3

The CORBA Security Architecture

3.1 Introduction

The main purpose of most security systems is to control access to information, based on a set of security policies. In Chapter 2, you learned how information security can be broken down into a number of different aspects, in particular, confidentiality, integrity, and availability. Accountability and dependability are often additional important security goals.

But these essential security requirements are not the only criteria for designing a useful security architecture. A number of additional considerations also need to be taken into account. Most importantly, the security architecture needs to be designed in such a way that it can be integrated into the system without breaking the functional requirements of the overall application architecture. After all, a security architecture is worthless if it enforces the necessary security policies but, at the same time, renders the system unsuitable for its original purpose. In such systems, users will go to great lengths to circumvent the security enforcement to get the functionality they need to do their jobs. There is a trade-off between making the security architecture as unobtrusive as possible and, at the same time, providing effective security enforcement, and it depends on the particular application where the best trade-off is. For CORBA security, being unobtrusive means that the security architecture has to preserve the design requirements of CORBA’s middleware architecture (as described in Chapter 1), in particular, interoperability, transparency, flexibility, portability, and scalability.
This chapter describes the abstract architectural design of the CORBA security architecture. We will start with a general description of a number of design principles (see Section 3.2) and then present the basic functional components of the security architecture (Section 3.3). Note that the effectiveness of the supported functionality will be discussed in more detail in subsequent chapters. For now, we will simply present an abstract architectural model of CORBA security without getting bogged down into too much technical detail or a discussion of its weaknesses.

### 3.2 Design Goals: CORBA Security Wish List

The CORBA security services specification [1] states that its architecture was designed with a number of goals in mind. These design goals have to be understood as a mission statement (or a wish list) rather than a realistic set of targets, as some of the design criteria are too ambitious and sometimes even conflict with others. For example, there is a fundamental clash between interoperability and flexibility [2], because flexibility involves the customization of functionality, whereas interoperability can only be accomplished through standardized functionality and protocols. Another conflict along these lines is between flexibility and assurance—real assurance can only be certified if the system as a whole is looked at, and when some component is changed, the certification needs to be reconsidered. But flexibility in CORBA security means that components can be changed without affecting any parts on the layers above (this property is also called portability).

The remainder of this section will discuss the main design requirements of the CORBA security architecture. There are also a number of requirements that are less interesting for our conceptual discussion, such as good performance and support for object-orientation. Performance depends heavily on the particular implementation, and it is clear that the CORBA security architecture has to be object-oriented in order to work with CORBA.

#### 3.2.1 Interoperability

The single most important requirement of the CORBA architecture is interoperability across heterogeneous systems. Remember that the OMG was originally founded to establish an architecture that enables interoperability between objects on top of ORBs from different vendors, which in turn run on different operating systems.

Also, as we have already mentioned, the security architecture should aim to integrate with CORBA in a nonobtrusive way, which means that it
has to preserve the main CORBA requirements. In order to preserve CORBA interoperability, the security architecture should therefore also be interoperable, which includes several aspects. First, it should be possible to provide a set of consistent security policies across a heterogeneous system in which different vendors provide different ORB and security products. That way, organizations can implement their distributed system without any vendor restrictions and can choose the most appropriate technology. For example, some ORB products are optimized to provide real-time or fault-tolerance properties, whereas others are optimized for speed or small code size. A possible scenario could use a heavy fault-tolerant ORB on the server side and a lightweight ORB product on a wireless client device. Similarly, there are a wide range of CORBA security products that support security technology for different purposes. In some Intranet environments, a full-fledged security system with auditing and nonrepudiation may be required, whereas a simple Secure Sockets Layer (SSL)-based security service may be sufficient for some less critical browser-based applications.

Second, objects that reside on a secured ORB should still be able to interoperate with objects that do not have any security. Of course, such communications will not be secured, and it depends on the particular security requirements of the application (on the security-enabled end) if this is advisable or not. From an architectural viewpoint, this requirement means that the security protocols have to be layered over the unsecured interoperability protocols, and that the security enforcement has to be integrated into the ORB communications path in such a way that it can be switched on or off, depending on the security policy for each invocation. There are plenty of possible application scenarios in which an object calls numerous other objects with varying levels of security. For example, an electronic shopping cart application calls a catalog object without any security features to allow the client user to browse the products on sale. Once the user has made a selection of goods, the application would then call the payment object to carry out the purchase—and, of course, this call would need to be secured.

A third aspect is interoperability across domains that support different security policies (e.g., different access control attributes). This feature can be useful in large-scale systems in which callers select their targets dynamically and therefore cannot agree a priori on a common set of attributes. The only viable way to achieve this would involve converters that map attributes from one security policy to the other, but even that may sometimes be impossible if the semantics of the attributes differ dramatically. The current version of the CORBA security specification therefore explicitly excludes interfaces and protocols for cross-domain interoperability.
Finally, it may appear to be useful to support interoperability across systems that support different security technology (e.g., different authentication mechanisms). The advantage of cross-mechanism interoperability would be that an appropriate set of security mechanisms could be chosen for each application environment without inhibiting interoperability. For example, the Secure European System for Applications in a Multivendor Environment (SESAME) would be too heavy to support on a wireless device, whereas SSL may be too limited in its functionality to protect corporate Intranet applications. But when the wireless device is used to connect to the Intranet, secure interoperability should still be possible.

Of course, there are various general problems with this scenario and with cross-mechanism interoperability. First, the representation of security policy attributes in CORBA (e.g., for authentication) is not fully mechanism-independent, which means that the policy can only be evaluated if the corresponding mechanism is supported. But even if the attributes were somehow mechanism-independent, there would be a number of semantic problems: An authenticated SSL identity does not always specify the same object as a Kerberos identity, so it is unclear which one is meant by the abstract identity in the policy [3]. In addition, the fact that a particular mechanism is used is implicitly part of the policy. For example, a target object that services callers that support a stronger authentication mechanism may want to reject callers that have been authenticated by weaker mechanisms. Therefore, it may not be desirable to abstract from the underlying security mechanism at all. On the other hand, not abstracting means that CORBA security will not be able to interoperate if the underlying security mechanisms do not match.

Without any abstraction, cross-mechanism interoperability is difficult to achieve, in particular if incompatible cryptography is used. As previously described for interoperability of policy attributes, it is possible to design converters that map invocations from one technology to another. But this way, the converter has to become a trusted third party with access to all cryptographic keys, which breaks end-to-end security, in particular peer authentication. Due to all these problems, the CORBA security specification should not be expected to ever support interoperability between security mechanisms.

### 3.2.2 Transparency and Abstraction

Another big advantage of CORBA is that it insulates application objects from lower-level details and complexities—the middleware layer abstracts technical details, such as locating remote objects and mediating invocations...
to them. The fact that CORBA is largely transparent on the application layer simplifies distributed applications programming, and therefore reduces the overall application development cost. In addition, application programmers do not have to be distributed systems experts to connect their application components.

The security model has to fit seamlessly into the overall middleware architecture to preserve these transparency requirements. As a result, application development should ideally be insulated from security policies and enforcement, just like all the other technical details that happen behind the scenes in CORBA-based systems. Ideally, the security system is simply inserted into the communications path, so that it can enforce the security policy whenever an invocation arrives at the ORB (either from the application above or from the underlying network). The security policies also reside on the middleware layer and specify the required protection (e.g., which invocations are to be passed on or if their occurrence should be logged).

Despite the general requirement that application developers are not required to know about security policies and enforce them for their applications, it should optionally be possible to enforce application-specific security for security-aware applications. As a result, the CORBA security specification distinguishes between security-unaware applications that have been developed without any concern about security, and security-aware applications that have been designed with application-specific security features in mind. Security-unaware applications are protected transparently by various ORB layer security features, whereas security-aware applications can access a range of application layer security features to enforce more application-specific and fine-grained policies. Full transparency is, of course, only a requirement for ORB layer security features.

But the security architecture should not just cater to application programmers. From an end-user perspective, security should also be as transparent as possible. In particular, the architecture should support single sign-on, so that users can log onto their computer once and then reuse the generated credentials for a number of applications until they log out or the credentials expire. This single sign-on feature allows application developers to write applications in which security enforcement is almost entirely transparent to users. Also, application development can be done without concern for security, as basic security policies can be enforced on the middleware layer (i.e., transparent to the applications).

Administrators have to carry the main burden for middleware security, as they have to set sensible security policies for all involved components. To do their job, administrators need to look behind the scenes of the
middleware layer to obtain the necessary in-depth knowledge about the functionality and location of different application components. The administrative model should be simple to understand and manage, and should provide a single consolidated view of the system. It should allow flexible and fine-grained security policies and, at the same time, support clustering of users and objects (e.g., in roles and domains) for scalability. Most issues of the specified CORBA security model become evident for the administrator—a number of security attributes in policies contain mechanism-specific information that breaks the single view of the system, as well as the abstraction from the underlying technology; moreover, some parts of the model are based on inappropriate security information such as the object interface type. We will discuss these problems in more detail and propose alternative solutions where possible (in Chapter 6).

3.2.3 Flexibility, Portability, and Integration

CORBA was designed to be flexible enough to work in a variety of different environments, ranging from relatively static Intranets with integrated legacy applications on the one end, to wireless devices on the dynamically changing Internet on the other. Such diverse CORBA objects can communicate if there is an IDL language mapping for all participating platforms and if all nodes communicate through the same CORBA protocol (e.g., the GIOP). Most other aspects of a CORBA system, such as the underlying transport layer, can ideally be flexibly replaced without affecting interoperability.

In line with this, the security architecture should also be flexible enough to fit a wide range of applications and environments with differing security requirements. The model should allow a variety of different security policies and security features, depending on the level of protection required for information on the system. To accomplish that, the model’s security attributes and attribute groups can be extended ad lib to reflect additional requirements. However, the use of extended attributes will inhibit interoperability, as they only work with security implementations that support the same extended attributes.

In addition, the CORBA security model should be independent of the underlying security technology. For example, interfaces specified for security of client-target object invocations should hide the used security mechanisms from both the application objects and the ORB (except for some security administrative functions). The CORBA security model is segregated from underlying security mechanisms that are only accessible through a
standardized interface (based on GSS-API [4]). The specification illustrates how security mechanisms are integrated for a number of standard mechanisms (e.g., Kerberos5 [5–6], SESAME [7–8], SPKM [4], SSL/Transport Layer Security (TLS) [9]).

As you have already learned, interoperability problems are caused if different CORBA nodes use differing security mechanisms. In fact, there is a fundamental clash between interoperability and flexibility [2], because flexibility involves the customization of functionality, whereas interoperability can only be accomplished through standardized functionality and protocols. On the other hand, the security architecture allows the replacement of underlying security mechanisms as long as they are replaced on all participating nodes at the same time. But there are implications for middleware layer security policies and attributes. For example, authenticated identities in security policies are mechanism-specific (e.g., an X.509 certificate), which means that the content of the attribute also needs to be replaced when the authentication mechanism changes. The same applies to message protection—the model should support both symmetric and asymmetric cryptography, but all keys will have to be changed when the mechanism is replaced.

To allow application objects to be ported to domains that enforce different security policies and use different security mechanisms, security-unaware applications should be independent from the underlying middleware security system. This way, the underlying technology and policies can be replaced without affecting the application code. Note that application portability without modification is only possible for security-unaware applications, as the security is enforced and administered entirely below the application. If an object enforces application-layer security, then the interfaces to the CORBA security services should hide the particular security mechanisms used (e.g., for authentication). Note that successful porting of applications that enforce their own security to systems with different underlying CORBA security services and mechanisms depends on the attributes used to describe the application layer policy. In most cases, policies have to be changed to reflect the new underlying technology.

But in many application scenarios, CORBA security does not operate on its own. Often, a security infrastructure of some sort is already in place.

1. In a number of countries, the export of cryptographic software is regulated as part of dual-use export regulations. The security model has to take this into account and make it possible to ship a CORBA security services implementation without the actual encryption part. The security service can then be integrated afterwards with a local cryptographic package.
that has to be integrated into the CORBA system. Therefore, if the system already provides security protocols and mechanisms, it should be possible to reuse these without the need for new cryptosystems, logons, access control repositories, user registries, or policy databases. To achieve that, the architecture is specified to be mostly independent from the underlying security mechanism and, thus, can integrate with a wide range of security mechanisms and environments, as long as the mechanisms provide sufficiently well-specified interfaces. In practice, this is only possible if the source code of the CORBA security service implementation is available, so that preexisting mechanisms can be integrated into the security architecture.\(^2\) The specified model supports a number of different security policy types for access control and audit, and integrates with various different security mechanisms, which can be reused if they are already in place in the particular environment. However, the model does not provide mechanism-independent security policy attributes, in particular for access control and audit.

In addition to integrating preexisting security mechanisms, the model should allow the provision of consistent security policies across heterogeneous systems that contain legacy applications. The middleware architecture can enforce security for legacy system components by putting so-called CORBA wrappers around them, which allow the provision of consistent security policies, as long as the CORBA security systems (and the underlying security technology) match on all participating nodes.

### 3.2.4 Scalability

The security model should support CORBA systems of different size, ranging from small to very large. The security model as such does not impose any size restrictions; potential upper limits to the number of participants or policy entries are purely implementation-specific. The size limit for secure CORBA systems is often restricted by underlying security mechanisms’ scalability restrictions and policy implementation.

But the CORBA security architecture should provide the means to make administration of large-scale secure systems easier. To reduce administrative overhead, individual identities should be grouped into *roles* (or *groups*) with the same privileges. Analogously, objects that share the same security policy should be grouped into *domains*. Note that despite its support

---

\(^2\) The source code of both MICO, the example of ORB used throughout this book, and its MICOSec security services implementation, is freely available under the GNU public license (see Section 4.2).
for roles and domains, the current architecture does not include any tools to manage them in an effective and interoperable way. The OMG is currently working on an additional security specification to manage domain membership [10] in order to solve this problem.

There is also a requirement to manage the distribution of cryptographic keys securely and with low administrative overhead. To provide key management facilities, the OMG currently specifies how public key infrastructures (PKI) should be integrated into CORBA security [11]. Again, the upper limit depends on the actual PKI implementation.

3.2.5 Reliability and Assurance

Assurance is a qualitative measure of trustworthiness [1]. A security architecture is trustworthy only if all applicable security policies are enforced on all actions. In the context of CORBA, this means that it should not be possible for malicious principals to bypass the security system and invoke a method on the target without triggering the security policy enforcement and protection required by the security policy. The policy might state in some cases that no security functionality should be applied, but this has to be determined as part of the security policy evaluation process, and so the security system is not bypassed after all.

CORBA provides standard ORB interfaces at which the CORBA request traffic can be intercepted as it gets passed up or down through the ORB message path. These so-called interceptors are a convenient place to integrate the security system with the ORB. However, this means that not just the security service implementation and all underlying security mechanisms have to be trusted to work reliably but, in fact, the whole ORB also has to be trusted. The U.S. Trusted Computer System Evaluation Criteria (TCSEC) [12] calls the set of all system components that has to be trusted the trusted computing base (TCB) and suggests that security-relevant components be segregated into a (preferably small and well-understood) security kernel. The underlying idea is that the system’s overall security is only as high as the security of the weakest component in the TCB.

Unfortunately, this monolithic TCB approach is not suitable for many CORBA environments, partly because of the complexity of distributed middleware systems, and partly because secure interoperability of CORBA applications and ORBs is often based on mutual suspicion. Therefore, the specification defines the concept of a distributed TCB, which denotes the collection of objects and mechanisms that must be trusted so that end-to-end security between client and target object is maintained.
The usefulness of such a definition for real-world systems is questionable, in particular since it is impossible to put this large number of TCB components into a small and well-understood security kernel. A distributed TCB in CORBA can potentially contain the whole stack below the application object: the ORBs and object adapters, the associated ORB services, the security services, underlying security technology, the supporting operating system, and the lower layer communications software (see Figure 3.1). The distributed TCB does not extend across several nodes if there is mutual suspicion between the caller and target. But this is normally not a problem as it is infeasible for attackers to bypass the remote TCB from the network (unless someone hacks into the underlying operating system). For example, even a malicious client with a rogue application program, ORB, and security service cannot invoke methods on any target without triggering the target-side security enforcement—the application object does not listen to any network ports, it is the ORB that will pick up any requests. Therefore, any irregularities in the authentication or message protection mechanisms (e.g., disabled encryption) would be noticed by the target security service and the request would not be passed up into the application.

But trust is also highly related to reliability, which is more an implementation issue than an architectural concern. Any CORBA security system may become unreliable if software bugs in the TCB can be exploited and, due to the high degree of complexity, bugs are almost unavoidable. But, of course, this applies to any complex system, not just to CORBA and CORBA security. For example, one of the authors (as an independent security

![Figure 3.1 Components of the distributed TCB.](image)
auditor) once encountered a CORBA security system that would encode a signed session ID into the transmitted byte stream to allow the receiving side to associate the message with a particular session. Messages without ID would be transmitted unencrypted. One day, the problem occurred that, due to inverse byte ordering on differing underlying hardware and software platforms, neither side could find any session IDs and assumed that all communications should be unencrypted. All testing had been done in a homogeneous environment where this bug did not manifest. It was assumed by the developers that the underlying Java Virtual Machine (in line with its specification) would abstract from these underlying differences. In summary, the resulting system would transmit sensitive information unencrypted for CORBA communications across differing underlying hardware, just because of a bug in one of the underlying layers.

The certification process approves the trustworthiness of a system to a certain degree and, as a result, establishes confidence in the implemented security measures. According to the specification, the security model should support accreditation as defined in government criteria, such as the European Information Technology Security Evaluation Criteria (ITSEC) [13] and the Common Criteria [14]. However, in practice, it depends on the quality of the implementation process and the style and quality of the implementation, rather than on the CORBA security model itself.

3.2.6 Simplicity

To most readers’ surprise, the 420-page CORBA security services specification [1] states in the beginning that the CORBA security model should be easy to understand and administer, which implies that it should only have a few concepts and components. Following our discussion about the desire to have a trusted computing base that defines a small and well-understood security kernel, it is clear that a simple model would be likely to result in a more trustworthy system. A simple model is normally easier to implement; therefore, the scope for bugs and misinterpretations is limited.

However, as the mere page number of the specification shows, the CORBA security architecture does not achieve this target. This is mainly because it was designed to incorporate a large number of features for a number of differing environments and scenarios (e.g., automatic security enforcement for security unaware applications but also fine-grained security functionality for security-aware applications). As a logical consequence of that, and because of the inherent complexity of distributed applications security, the resulting architecture is bound to be complex as well.
In Section 3.3, you will get a feel for the complexity of the security model, even though many tricky details are hidden at this highly conceptual level. It all makes sense when a number of boxes are drawn and the flow of data between them is described informally, but many complications only become obvious when the model is actually implemented in practice. You will learn more about intricate technical difficulties of practical CORBA security in Chapters 5 through 7.

3.3 Architectural Components

The CORBA security architecture consists of a number of functional components that provide applications with security. This section describes the security models of the core components for authentication, session establishment, delegation, policies and domains, access control, message protection, security audit, and nonrepudiation. The purpose of the diagrams presented is to illustrate on an abstract level how the main objects for each functional component interact. Note that they do not always reflect the exact object interfaces and data flows, but are rather meant to give a conceptual understanding of the model as a whole.

In principle, the following aspects are covered for each functional component: First, the corresponding security policy will be discussed; then the policy’s evaluation process is examined; and, finally, the policy’s enforcement is illustrated together with its possible location in the architecture (ORB layer or application layer).

Despite the fact that the model is logically separated into various functional components, it is important to appreciate the many interdependencies between them. For example, if the access control component uses the peer identity as an attribute, then it relies critically on the functionality provided by the authentication component.

3.3.1 Principals and Credentials

Before talking about the functional components of the security model, it is necessary to identify who is going to populate it. In classical security models such as Bell-LaPadula [15], there are two groups of entities in a system—subjects and objects. Users want to access resources, and the security system has to make sure that access to a resource is restricted to authorized users. Other models such as Clark-Wilson [16] introduce another entity called procedure,
which controls access to resources and can only be called by certain users. In such models, it is assumed that users cannot access resources directly.

The CORBA security model is based on active subjects who can invoke operations on target objects, but only by going through a security enforcement component. Subjects and objects can have an identity and a number of privileges (or credentials), which the security enforcement is based on.

3.3.1.1 Principals

A subject in CORBA security is an active entity in the system that tries to use the system or its resources. A subject is assumed to have an independent will, which makes it different from other system entities that only react to invocations. Subjects can be human users, but nonhuman subjects also exist. Programs can be nonhuman subjects if they carry out actions without specific commands from users, such as hardware devices or active software daemons.

In the security model, active entities are called principals, and one of the central properties of a principal is its identity. An identity describes its principal uniquely in the sense that no two principals’ identities may be the same. But at the same time, principals may have several different kinds of unique identities. For example, a principal may have both a unique audit identity and a unique access identity.

It may not be clear why not only human users, but also software components should be principals in the sense defined above. After all, they are always started by a user (or administrator) and could therefore—as in the operating systems world—inherit the identity of that user. The reason is that objects in complex CORBA systems often need to be authenticated by their own separate identity and not by the identity of the administrator who started it. An administrator might, for example, start a whole range of services of differing quality (e.g., response time, bandwidth), and the caller wants to make sure it is actually connecting to the type of service for which it is paying. This would be impossible if all these services had the same identity inherited from the administrator. To achieve this, each component has to authenticate itself to the CORBA security services with some stored authentication information, such as an identity certificate.

3.3.1.2 Credentials

The CORBA security system has to decide what it will do with each principal’s actions. The information used for this decision is stored in the security policy. To enforce the policy, the system needs to know for sure who the
principal is, but it also needs to be able to describe the security-relevant properties for all principals, which are called credentials.3

The goal of the CORBA authentication process is the generation of such credentials.4 Credentials are the information that describes the security attributes of a principal [1], whereby attributes can be identities or privileges of the principal (or both). A privilege is a security attribute that, as opposed to an identity, does not need to be uniquely associated with a principal. Examples of privileges include groups, roles, and clearances. An identity could be the name of the user, whereas a role could be his or her job title (e.g., administrator). The security attributes described in the credentials express the principal’s characteristics, which form the basis of the system’s policies governing that subject.

Whenever the CORBA security system encounters a new and unknown subject, it automatically assigns a default credential to it, which contains no identity and only one privilege attribute, called “public.” For such subjects, a default policy will be enforced. For some subjects, this may be sufficient. Other subjects require more (nonpublic) privileges, which can be done by authenticating to the security system (see Section 3.3.4). After establishing the authentic identity, the security system will assign additional attributes to it, based on the security policy (see Figure 3.2).

3.3.2 Administration: Policies and Domains

If all objects and principals in large-scale distributed systems were administered individually, security management would become more and more cumbersome the larger the systems get. Therefore, objects that have common security requirements are grouped into domains. A domain is a distinct

![Figure 3.2 Credentials and security attributes.](image_url)

3. Note that not only clients can have credentials. Targets can also have credentials that contain their identity attributes, etc.

4. Credentials can also be obtained through delegation, which will be described in Section 3.3.3.
scope, within which certain common characteristics (policies) are exhibited and common rules are observed (see Section 3.2.4). Domains can bring the security administration down to a manageable size.

CORBA security policies specify the level of protection required in the domain, as well as the actions to be taken for each event. The result of the policy evaluation depends on a number of security attributes, such as the caller’s authenticated identity and its other credentials. Domains can be different for different security policies (e.g., access control domain, audit domain). The ORB makes sure that the policies that apply within a domain are automatically enforced for each object in that particular domain. In practice, domains could be centrally managed in a domain server that contains the policies for its domain (e.g., access control or audit domain server).

Note that cryptographic keys may be necessary for clients to authenticate themselves to the domain server and to protect sensitive credentials from unauthorized access. These keys need to be set up for all clients, and there must be keys for secure communication between domain servers from different domains. Such key infrastructure support is currently not specified for CORBA, but there are some current efforts to integrate PKIs [11].

Several types of domains are relevant to security: security policy domains, security environment domains, and security technology domains. On a lower level of abstraction, protection domains are used to cluster objects that trust each other.

3.3.2.1 Security Policy Domains and Roles

Security policies are the rules and criteria that specify which security features have to be enforced to protect objects and resources. CORBA security policies define the rules for authentication, secure invocation, privilege delegation, access control, audit, and nonrepudiation [1]. Keeping the policies independent from the application code allows security administration for security-unaware objects without changing the application code.

Domains

A security policy domain clusters all objects to which a common security policy applies (see Figure 3.3). This way, it is possible to administer many objects with only a few policies, which helps deal with the problem of scale of distributed object systems. In addition, administrators’ authorities can be separated by delimiting the scope of administrative activities to one domain (this is often called “separation of duties”).

When an object is created, it automatically becomes a member of one or more domains and, therefore, is subject to the security policies of those
domains. In other words, a security policy domain is the scope over which a common security policy is enforced. A domain can have subdomains that reflect organizational subdivisions (e.g., departments).

In CORBA, it is possible to have overlapping policy domains, either of the same type or of different types (e.g., access control policy, audit policy). Also, domains can be federated, whereby each domain keeps most of its authority but agrees to give members of other domains limited rights. A federation must be able to handle policy differences across domains (e.g., mapping of access roles across domains).

Table 3.1 illustrates conceptually how such a policy table could look for a particular domain “Bank”.

**Roles**

So far, we have grouped objects into security policy domains. In larger systems, it also makes sense to cluster principals into roles (or groups) to reduce administrative overhead (see Section 3.2.4 and Figure 3.4). This way, the policy does not need to state how every individual identity has to be treated.

![Figure 3.3 Security policy domains.](image-url)

<table>
<thead>
<tr>
<th>Policy for domain “Bank”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caller identity = “Rudolf” Invoked operation = “balance”</td>
</tr>
<tr>
<td>Caller identity = “Ulrich” Invoked operation = “withdraw”</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>
Instead, it uses roles such as “administrator” or “auditor.” Any principal that owns such a role can carry out the actions associated with that role. Note that a role is not an identity, as it is shared among a number of principals. In particular, the audit and nonrepudiation functions often require a unique identity for each principal to be able to associate responsibilities for actions to individuals.

**ORB Layer and Application Layer Security Policy**

Security policies can be enforced either by the ORB or by the application. The ORB layer security policy is enforced by the ORB (and by the security services it used, as well as the underlying operating system (OS) that supports it). Policies of this type are transparently enforced for both security-aware and security-unaware objects.

Application security policies are enforced by security-aware application objects, which may have their own security requirements. Application security policies may either be grouped in domains or individually customized to a certain application.

Note that the contents of security policies are mostly domain-specific (e.g., specific to the used security technology), which often makes cross-domain communication impossible. To a certain extent, the specification tries to standardize the semantics of security information (e.g., access rights—see Section 3.3.6 for more details) to mitigate this problem. However, there are some fundamental problems with interoperability when policies should be flexible enough to fit a variety of different application domains [2].

![Figure 3.4 Roles.](image-url)
3.3.2.2 Other Domains

In addition to security policy domains, the specification defines some other, more conceptual, domains. These are security environment domains, security technology domains, and protection domains. Grouping principals and objects into these domains should make it easier to deploy and manage secure CORBA systems.

Security environment domains specify the scope over which policy enforcement may be achieved by some means local to that environment (i.e., not by the CORBA security services). For example, extra encryption may not be necessary when messages are being transferred between objects on the same machine, because there is no untrusted network. Environment domains should be exploited where possible, to optimize performance and resource use, since enforcement of one or more policies (and their associated mechanisms) is not needed. Two possible types of environment domains are considered in the specification: message protection domains, where integrity and confidentiality protection are available by some specific means; and identity domains, where objects can share the same identity.

Security technology domains are domains in which common security mechanisms are used to enforce policies. For example, the same technology is available for principal authentication and the same authentication services are used, or data in transit is protected in the same way, using common key distribution technology with identical algorithms. The purpose of security technology domains is to identify which objects use the same underlying security services. Distinguishing this type of domain helps set up and maintain the domain’s underlying services and administer entities in the way required by this security technology. Also, security technology domains can help identify where bridges and gateways are necessary for interoperability between security technology domains. Note that the specification does not include any security technology-specific administration interfaces.

Protection domains cluster components are assumed to trust each other. The security architecture should be designed to make sure (at least in theory) that components from different protection domains cannot interfere with each other. Maintaining integrity and confidentiality in a secure object system depends on proper segregation of objects, which may include separating security services from other components. It must be possible to guarantee that, to any required level of assurance, the applications cannot bypass them. Moreover, security services themselves must also be subject to security policies. The general approach is to establish protection boundaries around groups of components that are said to belong to a protection domain.
Components belonging to a protection domain are assumed to trust each other and no security is needed, but communication across protection boundaries has to be controlled. It must be possible to constrain interaction between components to controlled communication paths, for example, explicit message passing and implicit sharing of memory.

### 3.3.2.3 Security Administration

The purpose of security policy domains and roles is to make policy administration of large systems easier. In addition to domain management, security administration also involves setting up domain servers if required, policy management, interdomain interoperability and policy agreement between domains, security mechanism installation and management, providing secure storage for audit trails and keys, and security service set-up. In general, security administration tries to bring together all the different components in such a way that they provide a secure system without any loopholes (as opposed to just individual security-enforcing components).

In this section, we will only discuss policy and domain administration. The CORBA security service defines a DomainManager interface that allows policy objects to be created, deleted, and updated for different security functions. Interfaces are provided to locate DomainManagers, but management of policy domains and their members is currently not supported. Note that there is a submission for a Security Domain Membership Management Service (SDMM) [10], which maps objects to domains. This way, security enforcement can be based on unchanging domain names rather than the less useful interface type or the often unpredictable object reference.

Managing security environment domains and security technology domains is often done in an environment-specific and security technology-specific way; thus no interfaces are specified in the CORBA security service. The specification also does not cover administrative functions concerning the management of underlying mechanisms supporting the security services, such as authentication services, key distribution services, or certification authorities. Note that this raises some chicken and egg problems that have to be carefully considered.

### 3.3.2.4 Interoperability

Secure interoperability between objects from different security domains is often only possible if they both happen to use the same security attributes and mechanisms, or if a gateway translates object references and messages between domains. It is not a goal of CORBA to specify such security
gateways (to translate security mechanisms) and bridges (to translate policies), as their functionality is highly implementation-specific.

Instead, the specification tries to mitigate the problem by specifying tokens for some security mechanisms (e.g., Kerberos) and some standard security attributes (e.g., standard access rights), which have to be supported by all compliant implementations. This way, objects can at least interoperate across domain boundaries with a minimum level of protection.

### 3.3.3 Privilege Delegation

In distributed object systems like CORBA, an invoked object may, in turn, call other objects to perform parts of the task. The resulting chain of calls complicates the credentials model (see Section 3.3.1), as credentials will need to be passed from one object to the other, and intermediate objects will enforce their own policy on these credentials before they pass them on to the next target. For example, access decisions may need to be made at each point in the chain, whereby intermediate objects may use different authorization schemes and may, therefore, require different access control information to check which objects in the chain are permitted to invoke further operation on other objects.

This process is called privilege delegation—the act whereby one principal authorizes another to use its identity or privileges, perhaps with restrictions. The owner of the original credential is called the *initiator* of the request, and the recipient of a delegated request is called an *intermediate*. If an intermediate chooses to turn the initiator’s received credential into its invocation credential, then it becomes a *delegate*. The final recipient of the request (i.e., the object that performs the requested operation) is called the *target* of the request.

In privilege delegation, the initiating principal’s security attributes may be delegated to further objects in the chain to give the recipient the right to act on its behalf. Intermediate objects may need to use their own credentials for some operations and delegated ones for others. The delegation model allows the initiator to restrict delegation of some of its security attributes. If no restrictions are placed and only the initiator’s privileges are being used, this is called impersonation.

#### 3.3.3.1 Policy

There are delegation policies for initiators, all intermediates, and targets, which reflect the interests of all involved parties.
• The initiator can decide whether or not to allow its credentials to be delegated (by setting the `DelegationMode` either to `Delegate` or to `NoDelegate`). This is an important decision because it is often not easy to revoke delegated credentials once they have been given away.

• Each intermediate’s policy can specify (on a per-interface granularity) if it should pass on the initiator’s credentials (simple delegation), its own credentials (no delegation), or a combination of them (composite delegation, combined privileges delegation, traced delegation). Figure 3.5 shows the different delegation schemes supported by the CORBA security model.

• The target policy specifies which requests should be accepted, based on the way the associated credentials have been delegated. For example, an invocation could only be granted if composite delegation has been applied, so that the whole delegation chain can be evaluated.

3.3.3.2 Enforcement

Controls can be enforced on the client side (including initiators and intermediates) before initiating object invocations. Interfaces allow control of privileges delegated, control of target restrictions, and control of time restrictions, which means that a client can control the delegation of its credentials by specifying a time window during which the delegated privileges are valid. The privileges will automatically expire outside this time interval. The client can also specify the maximum number of method invocations for which a delegated credential is valid.

The intermediate object is able to extract received privileges from the request and the active security context, and use them in local security decisions, or when making the next invocation. It can also build (if permitted) new credentials with changed attributes, using the received ones.

The target object uses the received privileges for local security decisions when the delegated request arrives, for example, for access control. Note that all delegation policies are, in fact, enforced by each receiving side, and ultimately of the target object that carries out the operation intended by the initiator. All the client can do is protect the integrity of its delegation credentials so that no intermediate can modify them, but it cannot control what intermediates do with these credentials. For example, a malicious intermediate could pass on credentials even if the `NoDelegate` option is selected. It is
then up to the target to find out during authentication that the credentials do not belong to the intermediate and, consequently, reject the request.

3.3.3.3 ORB Layer and Application Layer

Of course, only security-aware applications can select delegation schemes and specify target restrictions for themselves. For security-unaware applications, the administrator has to specify which delegation policies should be used by default when an object acts as an intermediate. This allows many
applications to be unaware of the delegation options used, as many controls are done automatically by the ORB when the intermediate invokes the next object in the chain. The difference between security-aware and security-unaware intermediate objects is illustrated in Figure 3.6.

Note that CORBA security reuses delegation functionality of the underlying security mechanism and is, therefore, limited to the functionality provided by the security mechanism (which is, in most cases, inadequate or nonexistent). However, the Common Secure Interoperability (CSI) Version 2 Specification [17] solves this problem by introducing an additional protocol layer that supports the use of delegation tokens.

3.3.4 Principal Authentication

The CORBA security model divides the process of discovering who a principal is into three basic steps. First, the system asks a principal to identify itself. Then it authenticates the claimed identity, because the principal might be lying. Finally, after establishing that the identity is authentic, it checks the privileges of the principal within the system.

In other words, authentication is the process of verifying a claimant’s claimed identity, more specifically “The verification of a claimant’s entitlement to use a claimed identity and/or privilege set” [1]. Different types of authentication information, such as passwords or cryptographic keys, are used to establish a claimant’s entitlement to a claimed identity. The critical point here is that the authentication information is assumed to be known only to the entity associated with the identity. In the case of passwords, this means that a claimant has to know the corresponding password during login to verify the claimed username. In the case of cryptographic keys, the claimant has to provide a valid certificate (signed by a certification authority) that binds his cryptographic key to his identity. The cryptographic certificate validation mechanism then ensures that this binding is valid.

The authentication process bootstraps the entire security system. All other components rely, in one way or another, on the privileges verified during the authentication process. For example, access control is often based on the caller’s identity, which relies on successful authentication. The same applies to audit, where logged events normally have to be associated with the event initiator to provide accountability.

Note that this section is only concerned with the authentication of principals to the system. Peer authentication for remote communications is covered as part of the secure context establishment (see Section 3.3.5). In the CORBA security model, peer authentication relies on the credentials