programmers as it is done either by the middleware or by the stubs that are created by the middleware.

Distribution of secret keys to large numbers of objects is too complex.

The disadvantage of secret key encryption for distributed objects is that the number of keys that need to be created and distributed increases quadratically by the number of objects. Public key encryption overcomes this problem.

12.2.2 Public Key Encryption

Public key encryption generate pairs of keys of which only one is made publicly available and the other is kept private.

Methods based on secret keys use the same key for both encryption and decryption. It is therefore necessary to keep that key secret. If different keys are used for encryption and decryption, there is no need to keep secret the key that is used for encryption. This is the basic idea of public key encryption methods. They provide mechanisms to generate pairs of matching keys. The first key is made public and the second key is kept private. Public key methods can be used to ensure that a message has been sent from a particular sender or that only a particular receiver can receive the message.

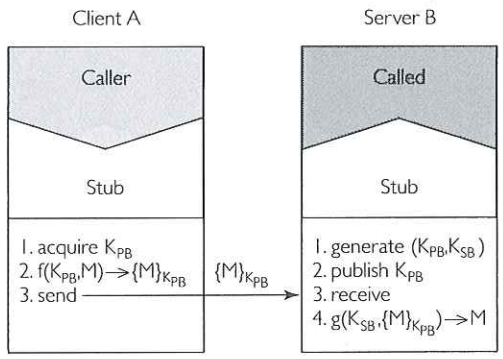

Figure 12.2
Secure Distributed Object Communication with Public Keys

Figure 12.2 shows the steps involved in transmitting messages securely using public keys. First the receiver B generates a pair of keys (K_{PB}, K_{SB}) where K_{PB} denotes the public and K_{SB} denotes the secret key. B publishes K_{PB} in a key database and retains the key K_{SB}. A sender A wishing to transmit a message M securely to B can then acquire the public key K_{PB}, apply the encryption function f with the key to the message and obtain an encrypted message $M_{K_{PB}}$. Transmission of the message across the network is secure because only B has the matching secret key and therefore only B can decrypt the message upon receipt. It does so by applying a decryption function g to the private key K_{SB} and the encrypted message.

Figure 12.2 also shows how public key encryption can be used for secure distributed object communication. The difference between public and secret key methods is that just one pair of keys is generated for every object and thus the number of keys needed is linear in relation to the number of objects. Because different functions f and g are used on the client side and on the server side, the use of public keys is more complicated for the reply message than with secret keys. For the reply message, A needs to generate a pair of keys and publish its public key K_{PA}, which B acquires to encrypt the reply message.

Public key methods have the advantage that the number of keys is only linear in relation to the number of objects. Public key encryption is used in Pretty Good Privacy (PGP) [Zimmermann, 1995]. However, they have a slight disadvantage in that execution of the encryption and decryption function is more complex. With the steady increase in computing power this is often not a problem any more. If the complexity is an issue, *hybrid encryption* approaches that combine secret and public keys are used. A typical hybrid approach that uses a public key to encrypt a secret key is the Secure Socket Layer (SSL) protocol of Netscape.

Public key encryption can be used to protect distributed object requests from eavesdropping and tampering.

SSL uses *Rivest Shamir Adelmann* (RSA) [Rivest et al., 1978] encryption, which is a public key approach and therefore RSA was incorporated into Netscape to support secure downloads of information from the Web. The SSL client generates a secret key for one session, that secret key is encrypted using the server's public key, and the session key is then forwarded to the server and used for any further communication between client and server. Most object-oriented middleware uses SSL rather than straight TCP as a transport protocol in order to prevent eavesdropping and message tampering of object request traffic.

12.2.3 Key Distribution

For both public and secret key encryption, the problem arises as to how keys are distributed in a secure manner. For secret key methods this is an obvious problem. Both a client and a server object need to be equipped with the same key. They need to obtain the key in such a way that no other object gets access to the key. In military encryption applications, keys are distributed by non-networked means, such as a messenger, but this is not feasible in a distributed object system. Hence, for distributed object applications, keys need to be distributed via the network too. While in transit, however, the key must be secured against eavesdropping and tampering, otherwise third parties could use the key for successive eavesdropping or message tampering.

Key distribution is also a problem, though less obvious, for public key methods. The request parameters that a client transmits should only become known to the desired server object. It is therefore important that the public key that the client uses for encrypting the message is the public key of the desired server object rather than a public key that was generated by an attacker. This can be achieved if a trusted key distribution service is used.

A *key distribution service* for both public and secret key encryption methods takes care of distributing keys between objects. A key distribution service has to be a trusted service and the registration of objects with that service has to be trustworthy, too. We now discuss the *Needham/Schroeder protocol*, upon which most key distribution services are based. There are different versions of the Needham/Schroeder protocol for secret and for public keys. We first consider the secret key distribution protocol because it is simpler.

Secure key distribution mechanisms are needed for both secret and public key encryption.

Distributing Secret Keys

Figure 12.3 shows an overview of the parties that are in involved in and the steps that are necessary to securely distribute secret keys. In general, the objects are a client C, a server S and a key distribution server AS. The protocol assumes that every participating object's

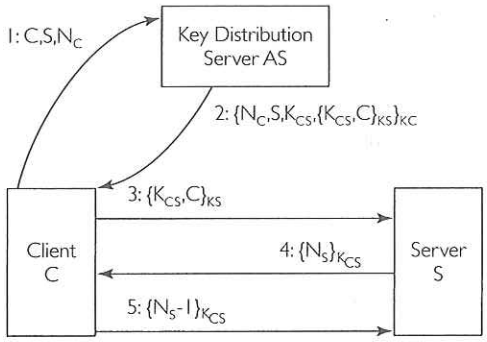

Figure 12.3
Needham/Schroeder Protocol for Secret Key Distribution

identity is registered with AS and that AS and the object share a secret key for mutual communication, which has been transmitted by different means.

A nonce is a piece of information that is used temporarily in order to detect replaying of messages.

As a first step, the client C requests generation of a secret key for communication with server S. As part of the request, the client also submits a nonce N_C that is used to avoid replaying. In response to that request, the key distribution service generates a key K_{CS} that C and S use for their communication. It then encrypts K_{CS} and the identity of the client C using the secret key KS that S uses to communicate with the key distribution service. It then encrypts this together with the nonce, the identity of S and the key K_{CS} using the key K_C that C uses for the communication with the key distribution service. In this way it is ensured that only C can decrypt the key K_{CS}. The client then provides S with the key and its identity that was encrypted by the key distribution server, knowing that only S can decrypt it. After decrypting this message, the server tests the key. To do so, the server generates a nonce N_S and returns it encrypted using K_{CS}. The client performs an agreed computation to the nonce and returns the result to the server in a message encrypted by K_{CS}.

Distributing Public Keys

The components that participate in the protocol for securely distributing public keys are the same as for secret key distribution. The protocol, however, is slightly more complicated as it is has to distribute two keys for every pair of objects that wish to communicate. Client C needs to obtain the public key of S in order to send messages to S that only S can decrypt and S has to obtain the public key of C.

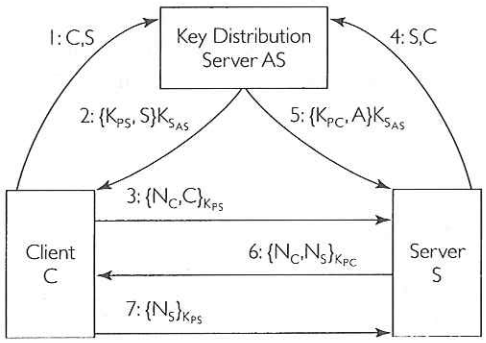

Figure 12.4
Need/Schroeder Protocol for Public Key Distribution

The protocol starts when client C requests transmission of a public key for server S from the key distribution service AS. The server then returns the public key of S encrypted with its own secret key $K_{S_{AS}}$. By decrypting it with AS's public key, the client can be certain that it can only be the key distribution service that has provided the public key of S. The client then sends its identity together with a nonce to the server encrypted with the server's public key K_{PS}. The server interprets this as a signal to obtain the public key for the client from the key distribution service in step 4. The service then provides that key encrypted using its own secret key, which the server decrypts using the public key of AS. The server then sends back the nonce created by the client with a freshly-generated nonce and encrypts this using the public key of the client K_{PC}. The client acknowledges receipt by transmitting the nonce back to the server encrypted using the server's public key.

12.2.4 Summary

We have seen how encryption can be used to prevent eavesdropping on and tampering with messages that are exchanged during the implementation of requests and replies between distributed objects. We have seen the two facets of encryption, secret and public key. They both assumed the availability of keys for the encryption of messages. We have discussed two variants of the Needham/Schroeder protocol with which the necessary key distribution can be achieved.

12.3 Higher-Level Security Services

We will now study higher-level security mechanisms and we will see that the encryption primitives that we have discussed above will also be used for preventing masquerading and for providing evidence that can be used in legal proceedings.

12.3.1 Firewalls

Figure 12.5 shows the principal idea of a *firewall*. The aim of all firewalls is to create a barrier that makes it as difficult as possible for an intruder from a public network to obtain network access to objects that execute within in the private network. To achieve this, access to the host of the firewall is usually extremely restricted and the security is administered with exceptionally great care to make it difficult to break into the host and somehow circumvent the firewall. The host of the firewall is therefore also sometimes referred to as a *bastion host*.

Firewalls are gateways that tightly control message traffic between private and public networks.

The levels of control that are exercised by a firewall can vary greatly. A firewall may provide only the facilities to monitor and audit the network traffic that passes the public–private network boundary. A firewall may also be set up in such a way that it only routes packets

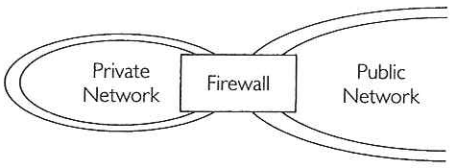

Figure 12.5
Separating Private and Public Networks with a Firewall