PART 2: NETWORK SECURITY APPLICATIONS

CHAPTER 4

KEY DISTRIBUTION AND USER AUTHENTICATION

No Singhalese, whether man or woman, would venture out of the house without a bunch of keys in his hand, for without such a talisman he would fear that some devil might take advantage of his weak state to slip into his body.

—*The Golden Bough*, Sir James George Frazer

This chapter covers two important, related concepts. First is the complex topic of cryptographic key distribution, involving cryptographic, protocol, and management considerations. This chapter gives the reader a feel for the issues involved and provides a broad survey of the various aspects of key management and distribution.

This chapter also examines some of the authentication functions that have been developed to support network-based user authentication. The chapter includes a detail discussion of one of the earliest and also one of the most widely used key distribution and user authentication services: Kerberos. Next, the chapter looks at key distribution schemes that rely on asymmetric encryption. This is followed by a discussion of X.509 certificates and public-key infrastructure. Finally, the concept of federated identity management is introduced.

4.1 SYMMETRIC KEY DISTRIBUTION USING SYMMETRIC ENCRYPTION

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the key distribution technique, a term that refers to the means of delivering a key to two parties that wish to exchange data, without allowing others to see the key. Key distribution can be achieved in a number of ways. For two parties A and B, there are the following options:

1. A key could be selected by A and physically delivered to B.
2. A third party could select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party could transmit the new key to the other, using the old key to encrypt the new key.
4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is only going to be exchanging data with its partner on the other end of the link. However, for end-to-end encryption over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system.

Option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent

keys are revealed. Even if frequent changes are made to the link encryption keys, these should be done manually. To provide keys for end-to-end encryption, option 4 is preferable.

For **option 4**, two kinds of keys are used:

- **Session key:** When two end systems (hosts, terminals, etc.) wish to communicate, they establish a logical connection (e.g., virtual circuit). For the duration of that logical connection, called a session, all user data are encrypted with a one-time session key. At the conclusion of the session the session key is destroyed.
- **Permanent key:** A permanent key is a key used between entities for the purpose of distributing session keys.

A necessary element of option 4 is a **key distribution center (KDC)**. The KDC determines which systems are allowed to communicate with each other. When permission is granted for two systems to establish a connection, the key distribution center provides a one-time session key for that connection.

In general terms, the operation of a KDC proceeds as follows:

1. When host A wishes to set up a connection to host B, it transmits a connection-request packet to the KDC. The communication between A and the KDC is encrypted using a master key shared only by A and the KDC.
2. If the KDC approves the connection request, it generates a unique one-time session key. It encrypts the session key using the permanent key it shares with A and delivers the encrypted session key to A. Similarly, it encrypts the session key using the permanent key it shares with B and delivers the encrypted session key to B.
3. A and B can now set up a logical connection and exchange messages and data, all encrypted using the temporary session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of users to access a number of servers and for the servers to exchange data with each other. The most widely used application that implements this approach is Kerberos, described in the next section.

4.2 KERBEROS

Kerberos is a key distribution and user authentication service developed at MIT. The problem that Kerberos addresses is this: Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.

3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in use. Version 4 [MILL88, STEI88] implementations still exist, although this version is being phased out. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120).

Because of the complexity of Kerberos, it is best to start with a description of version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Then, we examine version 5.

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

A Simple Authentication Dialogue In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines. To counter this threat, servers must be able to confirm the identities of clients who request service. Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an **authentication server (AS)** that knows the passwords of all users and stores these in a centralized database. In addition, the AS shares a unique secret key with each server. These keys have been distributed physically or in some other secure manner. Consider the following hypothetical dialogue:[1]

$$\textbf{(1)}\ \mathrm{C} \rightarrow \mathrm{AS}: \quad ID_{\mathrm{C}} \| P_{\mathrm{C}} \| ID_{\mathrm{V}}$$

$$\textbf{(2)}\ \mathrm{AS} \rightarrow \mathrm{C}: \quad Ticket$$

$$\textbf{(3)}\ \mathrm{C} \rightarrow \mathrm{V}: \quad ID_{\mathrm{C}} \| Ticket$$

$$Ticket = \mathrm{E}(K_v, [ID_{\mathrm{C}} \| AD_{\mathrm{C}} \| ID_{\mathrm{V}}])$$

[1]The portion to the left of the colon indicates the sender and receiver, the portion to the right indicates the contents of the message, and the symbol ‖ indicates concatenation.

where

C = client
AS = authentication server
V = server
ID_C = identifier of user on C
ID_V = identifier of V
P_C = password of user on C
AD_C = network address of C
K_v = secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V. The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password. The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V. If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic. To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID. This ticket is encrypted using the secret key shared by the AS and this server. This ticket is then sent back to C. Because the ticket is encrypted, it cannot be altered by C or by an opponent.

With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket. V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message. If these two match, the server considers the user authenticated and grants the requested service.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (ID_V) is included in the ticket so that the server can verify that it has decrypted the ticket properly. ID_C is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, AD_C serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name ID_C, and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

A More Secure Authentication Dialogue Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail-server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme, it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

To solve these additional problems, we introduce a scheme for avoiding plaintext passwords and a new server, known as the **ticket-granting server (TGS)**. The new (but still hypothetical) scenario is as follows.

Once per user logon session:

(1) C → AS: $ID_C \| ID_{tgs}$

(2) AS → C: $\mathrm{E}(K_c, Ticket_{tgs})$

Once per type of service:

(3) C → TGS: $ID_C \| ID_V \| Ticket_{tgs}$

(4) TGS → C: $Ticket_v$

Once per service session:

(5) C → V: $ID_C \| Ticket_v$

$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$

$Ticket_v = \mathrm{E}(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$

The new service, TGS, issues **tickets** to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password (K_C), which is already stored at the AS. When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user and the ID of the TGS.

This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket to spoof the TGS. To counter this, the ticket includes a **timestamp**, indicating the date and time at which the ticket was issued, and a **lifetime**, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4.

3. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
4. The TGS decrypts the incoming ticket using a key shared only by the AS and the TGS (K_{tgs}) and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.

5. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

The Version 4 Authentication Dialogue Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. The heart of the first problem is the lifetime associated with the ticket-granting

ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that there may be a requirement for servers to authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server then would be in a position to act as a real server, capture any information from the user, and deny the true service to the user.

We examine these problems in turn and refer to Table 4.1, which shows the actual Kerberos protocol.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires.

Table 4.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

(2) AS → C $\mathrm{E}(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

$$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $ID_v \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $\mathrm{E}(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$

$$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$$

$$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$$

$$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,tgs}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_3])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Ticket_v \| Authenticator_c$

(6) V → C $\mathrm{E}(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)

$$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$$

$$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,v}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_5])$$

(c) Client/Server Authentication Exchange to obtain service

To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information, again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 4.1a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_C), that contains the ticket. The encrypted message also contains a copy of the session key, $K_{C,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_C, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table 4.1b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{C,tgs}$. In effect, the ticket says, "Anyone who uses $K_{C,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{C,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 4.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Table 4.2 summarizes the justification for each of the elements in the Kerberos protocol, and Figure 4.1 provides a simplified overview of the action.

Table 4.2 Rationale for the Elements of the Kerberos Version 4 Protocol

Message (1)	Client requests ticket-granting ticket.
ID_C	Tells AS identity of user from this client.
ID_{tgs}	Tells AS that user requests access to TGS.
TS_1	Allows AS to verify that client's clock is synchronized with that of AS.
Message (2)	AS returns ticket-granting ticket.
K_c	Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2).
$K_{c,tgs}$	Copy of session key accessible to client created by AS to permit secure exchange between client and TGS without requiring them to share a permanent key.
ID_{tgs}	Confirms that this ticket is for the TGS.
TS_2	Informs client of time this ticket was issued.
$Lifetime_2$	Informs client of the lifetime of this ticket.
$Ticket_{tgs}$	Ticket to be used by client to access TGS.

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.

ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.
TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_c	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

(c) Client/Server Authentication Exchange

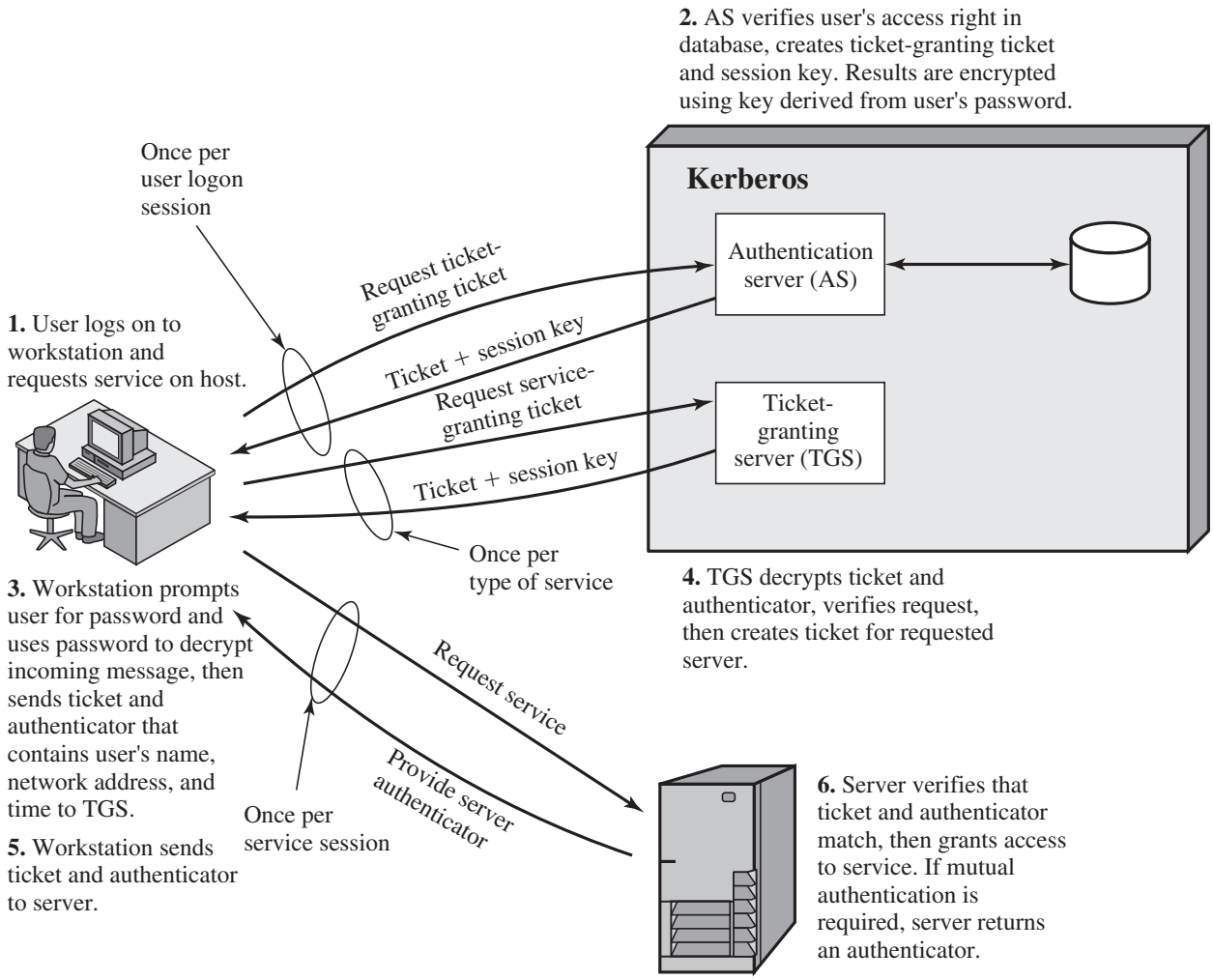


Figure 4.1 Overview of Kerberos

KERBEROS REALMS AND MULTIPLE KERBERI A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm**. The concept of **realm** can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which

is a service or user that is known to the Kerberos system. Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm also must be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 4.2): A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

The details of the exchanges illustrated in Figure 4.2 are as follows (compare Table 4.1).

(1) C → AS: $ID_C \| ID_{tgs} \| TS_1$

(2) AS → C: $\mathrm{E}(K_C, [K_{C,tgs} \| ID_{tgs} \| TS_2 \| Lifetime_2 \| Ticket_{tgs}])$

(3) C → TGS: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_C$

(4) TGS → C: $\mathrm{E}(K_{C,tgs}, [K_{C,tgsrem} \| ID_{tgsrem} \| TS_4 \| Ticket_{tgsrem}])$

(5) C → $\mathrm{TGS_{rem}}$: $ID_{Vrem} \| Ticket_{tgsrem} \| Authenticator_C$

(6) $\mathrm{TGS_{rem}}$ → C: $\mathrm{E}(K_{C,tgsrem}, [K_{C,Vrem} \| ID_{Vrem} \| TS_6 \| Ticket_{Vrem}])$

(7) C → $\mathrm{V_{rem}}$: $Ticket_{Vrem} \| Authenticator_C$

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N - 1)/2$ secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

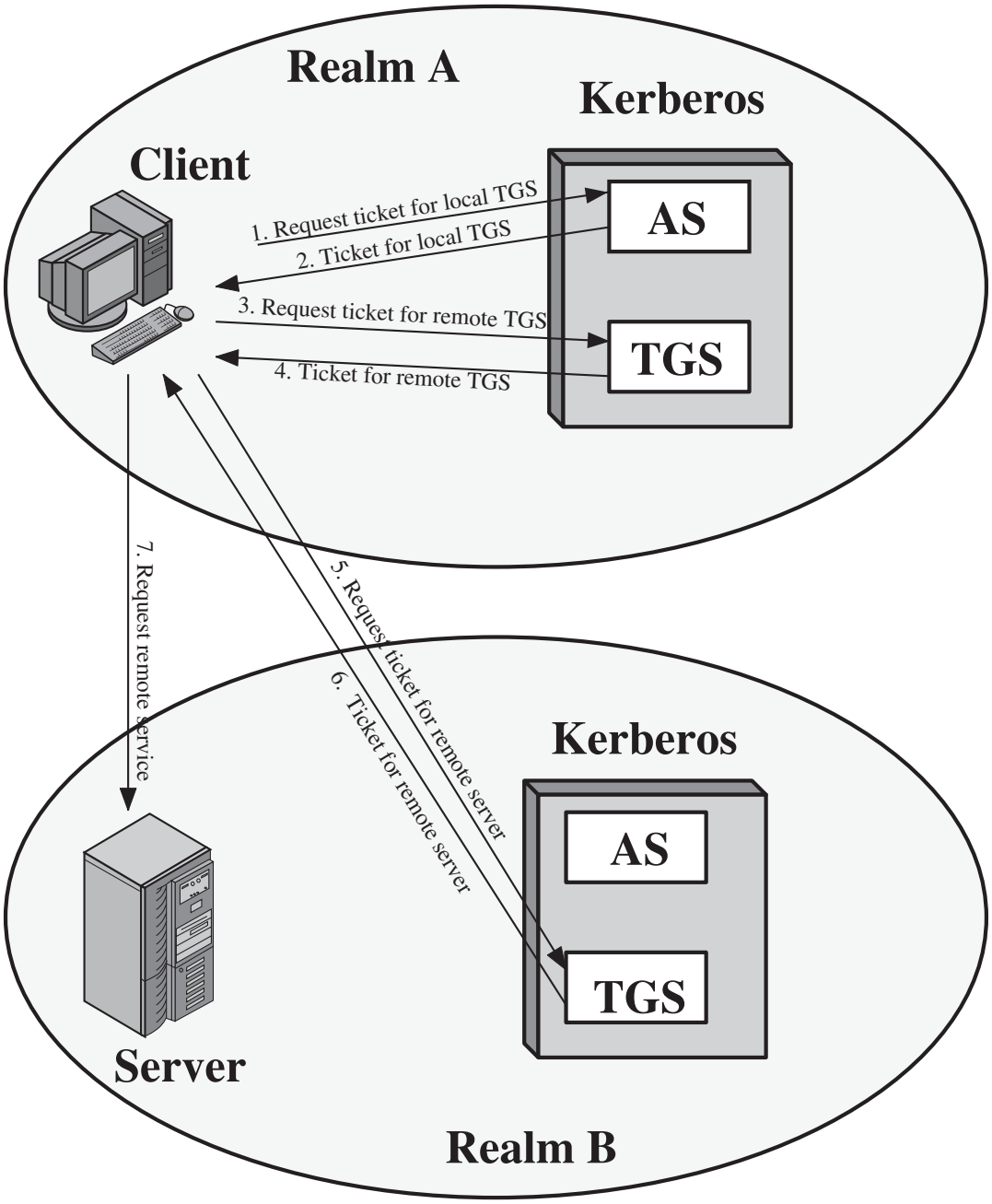


Figure 4.2 Request for Service in Another Realm

Kerberos Version 5

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

Differences between Versions 4 and 5 Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. We briefly summarize the improvements in each area. Kerberos version

4 did not fully address the need to be of general purpose. This led to the following **environmental shortcomings**.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.
2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.
4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes (a little over 21 hours). This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Interrealm authentication:** In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following.

1. **Double encryption:** Note in Table 4.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as **propagating cipher block chaining (PCBC)**.[2] It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms, allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.
3. **Session keys:** Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket. In addition, the session key subsequently may be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.
4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.[3] An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 9, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

The Version 5 Authentication Dialogue Table 4.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 4.1).

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **Realm:** Indicates realm of user.
- **Options:** Used to request that certain flags be set in the returned ticket.
- **Times:** Used by the client to request the following time settings in the ticket:

  from: the desired start time for the requested ticket

  till: the requested expiration time for the requested ticket

  rtime: requested renew-till time
- **Nonce:** A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.

[2]This is described in Appendix F.

[3]Appendix F describes the mapping of passwords to encryption keys.

Table 4.3 Summary of Kerberos Version 5 Message Exchanges

(1) C → AS $Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$

(2) AS → C $Realm_c \| ID_C \| Ticket_{tgs} \| \mathrm{E}(K_c, [K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}])$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$$

(a) Authentication Service Exchange to obtain ticket-granting ticket

(3) C → TGS $Options \| ID_v \| Times \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$

(4) TGS → C $Realm_c \| ID_C \| Ticket_v \| \mathrm{E}(K_{c,tgs}, [K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_v])$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Ticket_v = \mathrm{E}(K_v, [Flags \| K_{c,v} \| Realm_c \| ID_C \| \mathrm{AD}_C \| Times])$$

$$Authenticator_c = \mathrm{E}(K_{c,tgs}, [ID_C \| Realm_c \| TS_1])$$

(b) Ticket-Granting Service Exchange to obtain service-granting ticket

(5) C → V $Options \| Ticket_v \| Authenticator_c$

(6) V → C $\mathrm{E}_{Kc,v} [\, TS_2 \| Subkey \| Seq\# \,]$

$$\mathrm{Ticket}_v = \mathrm{E}(K_v, [\mathrm{Flags} \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

$$Authenticator_c = \mathrm{E}(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$$

(c) Client/Server Authentication Exchange to obtain service

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

- **Subkey:** The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{C,V}$) is used.

- **Sequence number:** An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5, because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5). The optional sequence number field specifies the starting sequence number to be used by the client.

4.3 KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

One of the major roles of public-key encryption is to address the problem of key distribution. There are actually two distinct aspects to the use of public-key encryption in this regard.

- The distribution of public keys.
- The use of public-key encryption to distribute secret keys.

We examine each of these areas in turn.

Public-Key Certificates

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large. Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

The solution to this problem is the **public-key certificate**. In essence, a certificate consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. Figure 4.3 illustrates the process.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic

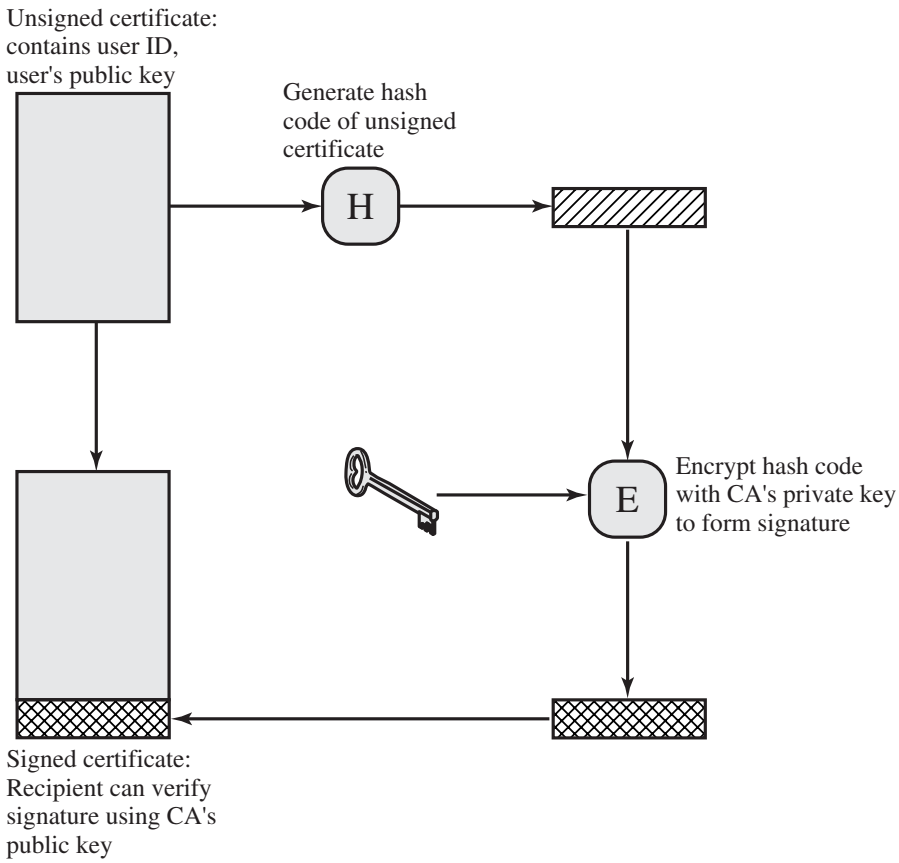


Figure 4.3 Public-Key Certificate Use

transactions (SET), and S/MIME—all of which are discussed in Part Two. X.509 is examined in detail in the next section.

Public-Key Distribution of Secret Keys

With conventional encryption, a fundamental requirement for two parties to communicate securely is that they share a secret key. Suppose Bob wants to create a messaging application that will enable him to exchange e-mail securely with anyone who has access to the Internet or to some other network that the two of them share. Suppose Bob wants to do this using conventional encryption. With conventional encryption, Bob and his correspondent, say, Alice, must come up with a way to share a unique secret key that no one else knows. How are they going to do that? If Alice is in the next room from Bob, Bob could generate a key and write it down on a piece of paper or store it on a diskette and hand it to Alice. But if Alice is on the other side of the continent or the world, what can Bob do? He could encrypt this key using conventional encryption and e-mail it to Alice, but this means that Bob and Alice must share a secret key to encrypt this new secret key. Furthermore, Bob and everyone else who uses this new e-mail package faces the same problem with every potential correspondent: Each pair of correspondents must share a unique secret key.

One approach is the use of Diffie-Hellman key exchange. This approach is indeed widely used. However, it suffers the drawback that, in its simplest form, Diffie-Hellman provides no authentication of the two communicating partners.

A powerful alternative is the use of public-key certificates. When Bob wishes to communicate with Alice, Bob can do the following:

1. Prepare a message.
2. Encrypt that message using conventional encryption with a one-time conventional session key.
3. Encrypt the session key using public-key encryption with Alice's public key.
4. Attach the encrypted session key to the message and send it to Alice.

Only Alice is capable of decrypting the session key and therefore of recovering the original message. If Bob obtained Alice's public key by means of Alice's public-key certificate, then Bob is assured that it is a valid key.

4.4 X.509 CERTIFICATES

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME (Chapter 7), IP Security (Chapter 8), and SSL/TLS (Chapter 5).

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90]; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation. Figure 4.3 illustrates the generation of a public-key certificate.

Certificates

The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user. The directory server itself is not responsible for the creation of public keys or for the

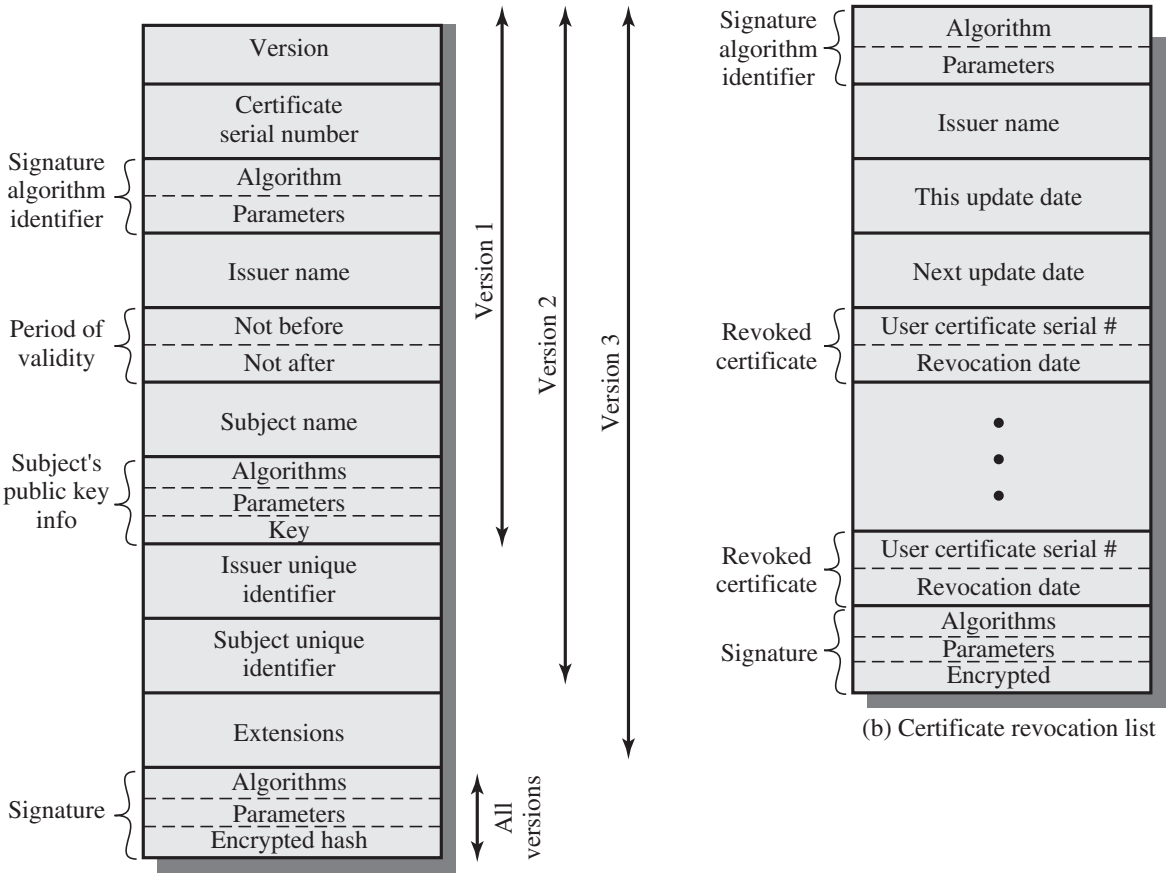


Figure 4.4 X.509 Formats

certification function; it merely provides an easily accessible location for users to obtain certificates.

Figure 4.4a shows the general format of a certificate, which includes the following elements.

- **Version:** Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate, together with any associated parameters. Because this information is repeated in the Signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 name of the CA that created and signed this certificate.
- **Period of validity:** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name:** The name of the user to whom this certificate refers. That is, this certificate certifies the public key of the subject who holds the corresponding private key.

- **Subject's public-key information:** The public key of the subject, plus an identifier of the algorithm for which this key is to be used, together with any associated parameters.
- **Issuer unique identifier:** An optional bit string field used to identify uniquely the issuing CA in the event the X.500 name has been reused for different entities.
- **Subject unique identifier:** An optional bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

The standard uses the following notation to define a certificate:

$$CA\langle\langle A\rangle\rangle = CA\{V, SN, AI, CA, UCA, A, UA, Ap, T^A\}$$

where

$Y\langle\langle X\rangle\rangle$ = the certificate of user X issued by certification authority Y

$Y\{I\}$ = the signing of I by Y; consists of I with an encrypted hash code appended

V = version of the certificate

SN = serial number of the certificate

AI = identifier of the algorithm used to sign the certificate

CA = name of certificate authority

UCA = optional unique identifier of the CA

A = name of user A

UA = optional unique identifier of the user A

Ap = public key of user A

T^A = period of validity of the certificate

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid. This is the typical digital signature approach, as illustrated in Figure 4.5.

Obtaining a User's Certificate User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

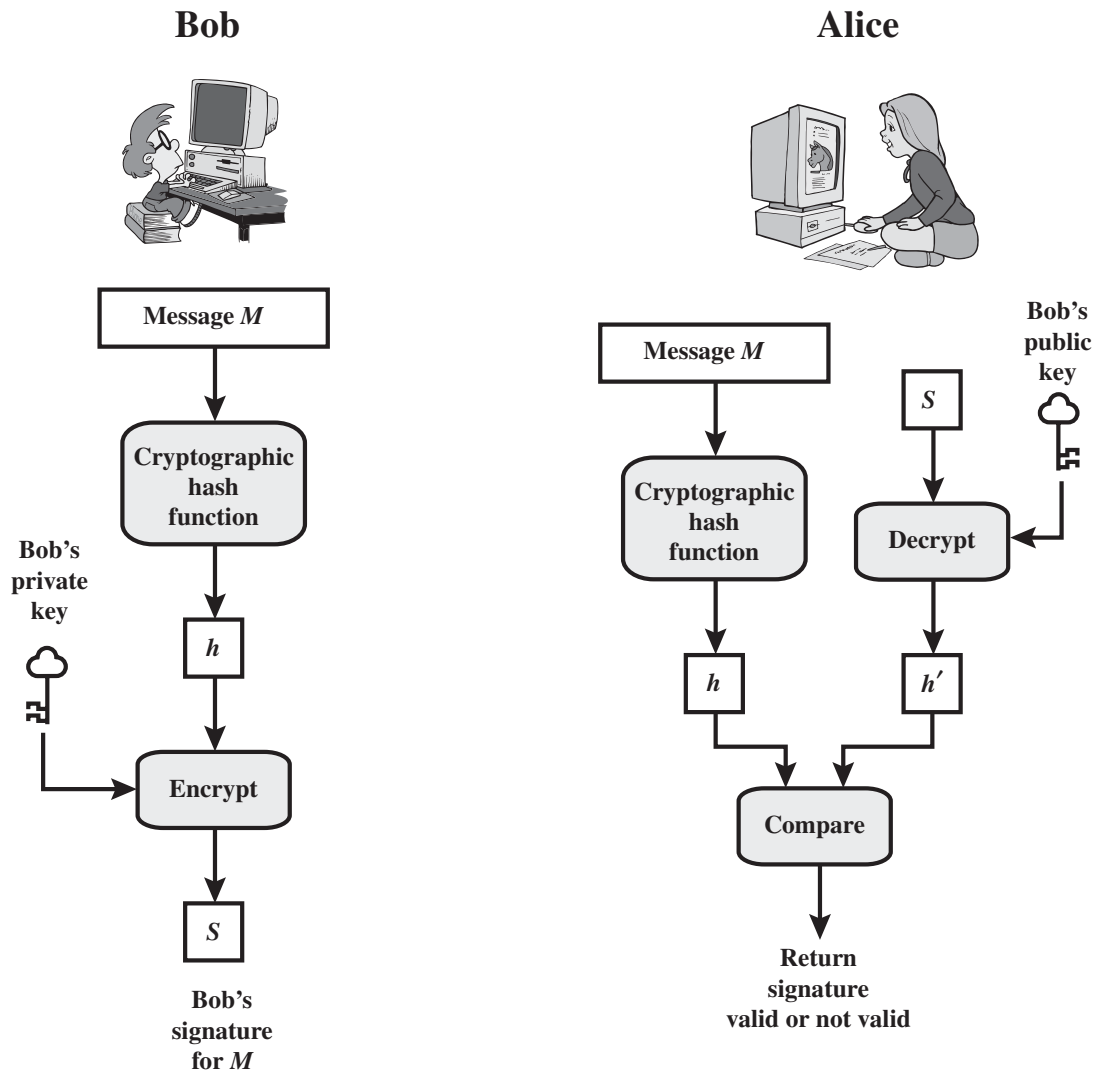


Figure 4.5 Simplified Depiction of Essential Elements of Digital Signature Process

Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure way (with respect to integrity and authenticity) so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key.

1. A obtains (from the directory) the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.
2. A then goes back to the directory and obtains the certificate of B signed by X_2. Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

$$X_1\langle\langle X_2\rangle\rangle\ X_2\langle\langle B\rangle\rangle$$

In the same fashion, B can obtain A's public key with the reverse chain:

$$X_2\langle\langle X_1\rangle\rangle\ X_1\langle\langle A\rangle\rangle$$

This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

$$X_1\langle\langle X_2\rangle\rangle\ X_2\langle\langle X_3\rangle\rangle\ldots X_N\langle\langle B\rangle\rangle$$

In this case, each pair of CAs in the chain (X_i, X_{i+1}) must have created certificates for each other.

All of these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 4.6, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs

In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

$$X\langle\langle W\rangle\rangle\ W\langle\langle V\rangle\rangle\ V\langle\langle Y\rangle\rangle\ Y\langle\langle Z\rangle\rangle\ Z\langle\langle B\rangle\rangle$$

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from

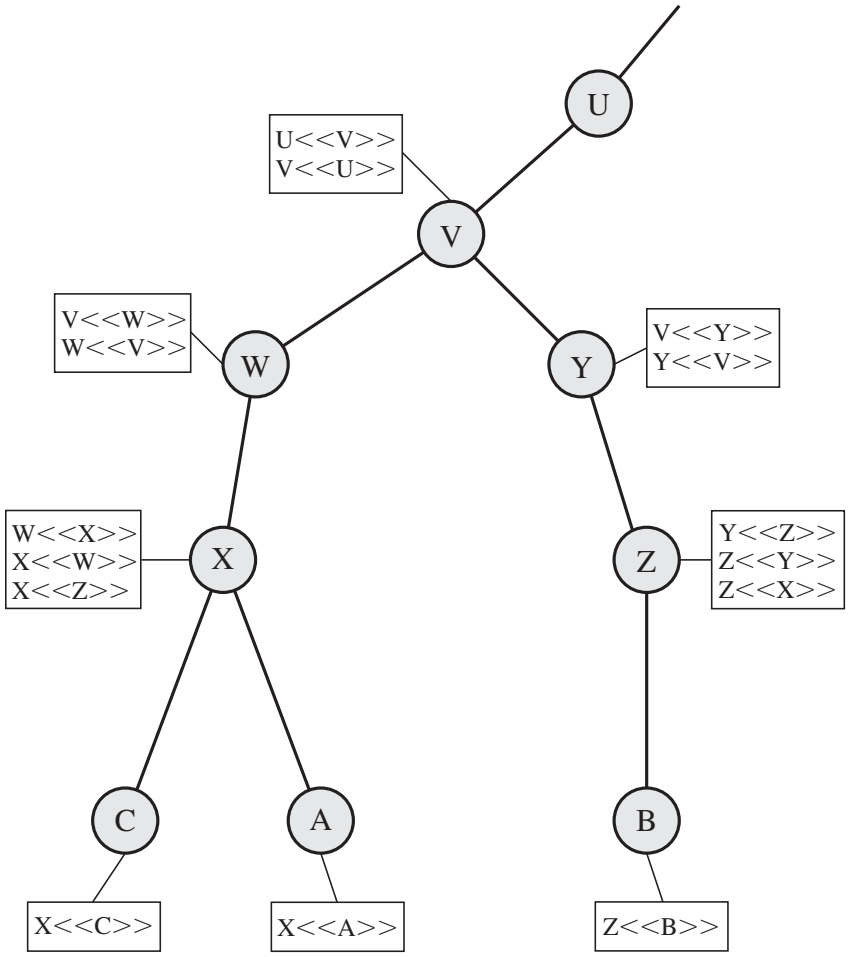


Figure 4.6 X.509 Hierarchy: A Hypothetical Example

B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the certification path:

$$Z\langle\langle Y\rangle\rangle\ Y\langle\langle V\rangle\rangle\ V\langle\langle W\rangle\rangle\ W\langle\langle X\rangle\rangle\ X\langle\langle A\rangle\rangle$$

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

Revocation of Certificates Recall from Figure 4.4 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires for one of the following reasons.

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA. Reasons for this include subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists also should be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 4.4b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate. Each entry consists of the serial number of a certificate and revocation date for that certificate. Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

1. The Subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.
2. The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
3. There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.
4. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
5. It is important to be able to identify different keys used by the same owner at different times. This feature supports key life cycle management, in particular the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a

particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).
- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, and CA signature verification on CRLs.
- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate Subject and Issuer Attributes These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.
- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.
- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

Certification Path Constraints These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs. The constraints may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

The extension fields in this area include:

- **Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.
- **Name constraints:** Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.
- **Policy constraints:** Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

4.5 PUBLIC-KEY INFRASTRUCTURE

RFC 2822 (*Internet Security Glossary*) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys. The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet. This section describes the PKIX model.

Figure 4.7 shows the interrelationship among the key elements of the PKIX model. These elements are

- **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate. End entities typically consume and/or support PKI-related services.
- **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more registration authorities.
- **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process, but can assist in a number of other areas as well.
- **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
- **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

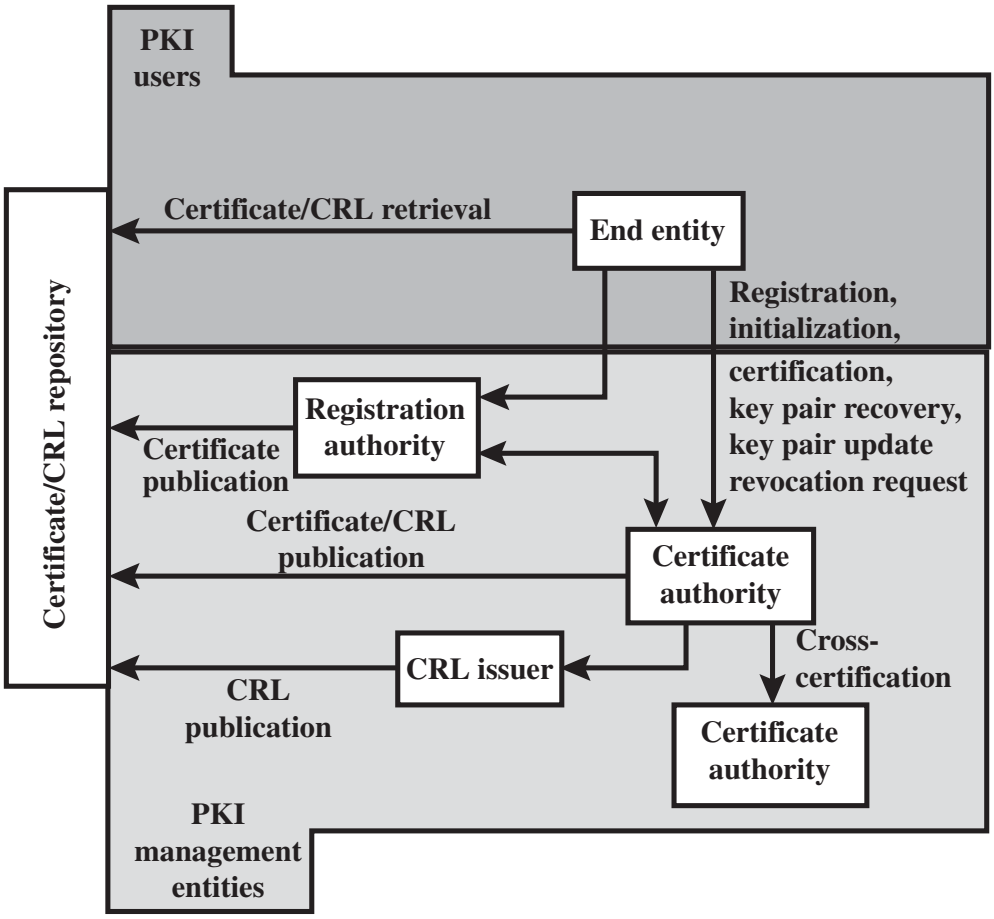


Figure 4.7 PKIX Architectural Model

PKIX Management Functions

PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 4.7 and include the following:

- **Registration:** This is the process whereby a user first makes itself known to a CA (directly, or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some off-line or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.
- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s) to be used in validating certificate paths.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key and returns that certificate to the user's client system and/or posts that certificate in a repository.

- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).
- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols

The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection. RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.

RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

4.6 FEDERATED IDENTITY MANAGEMENT

Federated identity management is a relatively new concept dealing with the use of a common identity management scheme across multiple enterprises and numerous applications and supporting many thousands, even millions, of users. We begin our overview with a discussion of the concept of identity management and then examine federated identity management.

Identity Management

Identity management is a centralized, automated approach to provide enterprise-wide access to resources by employees and other authorized individuals. The focus of identity management is defining an identity for each user (human or process),

associating attributes with the identity, and enforcing a means by which a user can verify identity. The central concept of an identity management system is the use of single sign-on (SSO). SSO enables a user to access all network resources after a single authentication.

[PELT07] lists the following as the principal elements of an identity management system.

- **Authentication:** Confirmation that a user corresponds to the user name provided.
- **Authorization:** Granting access to specific services and/or resources based on the authentication.
- **Accounting:** A process for logging access and authorization.
- **Provisioning:** The enrollment of users in the system.
- **Workflow automation:** Movement of data in a business process.
- **Delegated administration:** The use of role-based access control to grant permissions.
- **Password synchronization:** Creating a process for single sign-on (SSO) or reduced sign-on (RSO). Single sign-on enables a user to access all network resources after a single authentication. RSO may involve multiple sign-ons but requires less user effort than if each resource and service maintained its own authentication facility.
- **Self-service password reset:** Enables the user to modify his or her password.
- **Federation:** A process where authentication and permission will be passed on from one system to another—usually across multiple enterprises, thereby reducing the number of authentications needed by the user.

Note that Kerberos contains a number of the elements of an identity management system.

Figure 4.8 [LINN06] illustrates entities and data flows in a generic identity management architecture. A **principal** is an identity holder. Typically, this is a human user that seeks access to resources and services on the network. User devices, agent processes, and server systems may also function as principals. Principals authenticate themselves to an **identity provider**. The identity provider associates authentication information with a principal, as well as attributes and one or more identifiers.

Increasingly, digital identities incorporate attributes other than simply an identifier and authentication information (such as passwords and biometric information). An **attribute service** manages the creation and maintenance of such attributes. For example, a user needs to provide a shipping address each time an order is placed at a new Web merchant, and this information needs to be revised when the user moves. Identity management enables the user to provide this information once, so that it is maintained in a single place and released to data consumers in accordance with authorization and privacy policies. Users may create some of the attributes to be associated with their digital identity, such as address. **Administrators** may also assign attributes to users, such as roles, access permissions, and employee information.

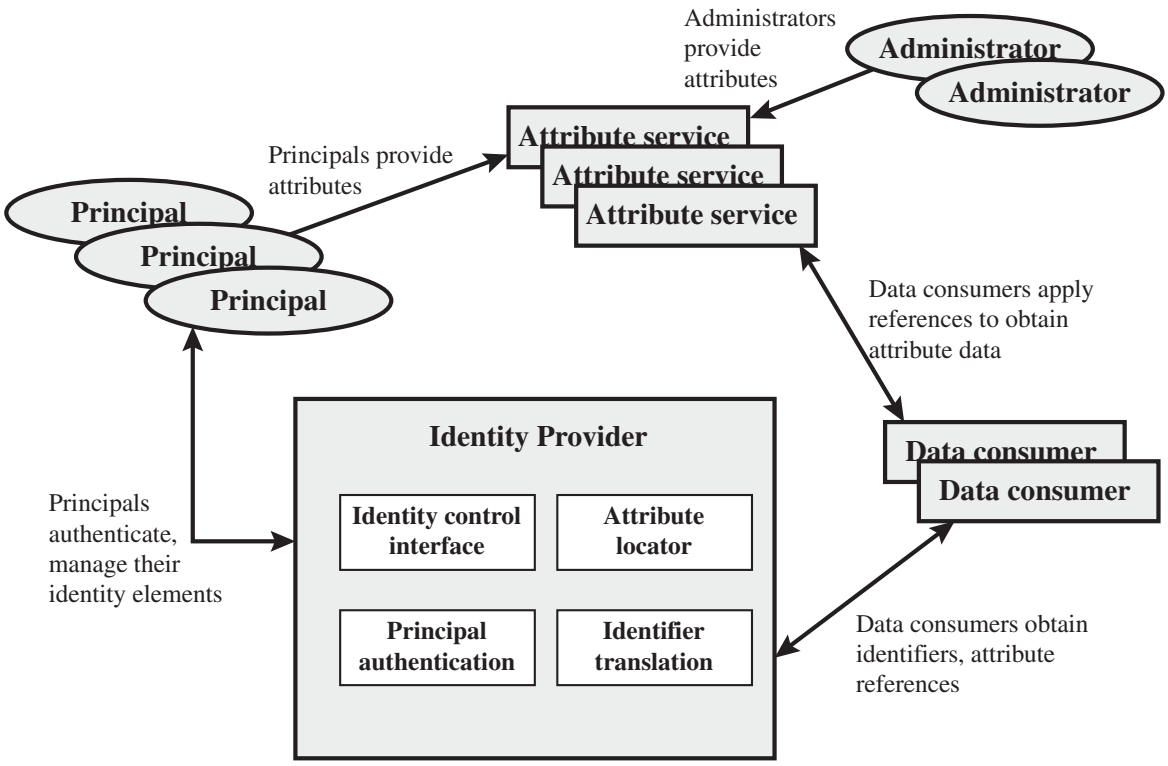


Figure 4.8 Generic Identity Management Architecture

Data consumers are entities that obtain and employ data maintained and provided by identity and attribute providers, which are often used to support authorization decisions and to collect audit information. For example, a database server or file server is a data consumer that needs a client's credentials so as to know what access to provide to that client.

Identity Federation

Identity federation is, in essence, an extension of identity management to multiple security domains. Such domains include autonomous internal business units, external business partners, and other third-party applications and services. The goal is to provide the sharing of digital identities so that a user can be authenticated a single time and then access applications and resources across multiple domains. Because these domains are relatively autonomous or independent, no centralized control is possible. Rather, the cooperating organizations must form a federation based on agreed standards and mutual levels of trust to securely share digital identities.

Federated identity management refers to the agreements, standards, and technologies that enable the portability of identities, identity attributes, and entitlements across multiple enterprises and numerous applications and supporting many thousands, even millions, of users. When multiple organizations implement interoperable federated identity schemes, an employee in one organization can use a single sign-on to access services across the federation with trust relationships associated with the identity. For example, an employee may log onto her corporate intranet and be authenticated to perform authorized functions and access authorized services on

that intranet. The employee could then access their health benefits from an outside health-care provider without having to reauthenticate.

Beyond SSO, federated identity management provides other capabilities. One is a standardized means of representing attributes. Increasingly, digital identities incorporate attributes other than simply an identifier and authentication information (such as passwords and biometric information). Examples of attributes include account numbers, organizational roles, physical location, and file ownership. A user may have multiple identifiers; for example, each identifier may be associated with a unique role with its own access permissions.

Another key function of federated identity management is identity mapping. Different security domains may represent identities and attributes differently. Furthermore, the amount of information associated with an individual in one domain may be more than is necessary in another domain. The federated identity management protocols map identities and attributes of a user in one domain to the requirements of another domain.

Figure 4.9 illustrates entities and data flows in a generic federated identity management architecture.

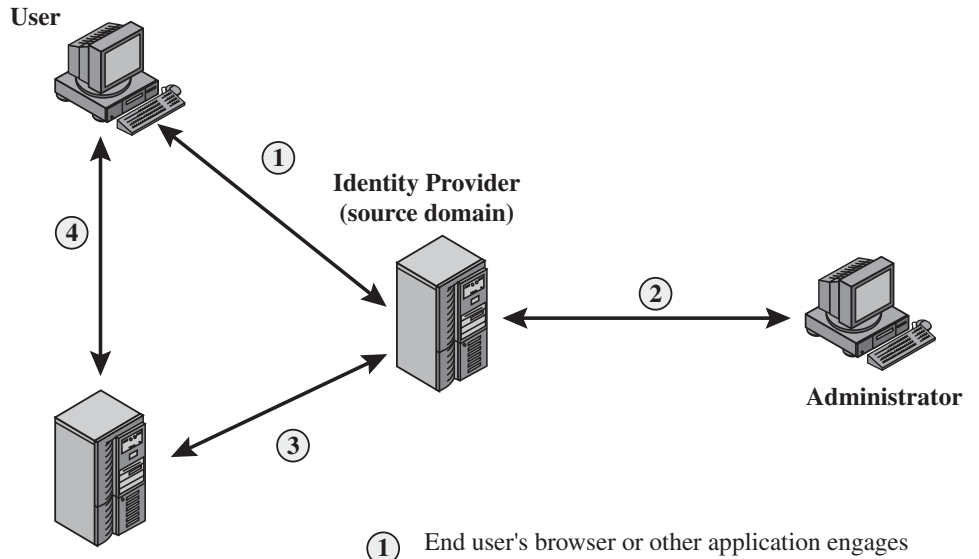


Figure 4.9 Federated Identity Operation

The identity provider acquires attribute information through dialogue and protocol exchanges with users and administrators. For example, a user needs to provide a shipping address each time an order is placed at a new Web merchant, and this information needs to be revised when the user moves. Identity management enables the user to provide this information once, so that it is maintained in a single place and released to data consumers in accordance with authorization and privacy policies.

Service providers are entities that obtain and employ data maintained and provided by identity providers, often to support authorization decisions and to collect audit information. For example, a database server or file server is a data consumer that needs a client's credentials so as to know what access to provide to that client. A service provider can be in the same domain as the user and the identity provider. The power of this approach is for federated identity management, in which the service provider is in a different domain (e.g., a vendor or supplier network).

Standards Federated identity management uses a number of standards as the building blocks for secure identity exchange across different domains or heterogeneous systems. In essence, organizations issue some form of security tickets for their users that can be processed by cooperating partners. Identity federation standards are thus concerned with defining these tickets in terms of content and format, providing protocols for exchanging tickets, and performing a number of management tasks. These tasks include configuring systems to perform attribute transfers and identity mapping and performing logging and auditing functions.

The principal underlying standard for federated identity is the Security Assertion Markup Language (SAML), which defines the exchange of security information between online business partners. SAML conveys authentication information in the form of assertions about subjects. Assertions are statements about the subject issued by an authoritative entity.

SAML is part of a broader collection of standards being issued by the Organization for the Advancement of Structured Information Standards (OASIS) for federated identity management. For example, WS-Federation enables browser-based federation; it relies on a security token service to broker trust of identities, attributes, and authentication between participating Web services.

The challenge with federated identity management is to integrate multiple technologies, standards, and services to provide a secure, user-friendly utility. The key, as in most areas of security and networking, is the reliance on a few mature standards widely accepted by industry. Federated identity management seems to have reached this level of maturity.

Examples To get some feel for the functionality of identity federation, we look at three scenarios, taken from [COMP06]. In the first scenario (Figure 4.10a), Workplace.com contracts with Health.com to provide employee health benefits. An employee uses a Web interface to sign on to Workplace.com and goes through an authentication procedure there. This enables the employee to access authorized services and resources at Workplace.com. When the employee clicks

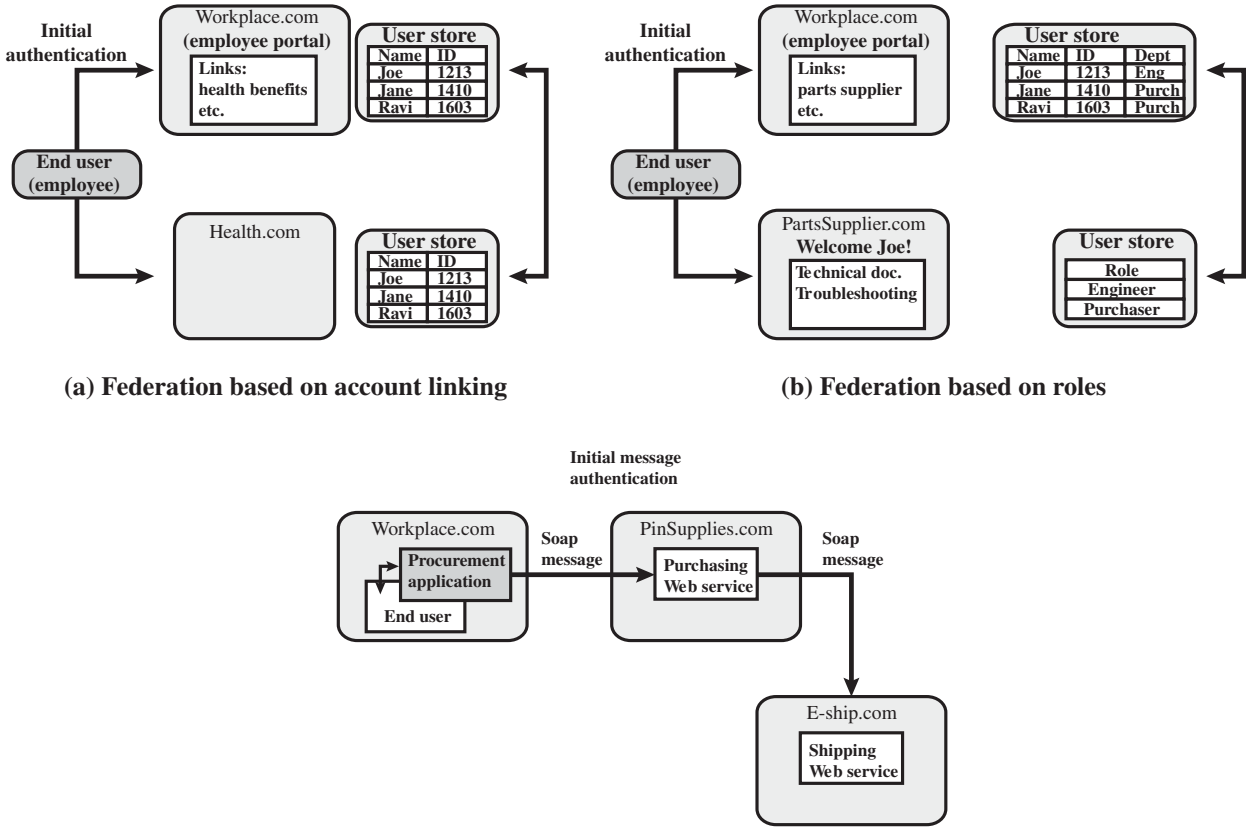


Figure 4.10 Federated Identity Scenarios

on a link to access health benefits, her browser is redirected to Health.com. At the same time, the Workplace.com software passes the user's identifier to Health.com in a secure manner. The two organizations are part of a federation that cooperatively exchanges user identifiers. Health.com maintains user identities for every employee at Workplace.com and associates with each identity health-benefits information and access rights. In this example, the linkage between the two companies is based on account information and user participation is browser based.

Figure 4.10b shows a second type of browser-based scheme. PartsSupplier.com is a regular supplier of parts to Workplace.com. In this case, a role-based access-control (RBAC) scheme is used for access to information. An engineer of Workplace.com authenticates at the employee portal at Workplace.com and clicks on a link to access information at PartsSupplier.com. Because the user is authenticated in the role of an engineer, he is taken to the technical documentation and troubleshooting portion of PartSupplier.com's Web site without having to sign on. Similarly, an employee in a purchasing role signs on at Workplace.com and is authorized, in that role, to place purchases at PartSupplier.com without having to authenticate to PartSupplier.com. For this scenario, PartSupplier.com does not have identity information for individual

employees at Workplace.com. Rather, the linkage between the two federated partners is in terms of roles.

The scenario illustrated in Figure 4.10c can be referred to as document based rather than browser based. In this third example, Workplace.com has a purchasing agreement with PinSupplies.com, and PinSupplies.com has a business relationship with E-Ship.com. An employee of Workplace.com signs on and is authenticated to make purchases. The employee goes to a procurement application that provides a list of Workplace.com's suppliers and the parts that can be ordered. The user clicks on the PinSupplies button and is presented with a purchase order Web page (HTML page). The employee fills out the form and clicks the submit button. The procurement application generates an XML/SOAP document that it inserts into the envelope body of an XML-based message. The procurement application then inserts the user's credentials in the envelope header of the message, together with Workplace.com's organizational identity. The procurement application posts the message to the PinSupplies.com's purchasing Web service. This service authenticates the incoming message and processes the request. The purchasing Web service then sends a SOAP message its shipping partner to fulfill the order. The message includes a PinSupplies.com security token in the envelope header and the list of items to be shipped as well as the end user's shipping information in the envelope body. The shipping Web service authenticates the request and processes the shipment order.

4.7 RECOMMENDED READING AND WEB SITES

An exhaustive and essential resource on the topics of this chapter is the three-volume NIST SP800-57 [BARK07b. BARK07c, BARK08]. [FUMY93] is a good survey of key management principles. Another interesting survey, which looks at many key management techniques, is [HEGL06].

A painless way to get a grasp of Kerberos concepts is found in [BRYA88]. One of the best treatments of Kerberos is [KOHL94].

[PERL99] reviews various trust models that can be used in a PKI. [GUTM02] highlights difficulties in PKI use and recommends approaches for an effective PKI.

[SHIM05] provides a brief overview of federated identity management and examines one approach to standardization. [BHAT07] describes an integrated approach to federated identity management couple with management of access control privileges.

BARK07b Barker, E., et al. *Recommendation for Key Management—Part 1: General.* NIST SP800-57, March 2007.

BARK07c Barker, E., et al. *Recommendation for Key Management—Part 2: Best Practices for Key Management Organization.* NIST SP800-57, March 2007.

BARK08 Barker, E., et al. *Recommendation for Key Management—Part* 3: *Specific Key Management Guidance.* NIST SP800-57, August 2008.

BHAT07 Bhatti, R.; Bertino, E.; and Ghafoor, A. "An Integrated Approach to Federated Identity and Privilege Management in Open Systems." *Communications of the ACM*, February 2007.

BRYA88 Bryant, W. *Designing an Authentication System: A Dialogue in Four Scenes.* Project Athena document, February 1988. Available at http://web.mit.edu/kerberos/www/dialogue.html.

FUMY93 Fumy, S., and Landrock, P. "Principles of Key Management." *IEEE Journal on Selected Areas in Communications*, June 1993.

GUTM02 Gutmann, P. "PKI: It's Not Dead, Just Resting." *Computer*, August 2002.

HEGL06 Hegland, A., et al. "A Survey of Key Management in Ad Hoc Networks." *IEEE Communications Surveys & Tutorials.* 3rd Quarter, 2006.

KOHL94 Kohl, J.; Neuman, B.; and Ts'o, T. "The Evolution of the Kerberos Authentication Service." in Brazier, F., and Johansen, D. *Distributed Open Systems.* Los Alamitos, CA: IEEE Computer Society Press, 1994. Available at http://web.mit.edu/kerberos/www/papers.html.

PERL99 Perlman, R. "An Overview of PKI Trust Models." *IEEE Network*, November/December 1999.

SHIM05 Shim, S.; Bhalla, G.; and Pendyala, V. "Federated Identity Management." *Computer*, December 2005.

Recommended Web Sites:

- **MIT Kerberos Site:** Information about Kerberos, including the FAQ, papers and documents, and pointers to commercial product sites.
- **MIT Kerberos Consortium:** Created to establish Kerberos as the universal authentication platform for the world's computer networks.
- **USC/ISI Kerberos Page:** Another good source of Kerberos material.
- **Kerberos Working Group:** IETF group developing standards based on Kerberos.
- **Public-Key Infrastructure Working Group:** IETF group developing standards based on X.509v3.
- **Verisign:** A leading commercial vendor of X.509-related products; white papers and other worthwhile material at this site.
- **NIST PKI Program:** Good source of information.

4.8 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

authentication authentication server (AS) federated identity management identity management	Kerberos Kerberos realm key distribution key distribution center (KDC)	key management master key mutual authentication nonce one-way authentication

propagating cipher block chaining (PCBC) mode	realm	ticket-granting server (TGS)
public-key certificate	replay attack	timestamp
public-key directory	ticket	X.509 certificate

Review Questions

4.1 List ways in which secret keys can be distributed to two communicating parties.
4.2 What is the difference between a session key and a master key?
4.3 What is a key distribution center?
4.4 What entities constitute a full-service Kerberos environment?
4.5 In the context of Kerberos, what is a realm?
4.6 What are the principal differences between version 4 and version 5 of Kerberos?
4.7 What is a nonce?
4.8 What are two different uses of public-key cryptography related to key distribution?
4.9 What are the essential ingredients of a public-key directory?
4.10 What is a public-key certificate?
4.11 What are the requirements for the use of a public-key certificate scheme?
4.12 What is the purpose of the X.509 standard?
4.13 What is a chain of certificates?
4.14 How is an X.509 certificate revoked?

Problems

4.1 "We are under great pressure, Holmes." Detective Lestrade looked nervous. "We have learned that copies of sensitive government documents are stored in computers of one foreign embassy here in London. Normally these documents exist in electronic form only on a selected few government computers that satisfy the most stringent security requirements. However, sometimes they must be sent through the network connecting all government computers. But all messages in this network are encrypted using a top secret encryption algorithm certified by our best crypto experts. Even the NSA and the KGB are unable to break it. And now these documents have appeared in hands of diplomats of a small, otherwise insignificant, country. And we have no idea how it could happen."

"But you do have some suspicion who did it, do you?" asked Holmes.

"Yes, we did some routine investigation. There is a man who has legal access to one of the government computers and has frequent contacts with diplomats from the embassy. But the computer he has access to is not one of the trusted ones where these documents are normally stored. He is the suspect, but we have no idea how he could obtain copies of the documents. Even if he could obtain a copy of an encrypted document, he couldn't decrypt it."

"Hmm, please describe the communication protocol used on the network." Holmes opened his eyes, thus proving that he had followed Lestrade's talk with an attention that contrasted with his sleepy look.

"Well, the protocol is as follows. Each node N of the network has been assigned a unique secret key K_n. This key is used to secure communication between the node and a trusted server. That is, all the keys are stored also on the server. User A, wishing to send a secret message M to user B, initiates the following protocol:

1. A generates a random number R and sends to the server his name A, destination B, and E(K_a, R).

2. Server responds by sending $E(K_b, R)$ to A.
3. A sends $E(R, M)$ together with $E(K_b, R)$ to B.
4. B knows K_b, thus decrypts $E(K_b, R)$ to get R and will subsequently use R to decrypt $E(R, M)$ to get M.

You see that a random key is generated every time a message has to be sent. I admit the man could intercept messages sent between the top secret trusted nodes, but I see no way he could decrypt them."

"Well, I think you have your man, Lestrade. The protocol isn't secure because the server doesn't authenticate users who send him a request. Apparently designers of the protocol have believed that sending $E(K_x, R)$ implicitly authenticates user X as the sender, as only X (and the server) knows K_x. But you know that $E(K_x, R)$ can be intercepted and later replayed. Once you understand where the hole is, you will be able to obtain enough evidence by monitoring the man's use of the computer he has access to. Most likely he works as follows: After intercepting $E(K_a, R)$ and $E(R, M)$ (see steps 1 and 3 of the protocol), the man, let's denote him as Z, will continue by pretending to be A and…

Finish the sentence for Holmes.

4.2 There are three typical ways to use nonces as challenges. Suppose N_a is a nonce generated by A, A and B share key K, and f() is a function (such as increment). The three usages are

Usage 1	**Usage 2**	**Usage 3**
(1) A → B: N_a (2) B → A: $E(K, N_a)$	(1) A → B: $E(K, N_a)$ (2) B → A: N_a	(1) A → B: $E(K, N_a)$ (2) B → A: $E(K, f(N_a))$

Describe situations for which each usage is appropriate.

4.3 Show that a random error in one block of ciphertext is propagated to all subsequent blocks of plaintext in PCBC mode (see Figure F.2 in Appendix F).

4.4 Suppose that, in PCBC mode, blocks C_i and C_{i+1} are interchanged during transmission. Show that this affects only the decrypted blocks P_i and P_{i+1} but not subsequent blocks.

4.5 In addition to providing a standard for public-key certificate formats, X.509 specifies an authentication protocol. The original version of X.509 contains a security flaw. The essence of the protocol is

$$\begin{aligned} A \rightarrow B: &\quad A\ \{t_A, r_A, ID_B\} \\ B \rightarrow A: &\quad B\ \{t_B, r_B, ID_A, r_A\} \\ A \rightarrow B: &\quad A\ \{r_B\} \end{aligned}$$

where t_A and t_B are timestamps, r_A and r_B are nonces, and the notation X {Y} indicates that the message Y is transmitted, encrypted, and signed by X.

The text of X.509 states that checking timestamps t_A and t_B is optional for three-way authentication. But consider the following example: Suppose A and B have used the preceding protocol on some previous occasion, and that opponent C has intercepted the preceding three messages. In addition, suppose that timestamps are not used and are all set to 0. Finally, suppose C wishes to impersonate A to B. C initially sends the first captured message to B:

$$C \rightarrow B: \quad A\ \{0, r_A, ID_B\}$$

B responds, thinking it is talking to A but is actually talking to C:

$$B \rightarrow C: \quad B\{0, r'_B, ID_A, r_A\}$$

C meanwhile causes A to initiate authentication with C by some means. As a result, A sends C the following:

$$A \rightarrow C: \quad A\ \{0, r'_A, ID_C\}$$

C responds to A using the same nonce provided to C by B.

$$C \rightarrow A: \quad C\ \{0, r'_B, ID_A, r'_A\}$$

A responds with

$$A \rightarrow C: \quad A\ \{r'_B\}$$

This is exactly what C needs to convince B that it is talking to A, so C now repeats the incoming message back out to B.

$$C \rightarrow B: \quad A\ \{r'_B\}$$

So B will believe it is talking to A, whereas it is actually talking to C. Suggest a simple solution to this problem that does not involve the use of timestamps.

4.6 Consider a one-way authentication technique based on asymmetric encryption:

$$A \rightarrow B: \quad ID_A$$

$$B \rightarrow A: \quad R_1$$

$$A \rightarrow B: \quad E(PR_a, R_1)$$

a. Explain the protocol.
b. What type of attack is this protocol susceptible to?

4.7 Consider a one-way authentication technique based on asymmetric encryption:

$$A \rightarrow B: \quad ID_A$$

$$B \rightarrow A: \quad E(PU_a, R_2)$$

$$A \rightarrow B: \quad R_2$$

a. Explain the protocol.
b. What type of attack is this protocol susceptible to?

4.8 In Kerberos, when Bob receives a ticket from Alice, how does he know it is genuine?

4.9 In Kerberos, when Bob receives a ticket from Alice, how does he know it came from Alice?

4.10 In Kerberos, Alice receives a reply, how does she know it came from Bob (that it's not a replay of an earlier message from Bob)?

4.11 In Kerberos, what does the ticket contain that allows Alice and Bob to talk securely?

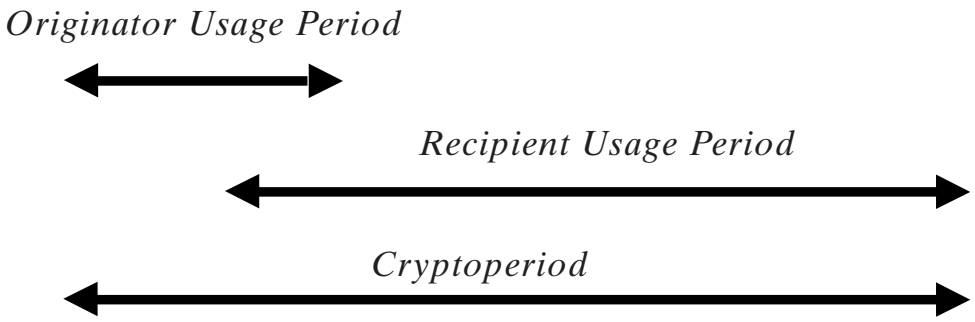
4.12 The 1988 version of X.509 lists properties that RSA keys must satisfy to be secure, given current knowledge about the difficulty of factoring large numbers. The discussion concludes with a constraint on the public exponent and the modulus n:

> It must be ensured that $e > \log_2(n)$ to prevent attack by taking the eth root mod n to disclose the plaintext.

Although the constraint is correct, the reason given for requiring it is incorrect. What is wrong with the reason given and what is the correct reason?

4.13 Find at least one intermediate certification authority's certificate and one trusted root certification authority's certificate on your computer (e.g. in the browser). Print screenshots of both the general and details tab for each certificate.

4.14 NIST defines the term cryptoperiod as the time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect. One document on key management uses the following time diagram for a shared secret key.



Explain the overlap by giving an example application in which the originator's usage period for the shared secret key begins before the recipient's usage period and also ends before the recipient's usage period.

4.15 Consider the following protocol, designed to let A and B decide on a fresh, shared session key K'_{AB}. We assume that they already share a long-term key K_{AB}.

1. $A \rightarrow B$: A, N_A
2. $B \rightarrow A$: $\mathrm{E}(K_{AB}, [N_A, K'_{AB}])$
3. $A \rightarrow B$: $\mathrm{E}(K'_{AB}, N_A)$

a. We first try to understand the protocol designer's reasoning:

- Why would A and B believe after the protocol ran that they share K'_{AB} with the other party?
- Why would they believe that this shared key is fresh?

In both cases, you should explain both the reasons of both A and B, so your answer should complete the following sentences.

A believes that she shares K'_{AB} with B since . . .
B believes that he shares K'_{AB} with A since . . .
A believes that K'_{AB} is fresh since . . .
B believes that K'_{AB} is fresh since . . .

b. Assume now that A starts a run of this protocol with B. However, the connection is intercepted by the adversary C. Show how C can start a new run of the protocol using reflection, causing A to believe that she has agreed on a fresh key with B (in spite of the fact that she has only been communicating with C). Thus, in particular, the belief in (a) is false.

c. Propose a modification of the protocol that prevents this attack.

4.16 What are the core components of a PKI? Briefly describe each component.

4.17 Explain the problems with key management and how it affects symmetric cryptography.

4.18 Consider the following protocol:

$$\begin{aligned}
&\mathrm{A} \rightarrow \mathrm{KDC}: && ID_A \| ID_B \| N_1 \\
&\mathrm{KDC} \rightarrow \mathrm{A}: && \mathrm{E}(K_a, [K_S \| ID_B \| N_1 \| \mathrm{E}(K_b, [K_S \| ID_A])) \\
&\mathrm{A} \rightarrow \mathrm{B}: && \mathrm{E}(K_b, [K_S \| ID_A]) \\
&\mathrm{B} \rightarrow \mathrm{A}: && \mathrm{E}(K_S, N_2) \\
&\mathrm{A} \rightarrow \mathrm{B}: && \mathrm{E}(K_S, \mathrm{f}(N_2))
\end{aligned}$$

a. Explain the protocol.

b. Can you think of a possible attack on this protocol? Explain how it can be done.

c. Mention a possible technique to get around the attack—not a detailed mechanism, just the basics of the idea.

Note: The remaining problems deal with a cryptographic product developed by IBM, which is briefly described in a document at this book's Web site in `IBMCrypto.pdf`*. Try these problems after reviewing the document.*

4.19 What is the effect of adding the instruction EMK_i?

$$EMK_i: X \rightarrow E(KMH_i, X) \quad i = 0, 1$$

4.20 Suppose *N* different systems use the IBM Cryptographic Subsystem with host master keys KMH[*i*] (i = 1, 2, . . . , *N*). Devise a method for communicating between systems without requiring the system to either share a common host master key or to divulge their individual host master keys. *Hint*: Each system needs three variants of its host master key.

4.21 The principal objective of the IBM Cryptographic Subsystem is to protect transmissions between a terminal and the processing system. Devise a procedure, perhaps adding instructions, which will allow the processor to generate a session key KS and distribute it to Terminal i and Terminal j without having to store a key-equivalent variable in the host.