from certain authorized users through or that it does not allow particular types of network packet, such as telnet or rlogin requests, from the public network into the private network.

A firewall between a private and a public network does not impact a distributed object system if object request and reply messages do not have to pass the firewall, that is if client and servers are either all connected to the private network or all on the public network. However, sometimes clients that reside on a host connected to a public network want to request operations from a server that is behind a firewall. Our direct banking application is a good example again. The client uses the Internet to connect to the direct banking server, which should for obvious reasons be shielded against attacks by a firewall. The problem is that the client needs to establish a network connection to a host that is hidden behind a firewall and that the firewall might not allow the client to establish a transport connection to the server.

Firewalls between distributed objects have to understand the encoding of object requests.

It is therefore necessary that the firewalls that are used with distributed object systems understand the message traffic that is exchanged between clients and servers. We refer to such firewalls as *distributed object firewalls* and they are designed for monitoring and controlling object requests that pass the public–private network boundary in either direction. Distributed object firewalls support filtering of object requests. These filters can be based on the identity of the server object, the requested operation, or the client that originated a particular request.

Distributed object firewalls base their decision about whether or not a transport connection can be established and whether or not to pass on request and reply messages on the content of the message. They therefore have to understand the representation for requests and replies that is used at the transport level. This means that the distributed object firewall has to understand the standard data representation that is used for marshalling object requests. A firewall for CORBA-based systems would, for example, utilize knowledge about the GIOP protocol and the OMG's Common Data Representation, which is used for marshalling requests and replies.

Firewalls have to be integrated with encryption techniques.

The fact that firewalls need to be able to interpret messages also has implications for the use of encryption. It is highly desirable to encrypt request and reply messages that are transported in a public network. A firewall, however, cannot interpret an encrypted message. Request and reply messages therefore have to be decrypted before they can enter the firewall. Whether or not the message is encrypted on the way from the firewall to the host on the private network is application-specific. The firewall therefore has to be integrated with the encryption mechanism that is used by the distributed object middleware. If, for example, the Secure Socket Layer protocol is used, the middleware has to know the secret session key so that it can decrypt the message before deciding whether to pass it on. Messages that the firewall passes on are then encrypted again using the session key.

## 12.3.2 Authentication

Authentication techniques establish trust in a principal and its credentials

The purpose of *authentication* is to establish a trusted association between the *principal*, who is using a client or on whose behalf a server is acting, and the credentials that the system maintains about this principal. Authentication is necessary to ensure, for example that the user who is using an Internet banking application is authorized by the account holder whose

bank account is manipulated. It is not only client objects that have to be authenticated, but authentication is frequently necessary for server objects, too. For example, a principal who is requested to provide bank account details or a credit card number wants to be assured that the server belongs the bank or the e-commerce site with which the principal wants to conduct business.

In order to achieve such a trusted association, clients and servers authenticate themselves with the security service of the object-oriented middleware. The security service provides authentication operations with which the client can obtain a certified set of their credentials. In order to do so, the clients and servers provide a ***security identifier***, such as the login name, and the ***authentication data*** of the principal. The authentication data is used by the service as evidence that the principal is an authorized user of the system.

The middleware stores the credentials that the authentication service provides in an attribute that the middleware keeps to store information about the session of the current principal. The CORBA `Current` object is such a session object. It uses the credentials for implementing the security of further object requests. As the credentials include, for example, the access rights that were granted to the principal, the middleware can then decide whether the principal is allowed to execute a certain operation or not.

The security service implements authentication using encryption techniques and both public and secret keys can be used for this purpose. The use of encryption avoids storing the authentication data of principals as plain text when their disclosure could be used by an attacker for masquerading purposes.

Authentication is implemented using encryption.

Authentication might also be performed using a ***challenge–response protocol***, where the security service challenges the principal during login with a phrase and the principal responds by encrypting the phrase using its private key (see Figure 12.6) The security service can then use the principal's public key to decrypt the phrase and authenticate the principal based on the assumption that only the principal could have encrypted the phrase. Using challenge–response protocols, the authentication server avoids storing authentication information at all.

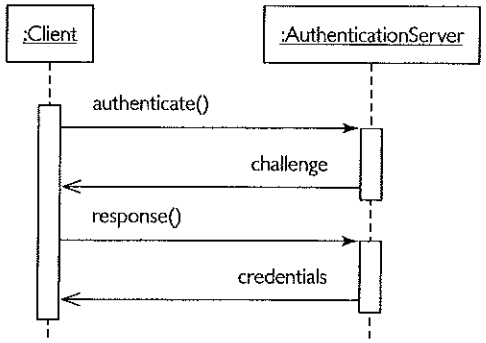


**Figure 12.6**
Use of the Challenge–Response Protocol for Authentication

### 12.3.3 Access Control

Access control mechanisms decide whether or not an object request can be granted to a client object.

We have so far assumed that a client object that has the object reference of a server object can actually request operation executions from the server. From a security point of view, this assumption is largely unreasonable. In our banking example, this would mean that a client object that can obtain an object reference of an account object can request execution of a credit operation. However, not all client objects that can obtain a reference to a server object should be able to request all operations. The *Access control mechanisms* that we discuss now provide the primitives that are needed for deciding whether an object request is granted to a particular client object or not. We discuss access control concepts using the terminology established in the CORBA Security service.

## Access Control Concepts

A principal is a human user or a system entity that is registered in and authenticated to a system.

Whether or not an object request can be granted does not so much depend on the identity of the client object, but rather on the principal on whose behalf the client object requests an operation. A principal has at least one *identity*, such as a user name, which is used as the computerized representation of the principal. Principals have security attributes that determine what they are allowed to do. These attributes are referred to as *credentials*. Figure 12.7 shows a conceptual overview of the constituents of credentials. We separate security attributes that are generally defined for all users of the system and attributes that are specific to a particular user. These specific attributes have to be validated upon login using authentication mechanisms that we discuss below.

Credentials contain the security attributes of a principal.

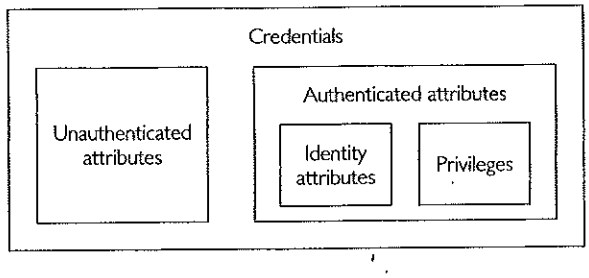


**Figure 12.7**
Security Attributes in Credential

A particular subset of the security attributes of a principal is relevant to access control. Those are referred to as *privileges*. Privileges can, for example, record that a principal has a particular role and may execute all operations that are granted to that role. Privileges are used in access policies. Two forms of access policies can be distinguished: object invocation access policies and application object access policies.

Object invocation access policies determine whether a particular principal is allowed to perform an object request.

*Object invocation access policies* are implemented by the object-oriented middleware. Object invocation access policies are enforced by the middleware and are implemented in *access decision functions*. The parameters to these access decision functions are the object reference that is to be accessed, the name of the operation that is requested and credential objects that characterize the access control privileges of the principal on whose behalf the operation is requested. Access decision functions base their decision on these parameters, on system-wide configurations and on the control attributes of individual server objects. *Control*

*attributes* are very often *access control lists*, which identify the principals or groups of principals that will be granted which operation executions.

*Application object access policies* are enforced at an application level and are, therefore, implemented by the application developer. They may use application-specific data in the implementation of the access decision function.

Application object access policies use request-specific data for access decisions.

**Example 12.10**
Invocation and Application Object Access Policies

Using object invocation policies, we can determine that only principals who have the role 'bank clerk' may execute the credit operation. Using application object access policies, we can determine that principals with the role 'bank clerk' may execute credit operations up to $5,000, while operations that credit more than $5,000 can only be executed by a principal that has the role 'bank manager'.

## Client's Perspective on Access Control

The client programmer's perspective on access control is rather simple: a request is either granted or not. Object-oriented middleware might provide an explicit interface that a client object programmer can use to check whether the right to execute a particular object request will be granted or not. If a client requests an operation execution that the principal associated with the client is not allowed to execute, an exception will be raised. In the case of object invocation access policies, the middleware will raise the exception. In the case of application object access policies, the application programmer of the server will raise the exception.

## Server's Perspective on Access Control

Access decision objects provide common functionality for application access policies to server programmers.

The server object programmer's point of view on access control distinguishes two cases, depending on whether an object invocation or an application object access policy is used. In case of the object invocation policy, access control is transparent on the server programmer as it is performed by the object-oriented middleware. If, however, an application object access policy is required access control is not transparent to the server programmer. The server object programmer has to implement the access decision function. Object-oriented middleware may provide standardized interfaces for these functions in *access decision objects* in order to support the use of a common infrastructure for application object access policy implementation. Access decision object implementations provide a boolean access decision function that determines whether or not to grant access. The parameters to these access decisions normally include:

1. the credentials of the principal that requested an operation execution,
2. the reference of the server object from which an execution is requested,
3. the requested operation, and
4. the parameters of the requested operation.

## Administrator's Perspective on Access Control

We have seen above that implementations of the object invocation access policy are transparent to both client and server object designers. The implementation is not transparent to