PART FIVE: MUTUAL TRUST

CHAPTER 15

CRYPTOGRAPHIC KEY MANAGEMENT AND DISTRIBUTION

LEARNING OBJECTIVES

After studying this chapter, you should be able to:

- Discuss the concept of a key hierarchy.
- Understand the issues involved in using asymmetric encryption to distribute symmetric keys.
- Present an overview of approaches to public-key distribution and analyze the risks involved in various approaches.
- List and explain the elements in an X.509 certificate.
- Present an overview of public-key infrastructure concepts.

The secure use of cryptographic key algorithms depends on the protection of the cryptographic keys. All keys need to be protected against modification, and secret and private keys need to be protected against disclosure. **Cryptographic key management** is the process of administering or managing cryptographic keys for a cryptographic system. It involves the generation, creation, protection, storage, exchange, replacement, and use of keys and enables selective restriction for certain keys. In addition to access restriction, key management also involves the monitoring and recording of each key's access, use, and context. A key management system will also include key servers, user procedures, and protocols, including cryptographic protocol design. The security of the cryptosystem is dependent upon successful key management.

The topics of cryptographic key management and cryptographic key distribution are complex, involving cryptographic, protocol, and management considerations. The purpose of this chapter is to give the reader a feel for the issues involved and a broad survey of the various aspects of key management and distribution. For more information, the place to start is the three-volume NIST SP 800-57, followed by the recommended readings listed at the end of this chapter.

15.1 SYMMETRIC KEY DISTRIBUTION USING SYMMETRIC ENCRYPTION

This section looks at techniques for distributing secret keys using only symmetric encryption techniques.

Key Distribution Options

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 who wish to exchange data, without allowing others to see the key. For two parties A and B, **key distribution** can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can 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 can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can 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 going to be exchanging data only 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 user or server may need to engage in exchanges with many other users and servers over time. Thus, each endpoint needs a number of keys supplied dynamically. The problem is especially difficult in a wide area distributed system.

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are n hosts, the number of required keys is

$$\frac{n(n-1)}{2}$$

If encryption is done at the application level, then a key is needed for every pair of users or processes that require communication. Thus, a network may have hundreds of hosts but thousands of users and processes. A network using node-level encryption with 1000 nodes would conceivably need to distribute as many as half a million keys. If that same network supported 10,000 applications, then as many as 50 million keys may be required for application-level encryption.

Returning to our list, 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 will be revealed. Furthermore, the initial distribution of potentially millions of keys must still be made.

For end-to-end encryption, some variation on option 4 has been widely adopted. In this scheme, a key distribution center is responsible for distributing keys to pairs of users (hosts, processes, applications) as needed. Each user must share a unique key with the key distribution center for purposes of key distribution.

Third-Party Key Distribution Options

Figure 15.1 illustrates two different options, each with two variations, for key distribution. The numbers along the lines represent the steps of the exchange. In these examples, there exists a connection between entities A and B, who wish to exchange information using cryptographic techniques. For this purpose, they require a

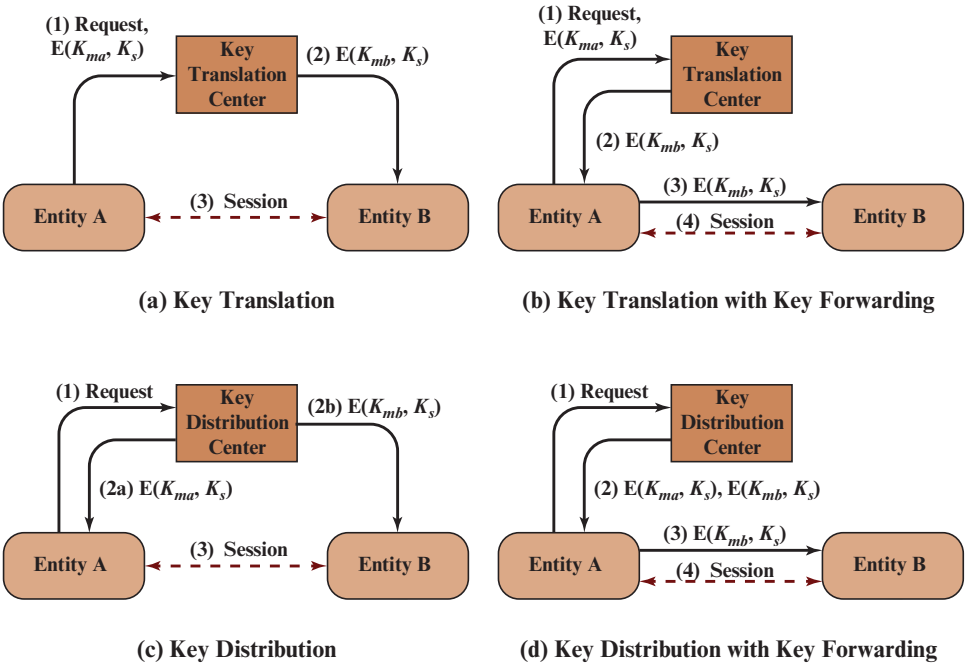


Figure 15.1 Key Distribution Between Two Communicating Entities

temporary **session key** that will last for the duration of a logical connection, such as a TCP connection. A and B each share a long-lasting **master key** with a third party that is involved in providing the session key. For this discussion, the session key is labeled K_s and the master key between entities A and B and the third party are labeled K_{ma} and K_{mb}, respectively.

A key translation center (KTC) transfers symmetric keys for future communication between two entities, at least one of whom has the ability to generate or acquire symmetric keys by themselves. Entity A generates or acquires a symmetric key to be used as a session key for communication with B. A encrypts the key using the master key it shares with the KTC and sends the encrypted key to the KTC. The KTC decrypts the session key, reencrypts the session key in the master key it shares with B, and either sends that reencrypted session key to A (Figure 15.1a) for A to forward to B or sends it directly to B (Figure 15.1b).

A **key distribution center (KDC)** generates and distributes session keys. Entity A sends a request to the KDC for a symmetric key to be used as a session key for communication with B. The KDC generates a symmetric session key, and then encrypts the session key with the master key it shares with A and sends it to A. The KDC also encrypts the session key with the master key is shares with B and sends it to B (Figure 15.1c). Alternatively, it sends both encrypted key values to A, and A forwards the session key encrypted with the master key shared by the KDC and B to B (Figure 15.1d).

The foregoing discussion leaves out a number of details. For example, parties that exchange keys need to authenticate themselves to each other.

Timestamps are often used to limit the time in which a key exchange can take place and/or the lifetime of an exchanged key. Chapter 16 examines several detailed approaches to third-party symmetric key exchange in the context of Kerberos.

Key Hierarchy

A common requirement in a variety of protocols, such as IEEE 802.11i and IPsec, discussed in Part Six, is for the encryption of a symmetric key so that it can be distributed to two parties for future communication. Quite often, a protocol calls for a hierarchy of keys, with keys lower on the hierarchy used more frequently, and changed more frequently to thwart attacks (Figure 15.2). A higher-level key, which is used infrequently and therefore more resistant to cryptanalysis, is used to encrypt a newly created lower-level key so that it can be exchanged between parties that share the higher-level key. The term **ephemeral key** in Figure 15.2 refers to a key that is used only once or at most is very short-lived.

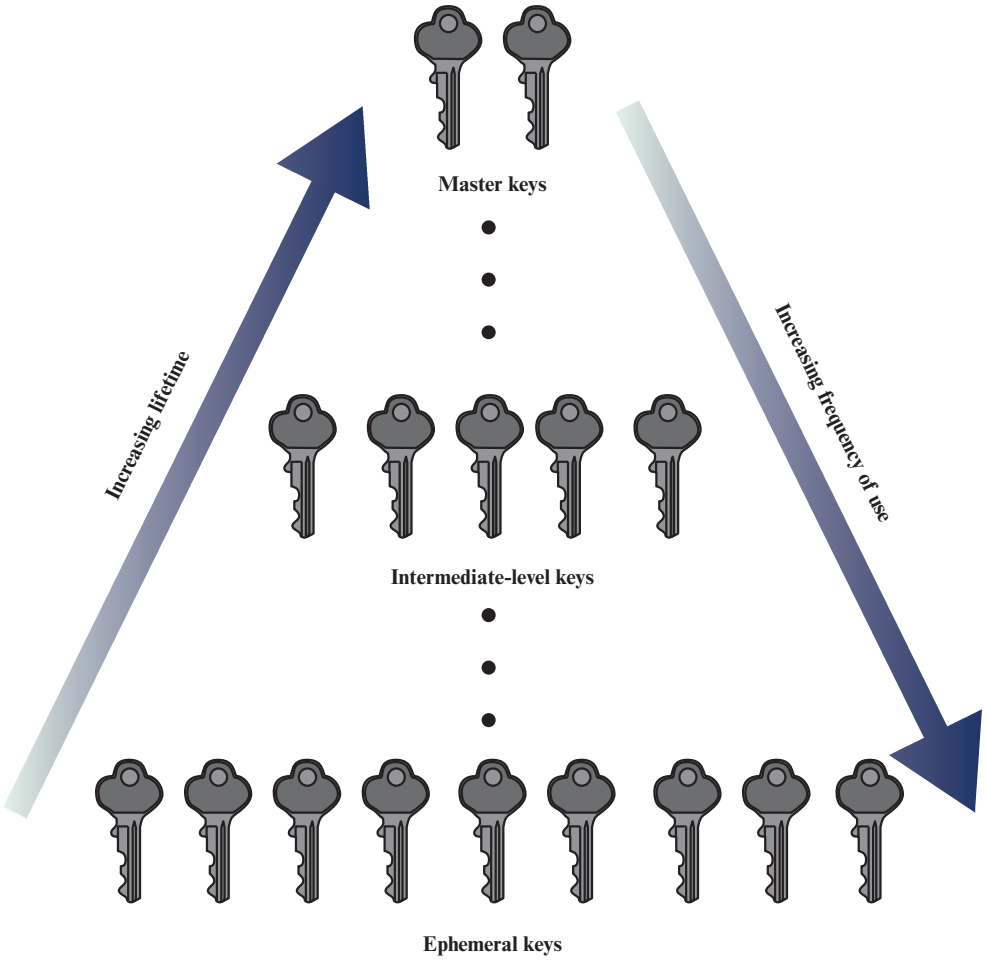


Figure 15.2 Symmetric Key Hierarchy

15.2 SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

Because of the inefficiency of public-key cryptosystems, they are almost never used for the direct encryption of sizable blocks of data, but are limited to relatively small blocks. One of the most important uses of a public-key cryptosystem is to encrypt secret keys for distribution. We see many specific examples of this in Part Five. Here, we discuss general principles and typical approaches.

Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [MERK79], as illustrated in Figure 15.3. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s. Despite its simplicity, this is an attractive protocol. No keys exist before the start of the communication and none exist after the completion of communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping.

The protocol depicted in Figure 15.3 is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message (see Figure 1.3c). Such an attack is known as a **man-in-the-middle attack** [RIVE84]. We saw this type of attack in Chapter 10 (Figure 10.2). In the present case, if an adversary, D, has control of the intervening communication channel, then D can compromise the communication in the following fashion without being detected (Figure 15.4).

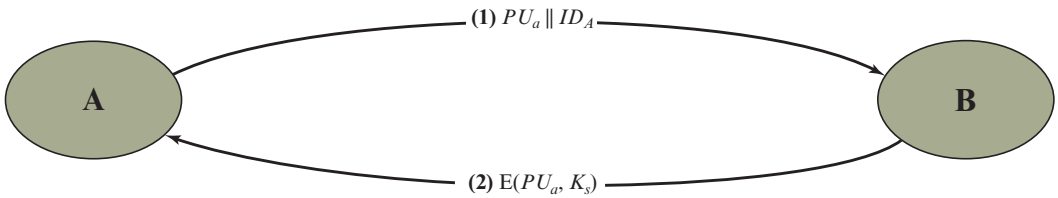


Figure 15.3 Simple Use of Public-Key Encryption to Establish a Session Key

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. D intercepts the message, creates its own public/private key pair $\{PU_d, PR_d\}$ and transmits $PU_d \| ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_d, K_s)$.
4. D intercepts the message and learns K_s by computing $D(PR_d, E(PU_d, K_s))$.
5. D transmits $E(PU_a, K_s)$ to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to D. A and B can now exchange messages using K_s. D no longer actively

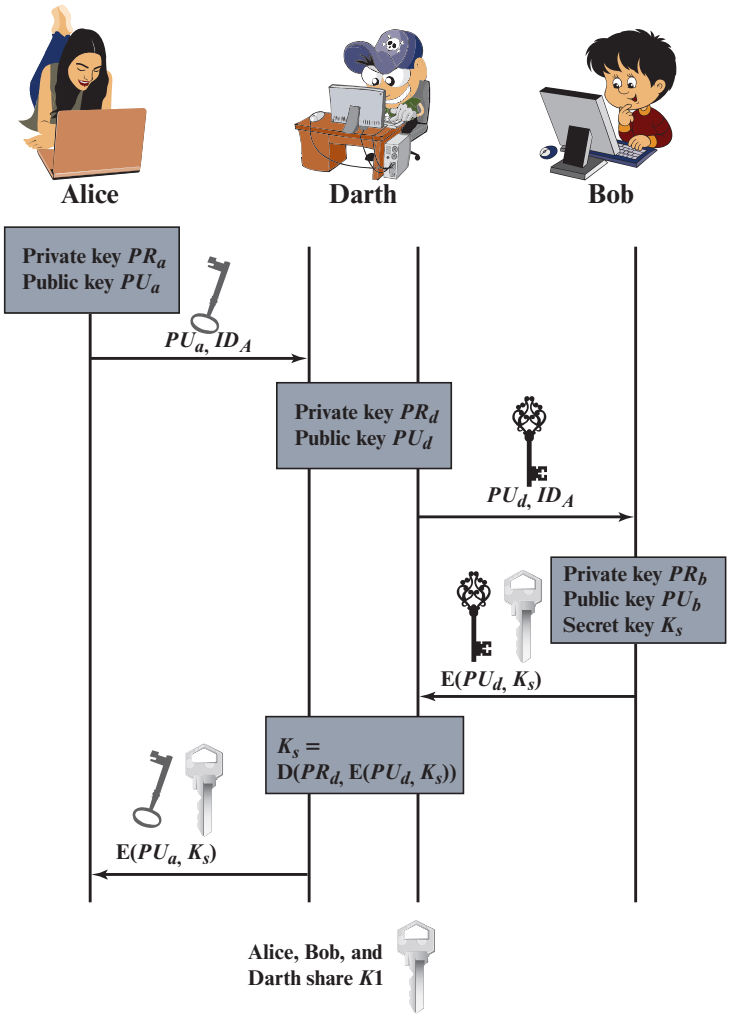


Figure 15.4 Another Man-in-the-Middle Attack

interferes with the communications channel but simply eavesdrops. Knowing K_s, D can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

Figure 15.5, based on an approach suggested in [NEED78], provides protection against both active and passive attacks. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described subsequently in this chapter. Then the following steps occur.

1. A uses B's public key to encrypt a message to B containing an identifier of A(ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2, encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = \text{E}(PU_b, \text{E}(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $\text{D}(PU_a, \text{D}(PR_b, M))$ to recover the secret key.

The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

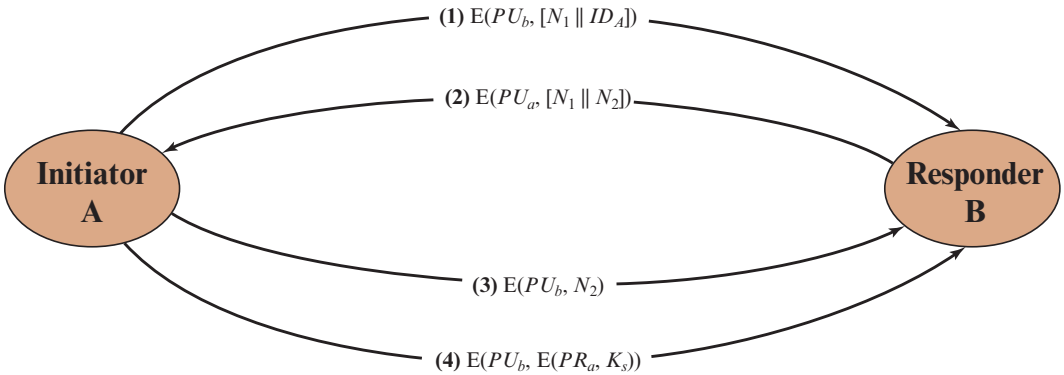


Figure 15.5 Public-Key Distribution of Secret Keys

15.3 DISTRIBUTION OF PUBLIC KEYS

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Public Announcement of Public Keys

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 (Figure 15.6).

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 (see Figure 9.3).

Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure 15.7). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.

Figure 15.6 Uncontrolled Public-Key Distribution

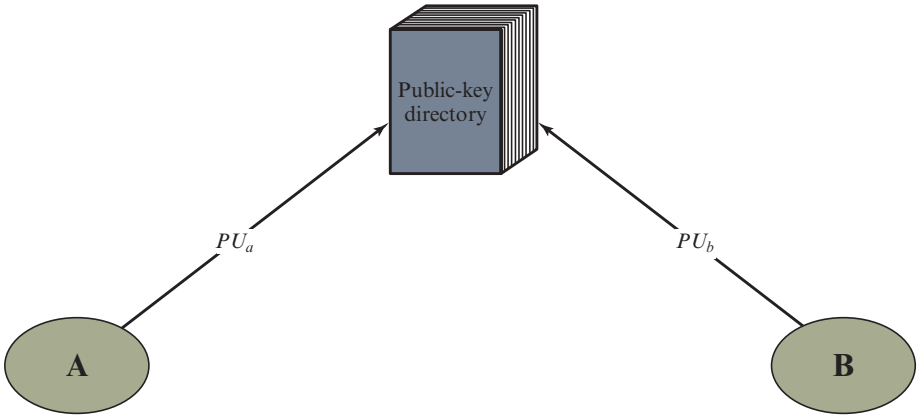


Figure 15.7 Public-Key Publication

3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure 15.8, which is based on a figure in [POPE79]. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps (matched by number to Figure 15.8) occur.

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.
2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth}. Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:

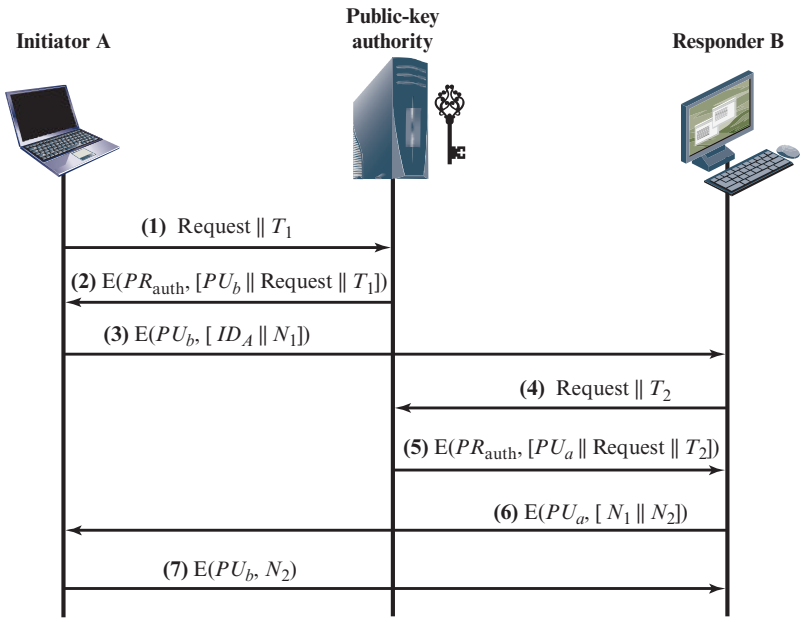


Figure 15.8 Public-Key Distribution Scenario

- B's public key, PU_b, which A can use to encrypt messages destined for B
- The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
- The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key

3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.

7. A returns N_2, which is encrypted using B's public key, to assure B that its correspondent is A.

Thus, a total of seven messages are required. However, the initial five messages need be used only infrequently because both A and B can save the other's public key for future use—a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

Public-Key Certificates

The scenario of Figure 15.8 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOHN78], is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. 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. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.

These requirements are satisfied by the original proposal in [KOHN78]. Denning [DENN83] added the following additional requirement:

4. Any participant can verify the time validity of the certificate.

A certificate scheme is illustrated in Figure 15.9. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

$$C_A = \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])$$

where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

$$\mathrm{D}(PU_{\mathrm{auth}}, C_A) = \mathrm{D}(PU_{\mathrm{auth}}, \mathrm{E}(PR_{\mathrm{auth}}, [T \| ID_A \| PU_a])) = (T \| ID_A \| PU_a)$$

The recipient uses the authority's public key, PU_{auth}, to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp counters the following scenario. A's private key is learned by an adversary. A generates a new

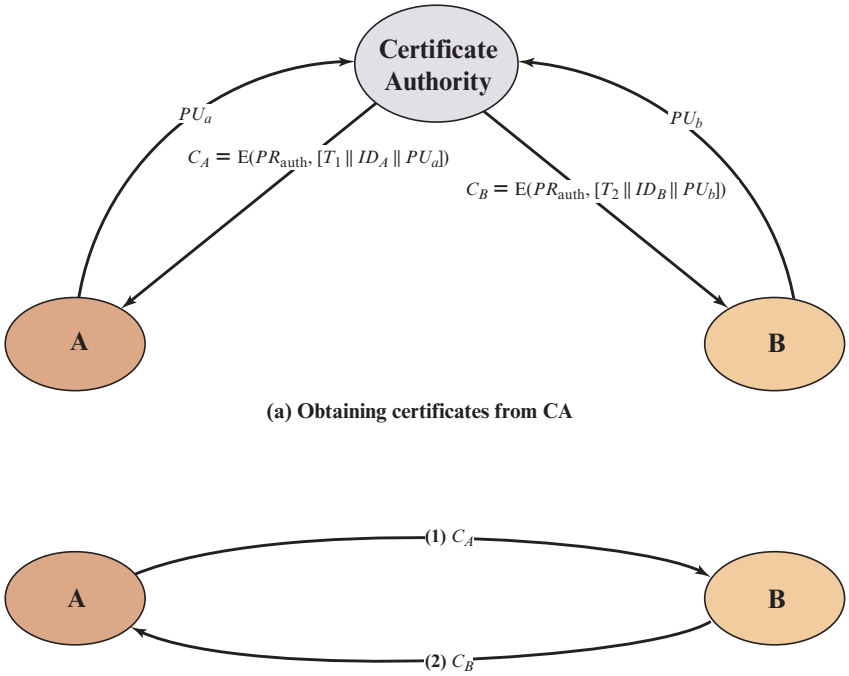


Figure 15.9 Exchange of Public-Key Certificates

private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

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, transport layer security (TLS), and S/MIME, all of which are discussed in Part Six. X.509 is examined in detail in the next section.

15.4 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 of the type discussed in Section 15.3. 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 21), IP Security (Chapter 22), and SSL/TLS (Chapter 19).

X.509 was initially issued in 1988. The standard was subsequently revised in 1993 to address some of the security concerns documented in [IANS90] and [MITC90]. The standard is currently at edition eight, issued in 2016.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific digital signature algorithm nor a specific hash function. Figure 15.10 illustrates the overall X.509 scheme for generation of a public-key certificate. The certificate for Bob's public key includes unique identifying information for Bob, Bob's public key, and identifying information about the CA, plus other information as explained subsequently. This information is then signed by computing a hash value of the information and generating a digital signature using the hash value and the CA's private key. Bob can then either broadcast this certificate to other users, or attach the certificate to any document or data block he signs. Anyone who needs to use Bob's public key can be assured that the public key contained in Bob's certificate is valid because the certificate is signed by the trusted CA.

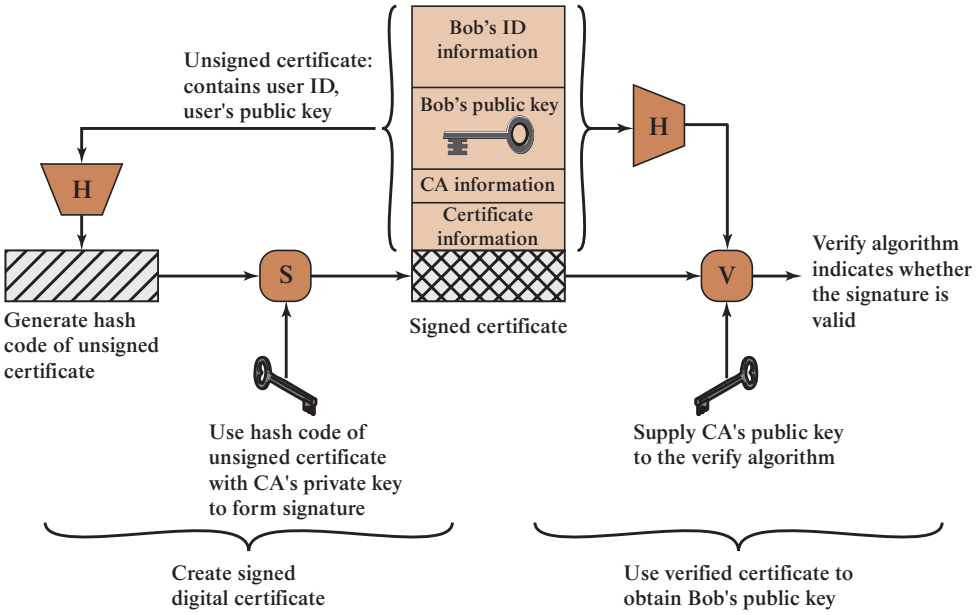


Figure 15.10 X.509 Public-Key Certificate Use

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

Figure 15.11a 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. Although the X.509 specification is currently at version 7, no changes have been made to the fields that make up the certificate since 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.

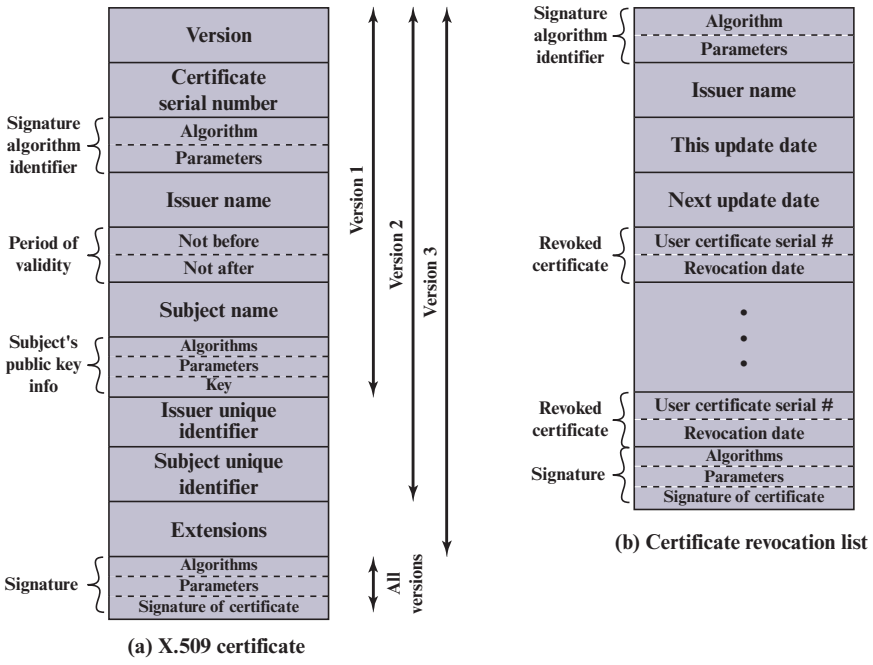


Figure 15.11 X.509 Formats

- **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. One component of this field is the digital signature applied to the other fields of the certificate. 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:

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

where

$Y \ll X \gg$ = the certificate of user X issued by certification authority Y
$Y \{I\}$ = the signing of I by Y. It 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 illustrated in Figure 13.2.

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.

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 (with respect to integrity and authenticity) way 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.

Step 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.

Step 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

$$X_1 \ll X_2 \gg X_2 \ll B \gg$$

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

$$X_2 \ll X_1 \gg X_1 \ll A \gg$$

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

$$X_1 \ll X_2 \gg X_2 \ll X_3 \gg \ \ldots \ X_N \ll B \gg$$

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

All 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 15.12, 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

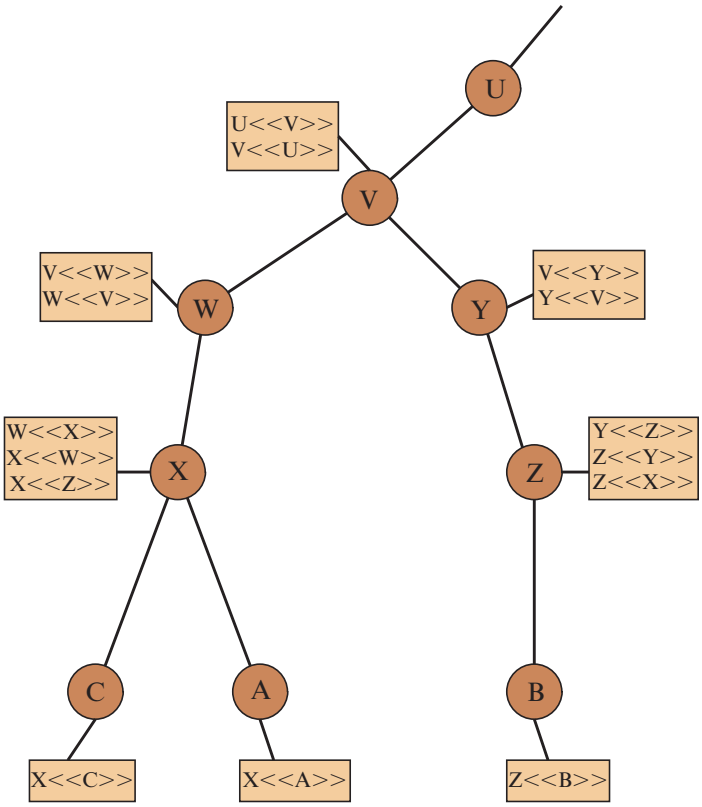


Figure 15.12 X.509 Hierarchy: A Hypothetical Example

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

$$X \ll W \gg \; W \ll V \gg \; V \ll Y \gg \; Y \ll Z \gg \; Z \ll B \gg$$

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

$$Z \ll Y \gg \; Y \ll V \gg \; V \ll W \gg \; W \ll X \gg \; X \ll A \gg$$

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 15.11 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 that the 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 should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 15.11b) 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 email 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 lifecycle 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, 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.

15.5 PUBLIC-KEY INFRASTRUCTURE

NIST SP 800-32 (*Introduction to Public Key Technology and the Federal PKI Infrastructure*) defines a public-key infrastructure (PKI) as a set of policies, processes, server platforms, software, and workstations used for the purpose of administering certificates and public–private key pairs, including the ability to issue, maintain, and revoke public key certificates. The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys.

A PKI architecture defines the organization and interrelationships among CAs and PKI users. PKI architectures satisfy the following requirements:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.
4. Any participant can verify the currency of the certificate.

Figure 15.13 provides a typical architecture for a PKI. The essential components are:

- **End entity:** This can be an end user; a device, such as a router or server; a process; or any item that can be identified in the subject name of a public key certificate. End entities can also be consumers of PKI-related services and, in some cases, providers of PKI-related services. For example, a Registration Authority is considered to be an end entity from the point of view of the Certification Authority.
- **Certification authority (CA):** An authority trusted by one or more users to create and assign public key certificates. Optionally the certification authority may create the subjects' keys. CAs digitally sign public key certificates, which effectively binds the subject's name to the public key. CAs are also responsible for issuing Certificate Revocation Lists (CRLs). The CRL identifies certificates previously issued by the CA that are revoked before their expiration date. A certificate could be revoked because the user's private key is assumed to be compromise, the user is no longer certified by this CA, or the certificate is assumed to be compromised.
- **Registration authority (RA):** An optional component that can be used to offload many of the administrative functions that a CA ordinarily assumes. The RA is normally associated with the end entity registration process. This includes the verification of the identity of the end entity attempting to register with the PKI and obtain a certificate for its public key.
- **Repository:** Denotes any method for storing and retrieving PKI-related information, such as public key certificates and CRLs. A repository can be an X.500-based directory with client access via the Lightweight Directory Access Protocol (LDAP). It also can be something simple, such as a means for

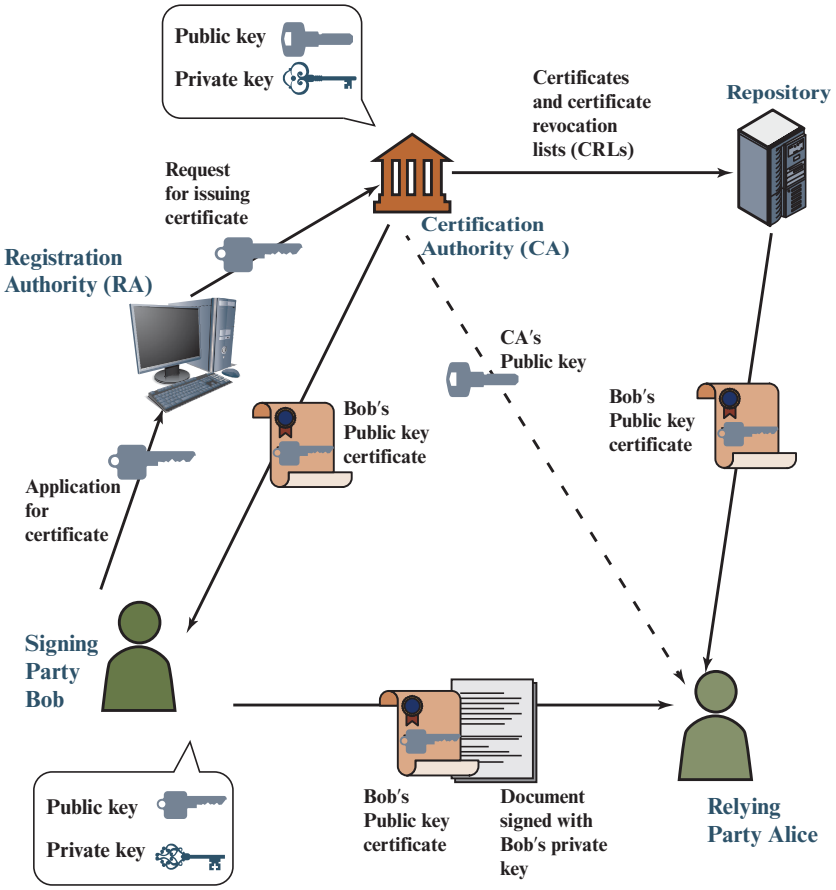


Figure 15.13 PKI Scenario

retrieval of a flat file on a remote server via the File Transfer Protocol (FTP) or the Hyper Text Transfer Protocol (HTTP).

- **Relying party:** Any user or agent that relies on the data in a certificate in making decisions.

Figure 15.13 illustrates the interaction of the various components. Consider a relying party Alice that needs to use Bob's public key. Alice must first obtain in a reliable, secure fashion a copy of the public key of the CA. This can be done in a number of ways and depends on the particular PKI architecture and enterprise policy. If Alice wishes to send encrypted data to Bob, Alice checks with the Repository to determine if Bob's certificate has been revoked, and if not obtains a copy of Bob's certificate. Alice can then use Bob's public key to encrypt data sent to Bob. Bob can also send a document to Alice signed with Bob's private key. Bob may include his certificate with the document or assume that Alice already has or can obtain the certificate. In either case, Alice first uses the CA's public key to verify that the certificate is valid, then uses Bob's public key (obtained from the certificate) to validate Bob's signature.

Rather than a single CA, an enterprise may need to rely on multiple CAs and multiple repositories. CAs can be organized in a hierarchical fashion, with a root CA that is widely trusted signing the public key certificate of subordinate CAs. Many root certificates are embedded in Web browsers so they have built-in trust of those CAs. Web servers, email clients, smartphones and many other types of hardware and software also support PKI and contain trusted root certificates from the major CAs.

15.6 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

end-to-end encryption key distribution key distribution center (KDC)	key management man-in-the-middle attack master key	public-key certificate

Review Questions

15.1 Explain why man-in-the-middle attacks are ineffective on the secret key distribution protocol discussed in Figure 15.8.

15.2 What is the difference between a session key and a master key?

15.3 What is a key distribution center?

15.4 What is one role that nonces play in key distribution using public-key cryptography?

15.5 List four requirements for the distribution of public keys using the public-key certificates scheme.

15.6 Discuss the potential security issues that arise due to a public-key–directory-based system.

15.7 What is a public-key certificate?

15.8 What are the requirements for the use of a public-key certificate scheme?

15.9 What is the purpose of the X.509 standard?

15.10 What types of certificates does an X.509 CA's directory entry contain?

15.11 What is a certificate revocation list?

Problems

15.1 One local area network vendor provides a key distribution facility, as illustrated in Figure 15.14. Describe the operation of the scheme.

15.2 "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."

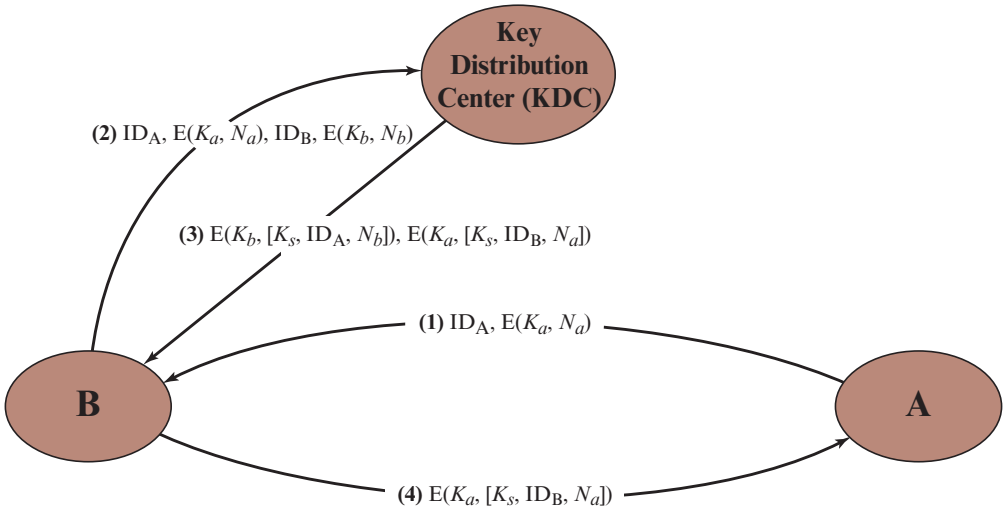


Figure 15.14 Figure for Problem 15.1

"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.

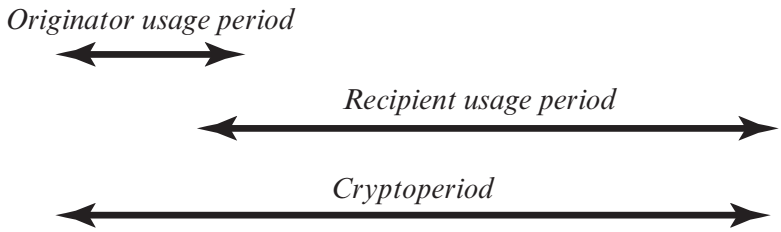
15.3 The 1988 version of X.509 lists properties that RSA keys must satisfy to be secure given current knowledge about the difficulty of factoring large numbers. The discussion concludes with a constraint on the public exponent and the modulus n:

> 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?

15.4 Determine the chain of certificates associated with the website *www.pearson.com*, describing the type of each CA.

15.5 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 recipients usage period.

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

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

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

15.9 What is the effect of adding the instruction EMKi

$$\text{EMK}_i\text{: } X \rightarrow \text{E}(KMH_i, X)\ i = 0, 1$$

15.10 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.

15.11 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.