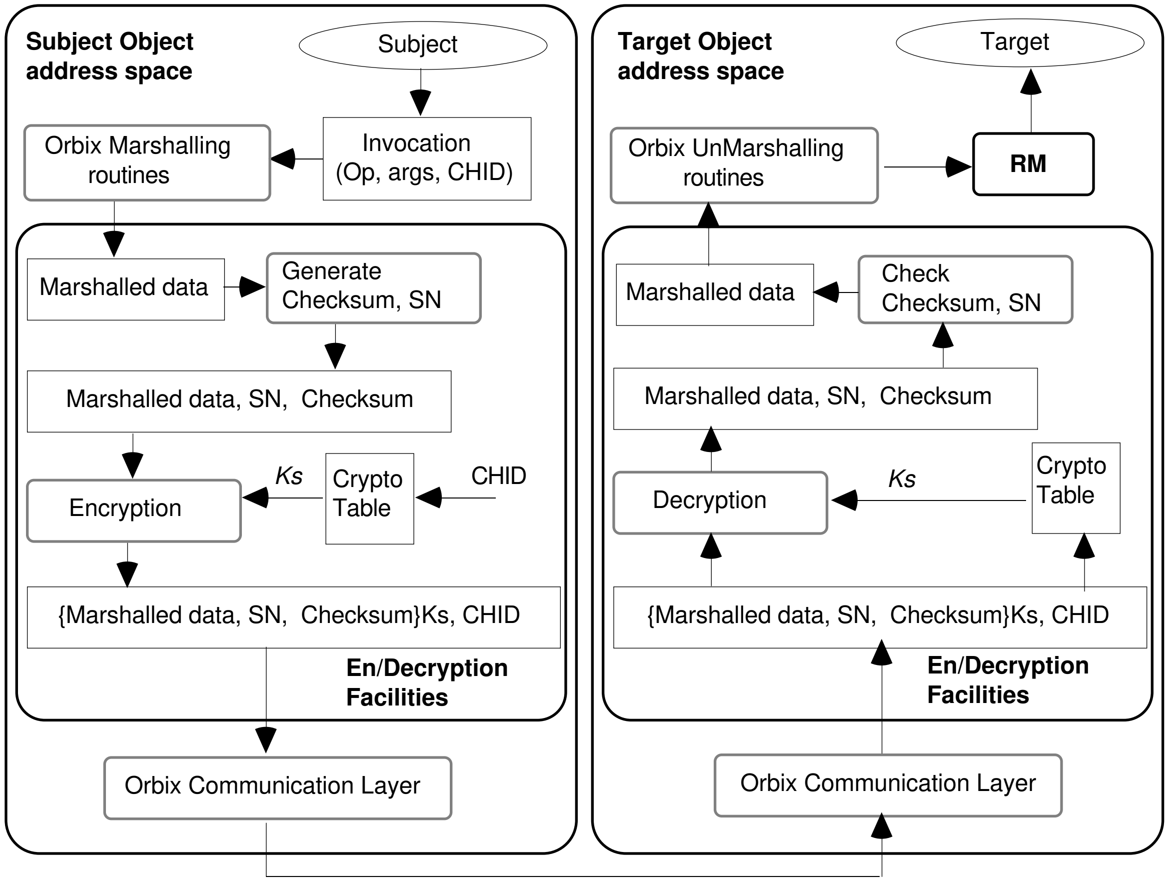


Figure 3.3: En/Decryption Facilities providing secrecy and integrity

may request authentication of domain membership of the target. The verified membership of the subject and target is associated with the established secure channel.

In addition an *Enabled Policy List* (EPL) containing those policies which permit the subject to access the target, is associated with the channel. It is a subset of the policies applying to the target, selected using the authenticated OID and domain membership of the subject. The EPL is given to the RM in the address space of the target which can then grant or reject access requests that are made through the established channel. In our example this list contains the policy P1.

The CHID of an established channel is given to the subject which includes it as a parameter in operation invocations. The target can also retrieve the CHID of the channel through which an invocation has been made. This feature is used when delegation of access rights is taking place (see section 5). The CHID is also used as identifier of the session key and it is transmitted in plain text along with the encrypted messages. This is illustrated in figure 3.3 where we assume that a cryptosystem that provides both secrecy and integrity is employed. Each EDF maintains a CryptoTable with entries to associate CHIDs with session keys and cryptosystems. The subject EDF uses the session key *Ks* and the cryptosystem associated with the CHID specified in the invocation to encrypt the marshalled data of the invocation along with a serial number (SN) and the checksum. The marshalled data contain the operation name, the arguments of the operation and the CHID. Unique serial numbers are used in the context of a channel to identify replayed messages.

The target EDF 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, the checksum and the SN are checked to decide whether the incoming message is fresh and has been sent by the subject associated with the CHID. If the check is successful, the marshalled data is passed to the Orbix unmarshalling routines. The invocation data (containing the CHID) is then passed to the RM. This retrieves the operation name and the CHID. At this stage, the RM believes that the invocation request has been made by the subject associated with the CHID. It therefore checks whether there is a policy in the EPL associated with the CHID that permits the operation. The target and subject EDF swap roles when a reply has to be transferred back to the subject. The following table illustrates how the information associated with a channel is distributed among the main security components. The CHID is used as a reference to this information. Delegation information is also associated with a channel. This is described in section 6.

Subject AA	Subject EDF	Target AA	Target ACA	Target RM	Target EDF
Subject OID, Membership		Subject OID, Membership	Subject OID, Membership		
Target OID, Membership		Target OID, Membership	Target OID, Membership		
			EPL	EPL	
Session Key	Session Key, Cryptosystem	Session Key	Security Attributes		Session Key, Cryptosystem

Table 3.1: Distribution of information related to a channel (see also figure 3.2)

3.3 Authentication service

The Authentication Service (AS) is transparent to the application objects as only the authentication agents interact with it. Authentication agents use the AS to authenticate users as well as remote authentication agents. Other objects cannot be authenticated directly as they are not registered with the authentication service. Only users and authentication agents are registered with the AS which significantly reduces the size and update rate of the security database used by the AS. Application objects and their servers are registered with their local AA and ACA. It is the responsibility of the user (or the object acting on her behalf) to create an application object on a node whose AA and ACA (and system software/hardware) can be sufficiently trusted. An AA believes that the AA and ACA on a remote host are sufficiently trustworthy to act on behalf of the application objects on that node. Note, however, that not all application objects on the same host are equally trusted; objects representing different interests may be maintained on the same host. The secrecy and integrity of communication between servers on the same node (for example between application and host manager server) is provided by the underlying system software and hardware which must be trusted anyway.

An *on-line* authentication service that is based on symmetric cryptography and employs replicated authentication servers with *minimal state* is being developed. It uses *private-key certificates* [12] to disperse the security database so that the authentication servers need to maintain no state apart from their master key. This permits very simple, replicated tamper-proof machines to be used as authentication servers and as *translators* (relays) – for a formal discussion on this see [7]. This function of the AS is of great importance in the proposed security architecture as it allows domain membership verification without establishing shared secret keys between the verifier and the domain servers that can certify the claimed domain membership. Specifically, membership statements encrypted with the secret keys of the AAs of the domain servers, certifying the claimed membership, can be re-encrypted by the AS with the secret key of the verifier. In addition, the *push method* [7] can be employed whereby the claimant can collect the necessary certificates which can later be re-encrypted by the authentication service with the secret key of the verifier.

Revocation of private-key certificates is accomplished by utilising a *revocation server* which periodically publishes a *revocation list* containing the OIDs of the revoked certificates. This list is used by the authentication agents involved in an authentication rather than the authentication service. The current design of this authentication system supports only intra-realm authentication. A more detailed description of this system and the membership verification mechanism is given in [13].

The authentication service could be based on asymmetric cryptography which eliminates the need for network interaction with on-line authentication servers. However, it dramatically increases the encryption/decryption time (according to [7], symmetric encryption is considered to be 1000-5000 times faster than asymmetric cryptography in software). As our system supports multiple domain membership authentication as well as delegation, a relatively large number of certificates/credentials are generated and verified for each secure channel. Thus, use of an asymmetric cryptosystem would significantly increase the processing required for secure channel establishment.

4 Secure channel establishment

This section describes the function of the main security components in relation to secure channel establishment. It also discusses the mechanism that makes access control decisions in the framework of an established secure channel.

4.1 Authentication agent

Authentication agents are registered with the authentication service, so each AA shares a secret key with the AS. Authentication between remote AAs is achieved by using a protocol described in [13]. which is similar to that employed by the Kerberos system [14]. The main difference is that the authentication server in our system is provided with the private-key certificates that contain the secret keys of the AAs involved in the protocol.

Figure 4.1 illustrates the operation of the subject and target AAs when a secure channel between two application objects is being established. If authentication between the two AAs has not already taken place, they execute the authentication protocol which results in the establishment of the shared secret key *Kxy*[1]. At this point AAx

[1]If the subject and target are on the same host, no authentication protocol is executed. The secure channel is established by the local AA and ACA. Notice that in this case integrity and secrecy are provided by the system software/hardware and therefore no cryptosystem is employed.

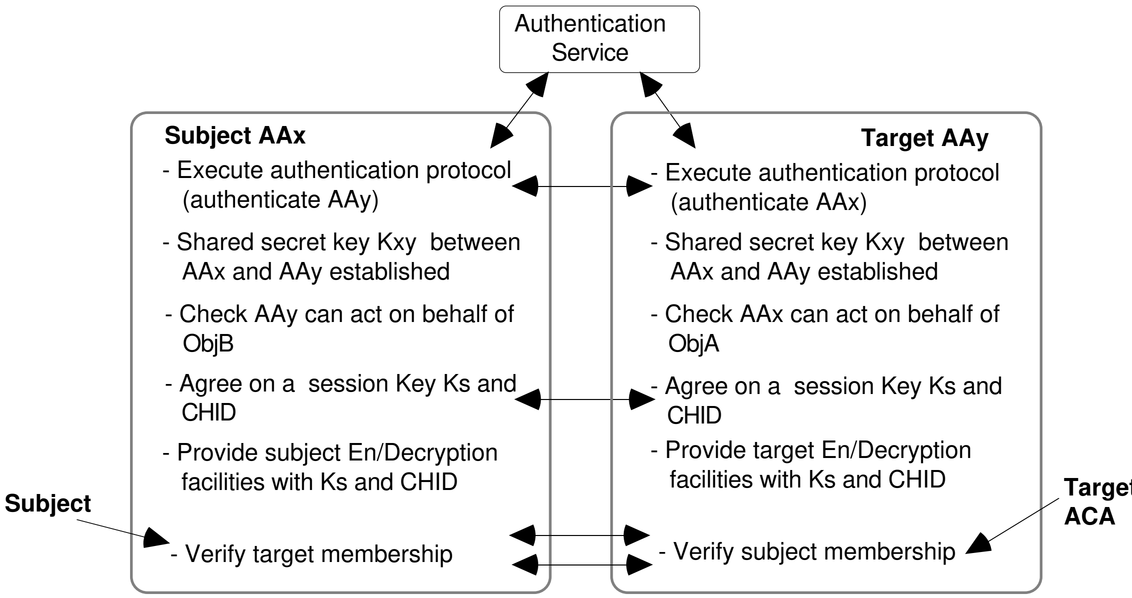


Figure 4.1: Participation of the AAs in channel establishment

establishes that *Kxy speaks for*[2] AAy and, similarly, AAy establishes that *Kxy speaks for* AAx where AAx and AAy denote the OIDs of the agents. Note that these OIDs contain the names of the hosts of the two agents which means that the location of the agents is also authenticated. This is important as an agent can easily verify whether a remote authenticated AA can be trusted to act on behalf of an object of which the OID and consequently the location are known. By definition, a remote AA is trusted to act on behalf of all objects on its node. Therefore, an AA simply checks whether the node name of the authenticated remote agent matches the node name of the object with which a channel has to be established. If mutual trust between the two agents has been established they proceed to choose a session key and a CHID. These are given to the en/decryption facilities of the two application objects. The CHID is generated by the subject AA and is based on its host name and a sequence number.

AAs also perform authentication of object membership in domains using the Domain Service. The target ACA passes to the local AA the membership of the subject that must be authenticated (see section 4.2) and the subject may also request its AA to authenticate a particular membership of the target. A description of the membership authentication mechanism is given in [13]. In addition, authentication agents support delegation of access rights as explained in section 5.

4.2 Access control agent

The Access Control Agent of the target determines the policies that apply to an established channel (i.e. the EPL), when the authentication of the subject and the verification of its domain membership have been accomplished by the local AA. The final access control decision for each invocation is made by the RM in the address space of the target. The ACA determines the policies that apply to local targets by processing the PETs it receives from the policy servers and the parent domains of the targets (see section 2.3.3). For each target it holds an *Access Control Policy List* (ACPL) containing the OIDs of the policies that apply to the target (i.e. policies for which the target scopes include the target). It also holds copies of these policies provided by the policy service. Note that it is necessary to authenticate the identity of the server that provides these copies. Though in a large distributed system the policy service should be provided by multiple servers, in the first implementation phase there is a single policy server whose OID is well known. The integrity of the policy copies is achieved by utilising the AS as a relay to decrypt them using the secret key of the policy server, and re-encrypt them with the secret key of the ACA.

The ACA also maintains a *Pseudo-Capability List* (PCL) for each subject on its host. This list contains the OIDs of the policies applying to it as a subject and so permitting it to invoke operations. When a secure channel has to be established, the PCL associated with the subject is given to the target ACA. The access control decisions made by the target ACA cannot be based on a pseudo-capability. It is used by the target ACA as a hint to find out the *candidate* policies for the EPL. Figure 4.2 illustrates the operation of the target ACA when a channel is being established. The target ACA is given the subject's PCL determined by the subject's ACA. This is used as a hint to produce a candidate EPL which contains the policy OIDs appearing in both the PCL of the subject and the ACPL of the target. The target ACA then requests authentication of specific domain membership of the subject, to ensure it is in the subject scopes of the candidate EPL. If the target AA manages to verify the

[2] *speak for* is used as in [7], that is, A *speaks for* B if the fact that A says something means that we can believe that B says the same thing.

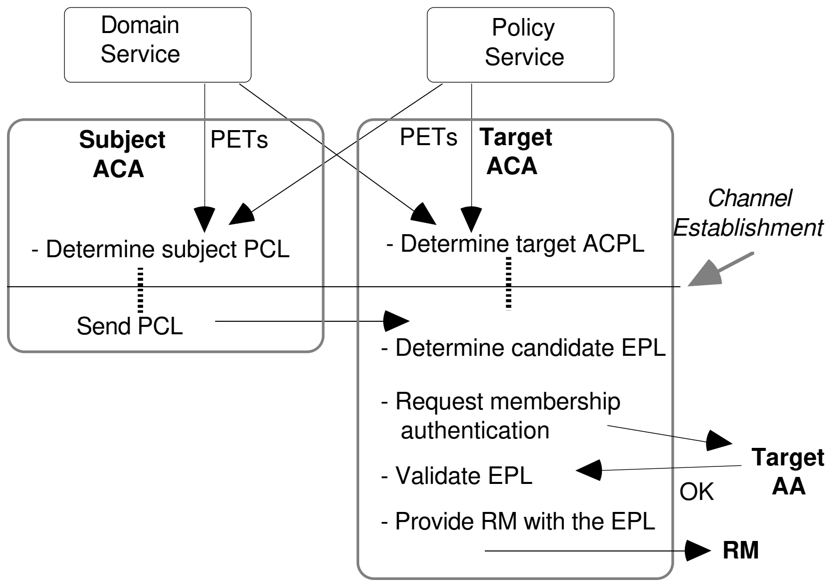


Figure 4.2: Participation of the ACAs in channel establishment

requested membership, the EPL is considered to be valid and it given to the RM of the target.

The PCL specifies indirectly the exact domain membership of the subject that should be checked and therefore makes the determination of the EPL more time-efficient as it considerably reduces the number of policies that the ACA has to check. If the target ACA is not given the PCL, it has to check all the policies in the ACPL of the target to find out which of them apply to the subject. This check, in general, is time consuming since the subject may be direct or indirect member of many domains. Note that if the PCL contains the OID of a policy that does not apply to the subject, verification of domain membership with respect to the subject scope of that policy fails; therefore that policy is not included in the EPL given to the RM.

We currently use the *constraints* field of a policy to determine applicability of policy with respect to dates or time of day, but it could be used to specify the degree of security that should be provided by the channel through which invocations are made. The ACA also checks that the established channel provides sufficient security for the policies in the EPL. For this reason, the ACA is aware of the security attributes of the cryptosystem employed in an established channel (see table 3.1).

5 Delegation of access rights

There is a need for a trustworthy mechanism that enables the security components of the end-point (see section 2.3.2) to verify that a delegation 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 case does not exhibit special interest as the local AA can easily handle the delegation.

Ideally, the grantor should be able to delegate the minimal access rights required by the grantee in order to act on behalf of the grantor. In practice, however, the grantor can rarely determine in advance the exact access rights required by the grantee. In [7] and [15] the rights to be delegated are specified in terms of the roles the grantor adopts. In our system, rights can be specified in terms of domain membership of the grantor. The current implementation does not support the facility which specifies what access privileges are delegated, so all the access privileges of the grantor are delegated to the grantee. Note, however, that extended access control policies may limit the rights a grantor can delegate (see section 2.3.2).

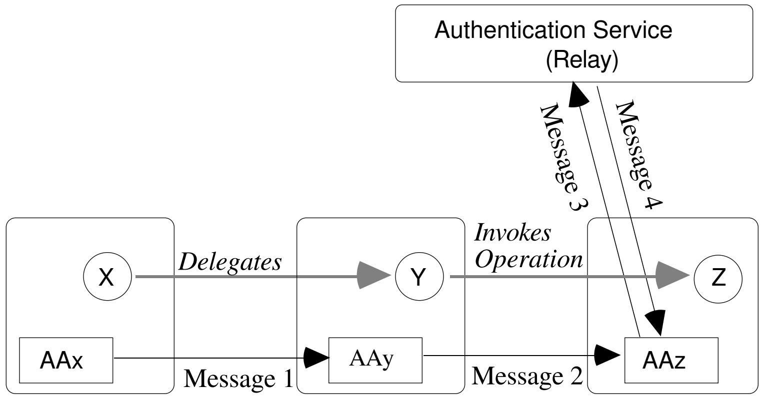
5.1 Delegation protocols

This section describes the protocol employed to verify delegation. This protocol involves the AS which is used as a relay. Figure 5.1 illustrates the protocol sequence for one step delegation.

The protocol uses an encrypted *delegation token (DT)* for each delegation step which is represented as:

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

DID is the identifier of the delegation token, created by the AA of the grantor using its host name plus a time stamp.
PDT is the DID of the Preceding Delegation Token for cascaded delegation.
Tde is the expiration time of the delegation – assuming *loosely synchronised* clocks.
DAR specifies the delegated rights (for future use) – omitted in the presented protocols.
Grantor is the OID of the grantor.
Grantee is the OID of the grantee.



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: $\{DT\}Kz$
(*receive translated DT)*

$\boldsymbol{DT} = \langle DID, \text{-}, Tde, X, Y\rangle$, $\boldsymbol{PKCx} = \{Te_1, AAx \xleftrightarrow{Kx} AS\}_{Km}$

$\boldsymbol{PKCz} = \{Te_2, AAz \xleftrightarrow{Kz} AS\}_{Km}$

Figure 5.1: One step delegation protocol

The grantor AA encrypts the delegation token with its secret key and sends it to the grantee AA. The grantee AA forwards it to the end-point AA as a proof that a delegation has taken place. Since the delegation token is encrypted with the grantor AA's secret key, its integrity cannot be verified by the end-point AA. The DT has to be re-encrypted with the end-point AA's secret key by the AS which can retrieve the secret keys of both the AAs from the private-key certificates that are sent in message 3 along with the encrypted delegation token. In figure 5.1 the private-key certificate *PKCi* contains the secret key *Ki* of AAi and is represented as:

$\{Te, AAi \xleftrightarrow{Ki} AS\}_{Km}$ where

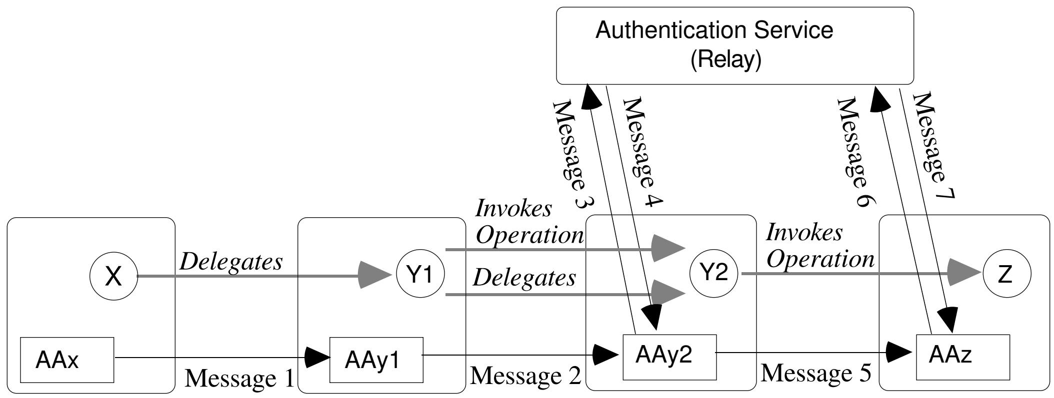
Km is the *master key* of the AS. (We assume that a cryptosystem that provides both secrecy and privacy is used.)
Te is the expiration time of the certificate

The authentication server that translates the DT checks that the AA that encrypted the certificate can speak for the grantor whose OID is mentioned in the DT. It does so by comparing the host names in the OIDs of the AA and the grantor. If the server does not perform this check, anyone can make the AS translate a DT that represents a delegation which has never taken place. The "translated" DT is given back to the end-point AA which can now establish that X has delegated to Y. In fact, whenever an authentication server translates an encrypted statement of the form $\{A \text{ says } s\}_{Kx}$, it checks that the agent that encrypted the statement actually *speaks for A*.

In addition, the grantor AA sends to the grantee AA the same DT encrypted with the shared secret key that has been established between the grantor and the grantee AAs (*Kxy* in figure 5.1, message 1). The grantee AA can decrypt this token and verify its integrity. In this way the grantor is aware of the delegation details. The integrity of the DT is of importance when cascaded delegation takes place. Note that the DT encrypted with the secret key of the grantor cannot be read directly by the grantee; it has to be translated by a relay.

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 secret key shared with the next grantee. Figure 5.2 illustrates the protocol sequence in a two-step cascaded delegation.

The PDT field 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. For instance, consider the scenario illustrated in figure 5.3. X1 delegates its rights to Y1 (token *DT1*) which subsequently delegates these 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.



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: *$\{DT1\}_{Kz}$, $\{DT2\}_{Kz}$*
(receive translated DT1 and DT2)

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

$$\boldsymbol{PKCx} = \{Te_1, AAx \xleftrightarrow{Kx} AS\}_{Km}, \boldsymbol{PKCy1} = \{Te_2, AAy1 \xleftrightarrow{Ky1} AS\}_{Km}$$

$$\boldsymbol{PKCy2} = \{Te3, AAy2 \xleftrightarrow{Kz} AS\}_{Km}, \boldsymbol{PKCz} = \{Te4, AAz \xleftrightarrow{Kz} AS\}_{Km}$$

Figure 5.2: Two step delegation protocol

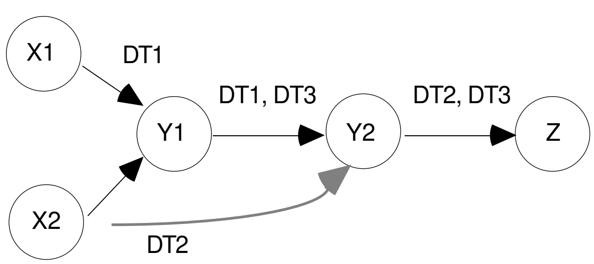


Figure 5.3: Misuse of Delegation Token if the PDT field is omitted

Another possible attack is illustrated in figure 5.4. 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 *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 the grantor AA is trusted not to repeat the DIDs it generates.

The expiration time of the delegation tokens provides a simple revocation mechanism. However, we are working on a more drastic revocation mechanism, based on a trusted server that periodically publishes a revocation list containing the identifiers of the revoked DTs. The integrity of this list is checked using the AS as a relay.

5.2 Channels and delegation

A delegation takes place in the framework of an established channel between the grantor and the grantee. Thus, a CHID can be used as a reference to a delegation. In addition, when a subject intends to invoke operations as a grantee, it requests the establishment of a channel declaring that it is acting on behalf of certain grantors by specifying the CHID related to the last delegation step. This information is used by the ACAs and AAs to perform the necessary authentication and verification of the claimed delegation, and subsequently determine the EPL that enables the grantee to invoke operations on behalf of the grantor.

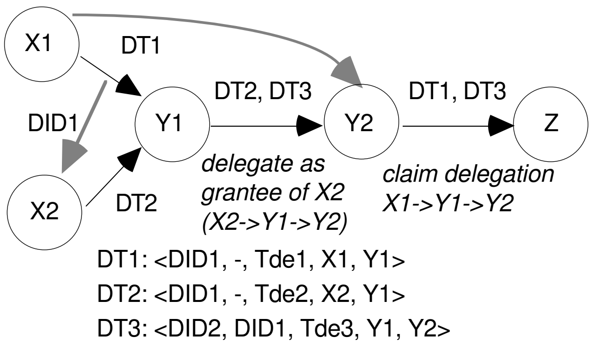


Figure 5.4: Attack on the delegation protocol using repeated DIDs

In figure 5.5, X delegates to the target of the channel Ch1. The expiration time of the delegation is specified by X. This delegation is associated with Ch1. When Y requests the establishment of a channel between itself and Z, it declares that it intends to act as grantee with respect to the delegation associated with Ch1. The subject of the resulting channel Ch2 is the object Y *acting on behalf of* X. This information is used by the AA and ACA on the node of Z in order to determine the EPL that enables Y to invoke operations as grantee of X. The next section discusses the mechanism employed to determine the EPL when delegation is involved.

5.3 Delegation and access control

The ACA is aware of the policies applying to the subject on its host. The list of the policies allowing a subject to delegate rights is referred to as the **Delegation Pseudo-Capability List** (DPCL). In other words, the DPCL associated with a subject contains the e*xtended* access control policies that appear in the PCL of the subject. Furthermore, each object is associated with a **Delegation Policy List** (DPL). This list contains the OIDs of the policies that enable the object to act as grantee. The intersection of the DPCL of the grantor and the DPL of the grantee gives the DPCL of the grantee with respect to the channel associated with the delegation. The DPCL of the grantee is forwarded to the ACA of the end-point in order to determine the candidate EPL for a channel of which the subject acts as grantee.

We refer to the example illustrated in figure 5.6 in order to show how the delegation mechanism can be integrated with the access control mechanism. We assume that X on host A invokes an operation Op1 on the object Y on host B, which triggers the invocation of a second operation Op2 on the object Z on host C. We also assume that there is an access control policy P1 that enables X to invoke Op1 on Y and an extended policy P2 that gives to X the right to invoke Op2 on Z as well as to delegate that right to Y. For brevity, we assume that no other policies apply to these objects. X, Y, Z, P1 and P2 represent OIDs.

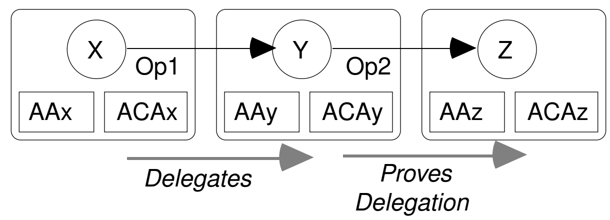


Figure 5.6: A simple delegation scenario

Initially, a secure channel Ch1 between X and Y is established. The PCL given to ACAy is (P1, P2) (see figure 5.7). The ACPL of Y is (P1), therefore the EPL given to the RM of Y contains the policy P1 which permits X to invoke Op1 through Ch1. When X delegates to Y, ACAy is given the DPCL of X which is (P2). As the DPL of Y contains the same policy OID, the DPCL of Y with respect to Ch1 also contains P2. At this stage, AAy

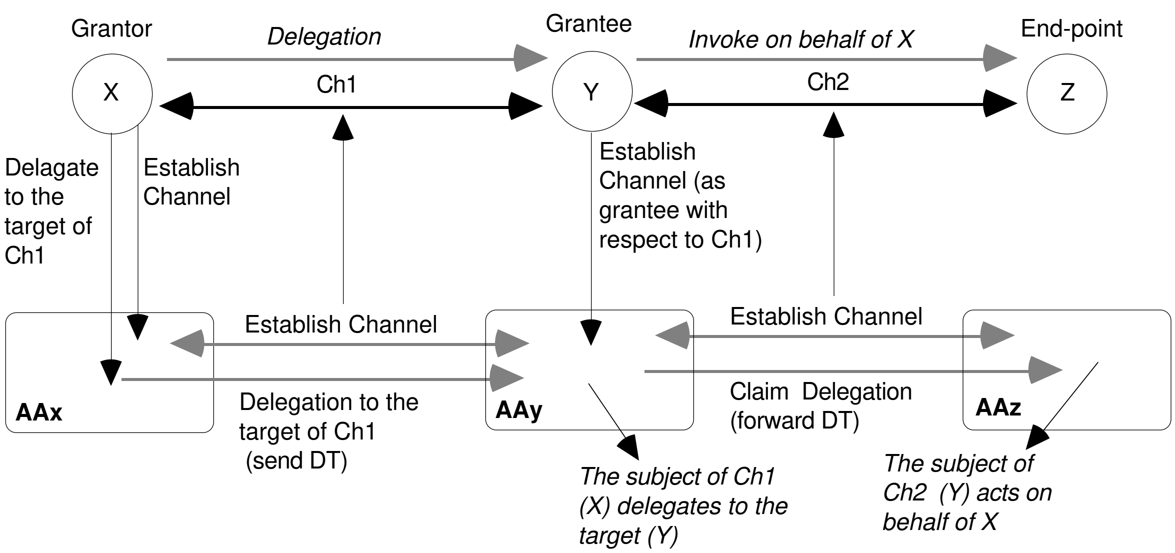


Figure 5.5: One step delegation associated with channel Ch1

requests the grantor AA (AAx) to provide the membership certificates that ensure that X is in the subject scope of P2. These certificates are encrypted with the secret keys of the AA of the domain servers that certify the requested membership. AAy will later forward these certificates to the end-point AA.

Following the invocation of Op1, Y, *as a grantee of* X, requests the establishment of a channel between itself and Z. ACAz is given the DPCL of Y with respect to Ch1. This list contains P2. Since the ACPL of Z also contains the same OID, ACAz determines a candidate EPL that contains P2. In order to validate this EPL, AAz requests the authentication of the domain membership of X and Y that ensure that these objects are in the subject and grantee scopes of P2. In addition, it requests the verification of the claimed delegation. AAy forwards the required membership certificates of X (already given by AAx), and sends the membership certificates of Y as well as the delegation token. These certificates and the delegation token are re-encrypted with the secret key of AAx by the AS. As soon as AAz verifies the membership of X and Y, and the claimed delegation, ACAz validates the EPL which is then given to the RM of Z. Y can then invoke Op2 as a grantee of X.

6 Conclusion and future work

This paper has described a security framework for object-based distributed systems aimed at enforcing access control policy specified in terms of domains. Domains provide a means of specifying policies in terms of groups of objects so that it is not necessary to specify policies for individual objects in large-scale systems containing millions of objects.

A high degree of authentication and access control transparency is achieved by employing security agents on a per-node basis. These agents are trusted to act on behalf of the application objects maintained at their node. The concept of the secure channel is used as a means of integrating access control, cryptographic and authentication information. A subject is permitted to invoke operations on a target in the context of secure channels established by their local security agents, provided that there are policies permitting these operations. Pseudo-capabilities are sent by the subject as a hint to locate the relevant access control policy at the target.

The access control mechanism has been extended to deal with the delegation of access rights at both the policy specification level and at the implementation level, although extensions are need to allows the grantor to delegate a limited set of access rights to the grantee. This presupposes that the application object is capable of getting information about its access rights and its domain membership.

An important issue that has not been covered in detail in this paper is the verification of domain membership. This is achieved by utilising an authentication service that is based on symmetric cryptography and employs authentication servers with minimal state. Development of that authentication system is in progress.

The described security framework is currently being implemented in the CORBA-compliant Orbix environment. The domain service and policy service have been implemented using ANSAware as the distributed

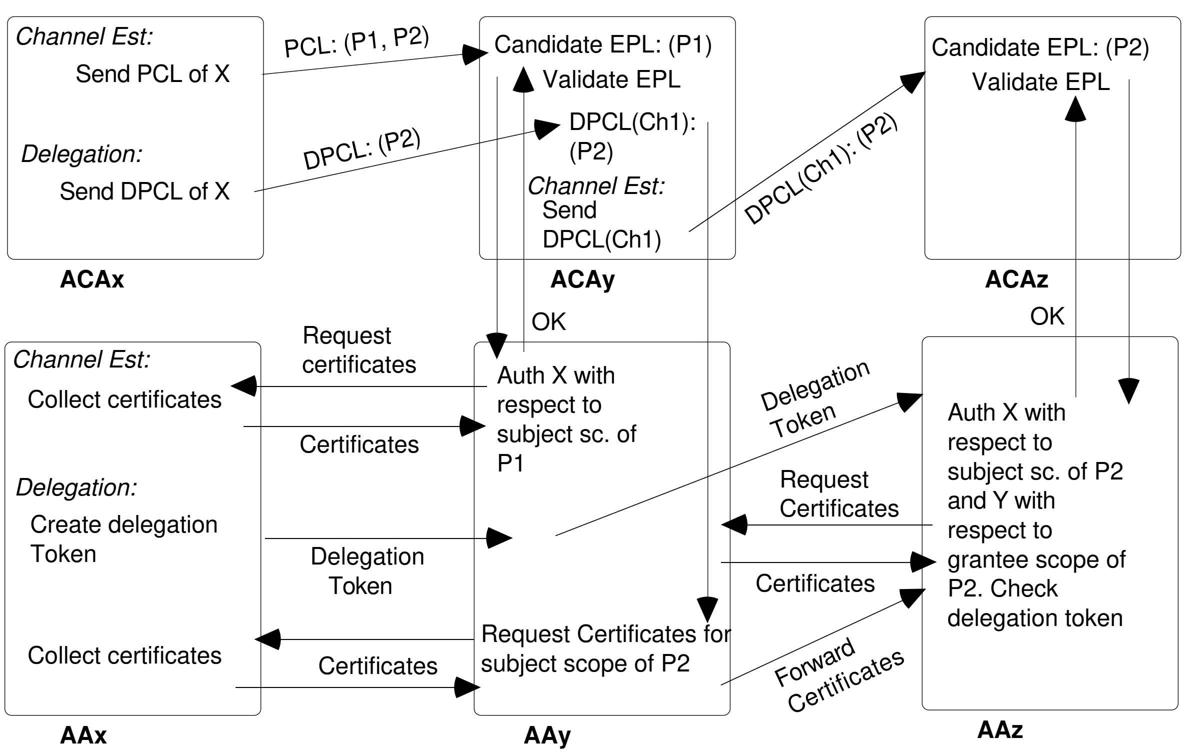


Figure 5.7: Access control mechanism in one step delegation

platform and Tcl/Tk to provide the graphical environments but it is also being ported to the Orbix environment.

Acknowledgements

We gratefully acknowledge financial support from the Swiss Bank Corporation (London) and Esprit SysMan (7026) project. We also acknowledge the contribution of our colleagues, working on these projects, to the concepts discussed in this paper.

References

[1] M. Sloman, "Policy Driven Management for Distributed Systems", *Journal of Network and Systems Management*, Vol. 2(4), pp. 333-361, 1994.

[2] P Kaijser, T. Parker and D. Pinkas, "SESAME: The solution to security for open distributed systems", *Computer Communications,* Vol. 17(7), pp. 501-518, 1994.

[3] IONA, "Orbix™ - A Technical Overview", Technical Report PN: PR-TEC-7-5, IONA Technologies Ltd. Dublin, Ireland, 1993.

[4] K. Becker, M. Sloman and K. Twidle (eds), "Domain and Policy Service Specification", IDSM Deliverable D6, SysMan Deliverable MA2V2 S-SI-07-I-2-R, 1993.

[5] E.C. Lupu and M.S. Sloman, "An approach to Role based management for Distributed Systems", Imperial College Research Report DoC 95/9, 1995, ftp://dse.doc.ic.ac.uk/dse-papers/management/rmds.ps.Z.

[6] M. Abadi, M. Burrows, B. Lampson and G. Plotkin, "A Calculus for Access Control in Distributed Systems", *ACM Transactions on Programming Languages and Systems*, Vol. 15(4), pp. 706-734, 1993.

[7] B. Lampson, M. Abadi, M. Burrows and E. Wobber, "Authentication in Distributed Systems: Theory and Practice", *ACM TOCS* Vol. 10(4), pp. 265-310, 1992.

[8] D. Marriott and M. Sloman, "Management Policy Service for Distributed Systems", Imperial College Research Report DoC 95/10, 1995, ftp://dse.doc.ic.ac.uk/dse-papers/management/maps.ps.Z.

[9] J. Voydock and S. Kent, "Security Mechanisms in high-level network protocols", *ACM Computing Surveys*, Vol. 15(2), pp. pp135-171, 1983.

[10] R. Rivest, "The MD5 Message-Digest Algorithm", RFC RFC1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., 1992.

[11] M. Burrows, M. Abadi and R. Needham, "A Logic of Authentication", *ACM Transactions on Computer Systems*, Vol. 8(1), pp. 18-36, 1990.

[12] D. Davis and R. Swick, "Network Security via Private-Key Certificates", *ACM SIGOPS Operating Systems Review*, Vol. 24(4), pp. 64-67, 1990.

[13] N. Yialelis and M. Sloman, "An Authentication Service Supporting Domain Based Access Control Policies", Imperial College Research Report DoC 95/13, 1995, ftp://dse.doc.ic.ac.uk/dse-papers/management/auth.ps.Z.

[14] S.P. Miller, B.C. Neuman, J.I. Schiller and J.H. Saltzer "Kerberos Authentication and Authorization System", Technical Plan MIT, 1987.

[15] M. Gasser and E. McDermott. "An Architecture for Practical Delegation in a Distributed System", In *Proceedings of the IEEE Symposium on Research in Security and Privacy,* pp. 20-30, 1990.

Appendix I

A simplified definition of scope expressions is given below:

Syntax:

```
SC_EXPR ::= *object |
            @object |
            { object } |
            SC_EXPR + SC_EXPR |
            SC_EXPR - SC_EXPR |
            SC_EXPR ^ SC_EXPR |
            (SC_EXPR)
```

Operators:

+ set union

− set difference

^ set intersection

* when applied on a domain object, a set is returned that contains all *direct and indirect* members of the domain and the domain object itself; otherwise a set is returned that contains the object itself.

@ when applied on a domain object, a set is returned that contains all *direct* members of that domain; otherwise Ø is returned.

{ } returns a set that contains the object on which it is applied.

The interpretation of the expressions is from left to right.

Appendix II

The following abbreviations are used in this paper:

AA	Authentication Agent
ACPL	Access Control Policy List
AS	Authentication Service
CHID	Channel Identifier
DCPL	Delegation Pseudo-Capability List
DPL	Delegation Policy List
DT	Delegation Token
EDF	Encryption/Decryption Facilities
EPL	Enabled Policy List
OID	Object Identifier
PCL	Pseudo-Capability List
PET	Policy scope Evaluation Token
RM	Reference Monitor