Key management and Distribution 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. Therefore, the 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.

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 is mostly based on 1 or 2 occurring first.

A third party, whom all parties trust, can be used as a **trusted intermediary** to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As number of parties grow, some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

Key distribution centre:

- The use of a **key distribution center** is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used.
- Communication between end systems is encrypted using a temporary key, often referred to as a **Session key**.
- Typically, the session key is used for the duration of a logical connection and then discarded
- **Master key** is shared by the key distribution center and an end system or user and used to encrypt the session key.

Key Distribution Scenario:

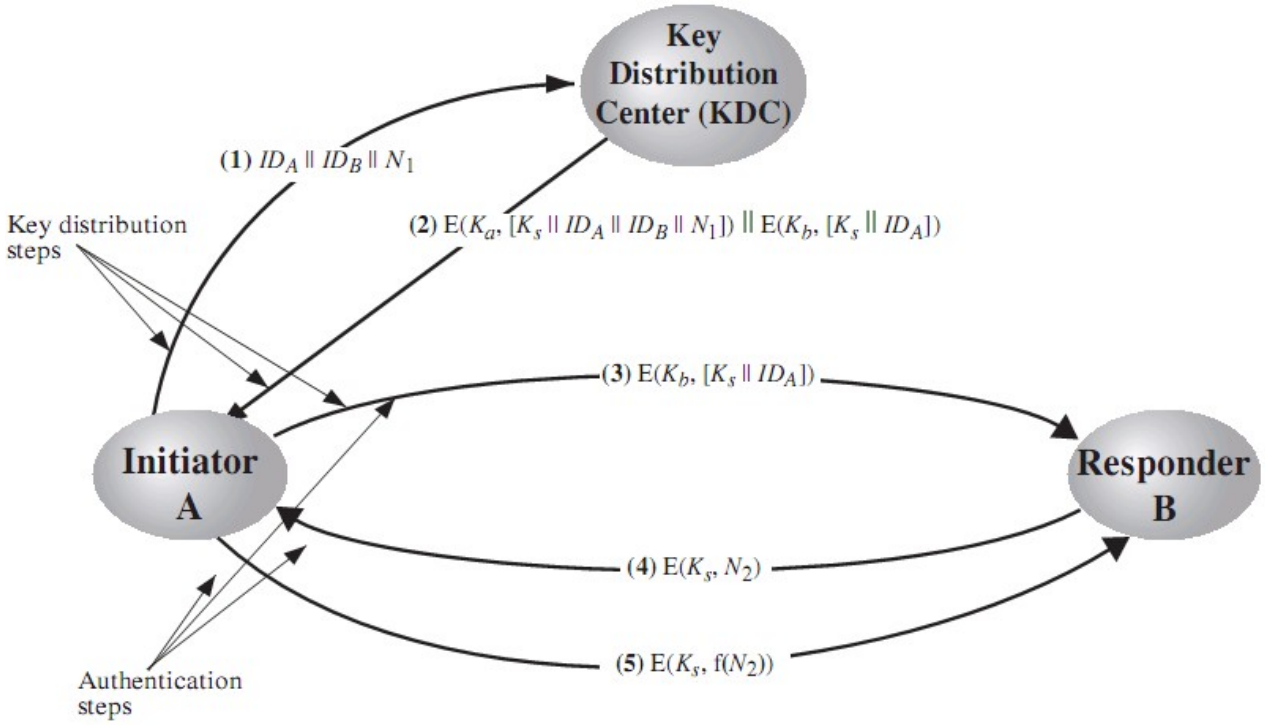


Figure 14.3 Key Distribution Scenario

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, K_a, known only to itself and the KDC; similarly, B shares the master key K_b with the KDC. The following steps occur:

1 A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1, for this transaction, which we refer to as a **nonce**. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is **that it differs with each request.** Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.

2. The KDC responds with a message encrypted using Ka Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:

- The **one-time session key, Ks,** to be used for the session
- The **original request message,** including the nonce, to enable A to match this response with the appropriate request

Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request. In addition, the message includes two items intended for B:

- The one-time session key, Ks to be used for the session
- An identifier of A (e.g., its network address), IDA

These last two items are encrypted with Kb (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, $E(K_b, [K_s \| ID_A])$. Because this information is encrypted with K_b, it is protected from eavesdropping. B now knows the session key (K_s), knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b). At this point, a session key has been securely delivered to A and B, and they may begin

their protected exchange. However, two additional steps are desirable:

4. Using the newly minted session key for encryption, B sends a nonce, N_2, to A.

5. Also using K_s, A responds with $f(N_2)$, where f is a function that performs some transformation on N_2 (e.g., adding one).

These steps assure B that the original message it received (step 3) was not a replay.

Note that the actual key distribution involves only steps 1 through 3 but that steps 4 and 5, as well as 3, perform an authentication function.

Major Issues with KDC:

Hierarchical Key Control

- It is not **necessary to limit** the key distribution function to a single KDC.Indeed,for very large networks,it may not be practical to do so.As an alternative,a hierarchy of KDCs can be established.
- For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building.
- If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC.
- The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.
- A hierarchical scheme **minimizes the effort involved in master key distribution**, because most master keys are those shared by a local KDC with its local entities.

Session Key Lifetime

- The distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.

- For **connection-oriented protocols**, one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session.
- If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.
- For a **connectionless protocol**, such as a transaction-oriented protocol, there is no explicit connection initiation or termination.
- Thus, it is not obvious how often one needs to change the session key. The most secure approach is to use a **new session key for each exchange**.
- A better strategy is to use a given **session key for a certain fixed period only or for a certain number of transactions.**

A Transparent Key Control Scheme

- The approach suggested in Figure 14.3is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users.
- The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP.
- The noteworthy element of this approach is a session security module (SSM), which may consist of functionality at one protocol layer,that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

The steps involved in establishing a connection are shown in Figure 14.4.

1. When one host wishes to set up a connection to another host, it transmits a connection-request packet.
2. The SSM saves that packet and applies to the KDC for permission to establish the connection.
3. The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC.If the KDC approves the

connection request,it generates the session key and delivers it to the two appropriate SSMs,using a unique permanent key for each SSM.

4. The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems.
5. All user data exchanged between the two end systems are encrypted by their respective SSMs using the onetime session key.

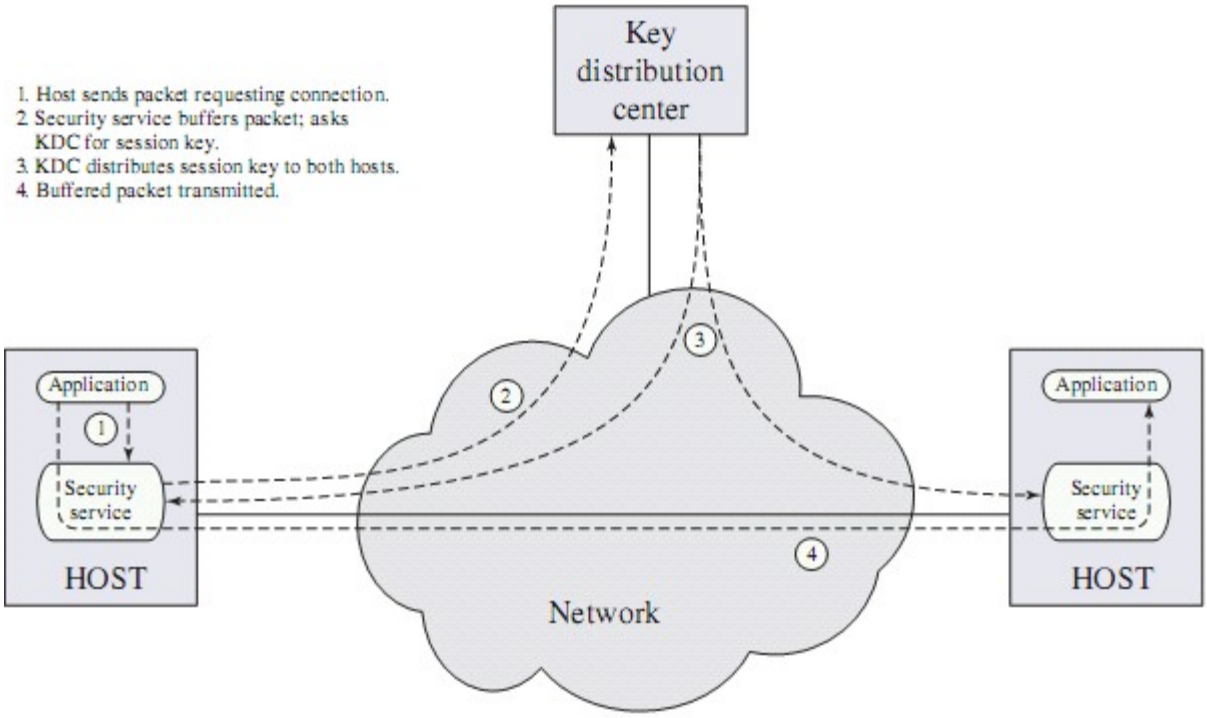


Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

- The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

Decentralized Key Control

- The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion. This requirement can be avoided if key distribution is fully decentralized.

- Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.
- A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution.
- Thus, there may need to be as many as $n\ (n\ -1)/2$ master keys for a configuration with n end systems.
- A session key may be established with the following sequence of steps (Figure 14.5).

1. A issues a request to B for a session key and includes a nonce, .

2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B,an identifier of B,the value f(N_1), and another nonce N_2.

3. Using the new session key,A returns f($N_{2)}$ to B.

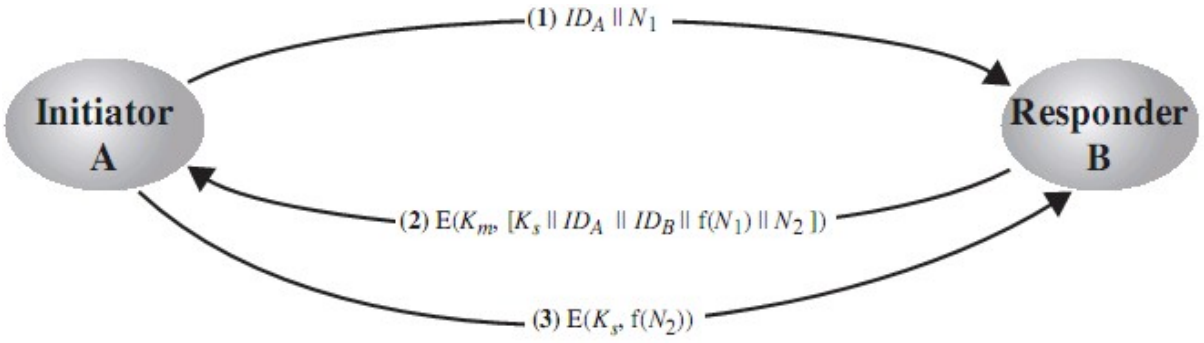


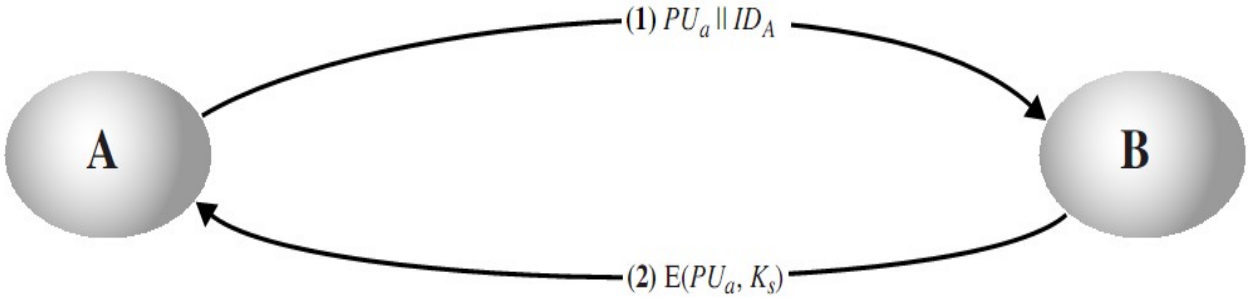
Figure 14.5 Decentralized Key Distribution

SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

- Once public keys have been distributed or have become accessible, secure communication that thwarts eavesdropping, tampering, or both, is possible.
- Public-key encryption provides for the distribution of secret keys to be used for conventional encryption.

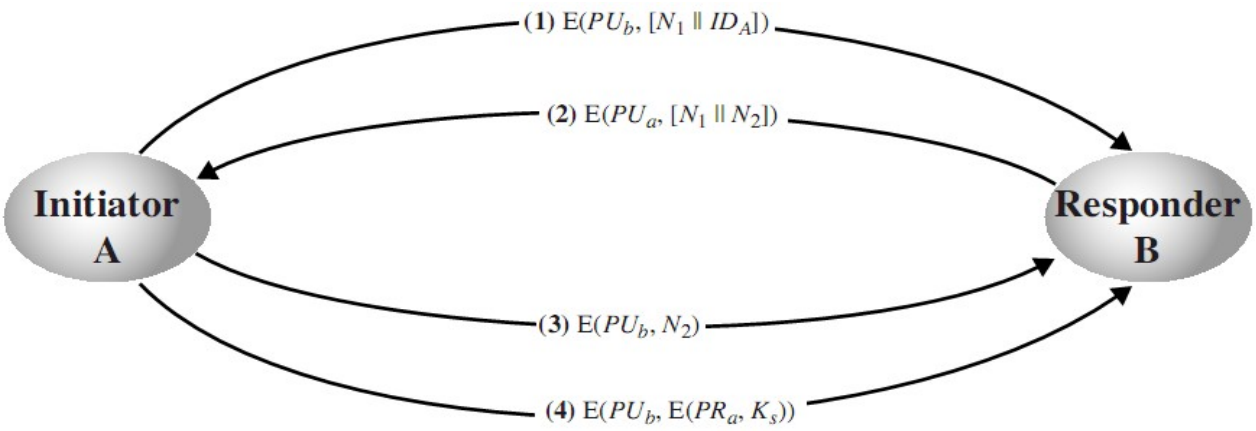
Simple Secret Key Distribution

- A generates a public/private key pair {PUa, PRa} and transmits a message to B consisting of PUa and an identifier of A, IDA
- B generates a secret key, Ks, and transmits it to A, encrypted with A's public key.
- A computes D(PRa, E(PUa, Ks)) to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of Ks.
- A discards PUa and PRa and B discards PUa.



Here third party can intercept messages and then either relay the intercepted message or substitute another message Such an attack is known as a **man-in-the-middle attack.**

Secret Key Distribution with Confidentiality and Authentication:



- 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
- 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
- A returns N_2 encrypted using B's public key, to assure B that its correspondent is A.
- A selects a secret key K_s and sends M = E(PU_b, 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.
- B computes D(PU_a, D(PR_b, M)) to recover the secret key.

Distribution of Public Keys:

Several techniques have been proposed for the distribution of public keys, which can mostly be grouped into the categories shown.

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

Public Announcement of Public Keys

The point of public-key encryption is that the public key is public, hence any participant can send his or her public key to any other participant, or broadcast the key to the community at large. eg. append PGP keys to email messages or post to news groups or email list

Figure 10.1 Uncontrolled Public Key Distribution

Its major weakness is forgery, anyone could pretend to be user A and send a public key to another participant or broadcast such a public key. Until the forgery is discovered they can masquerade as the claimed user.

Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:

  - The authority maintains a directory with a {name, public key} entry for each participant.
  - Each participant registers a public key with the directory authority.
  - A participant may replace the existing key with a new one at any time because the corresponding private key has been compromised in some way.
  - Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

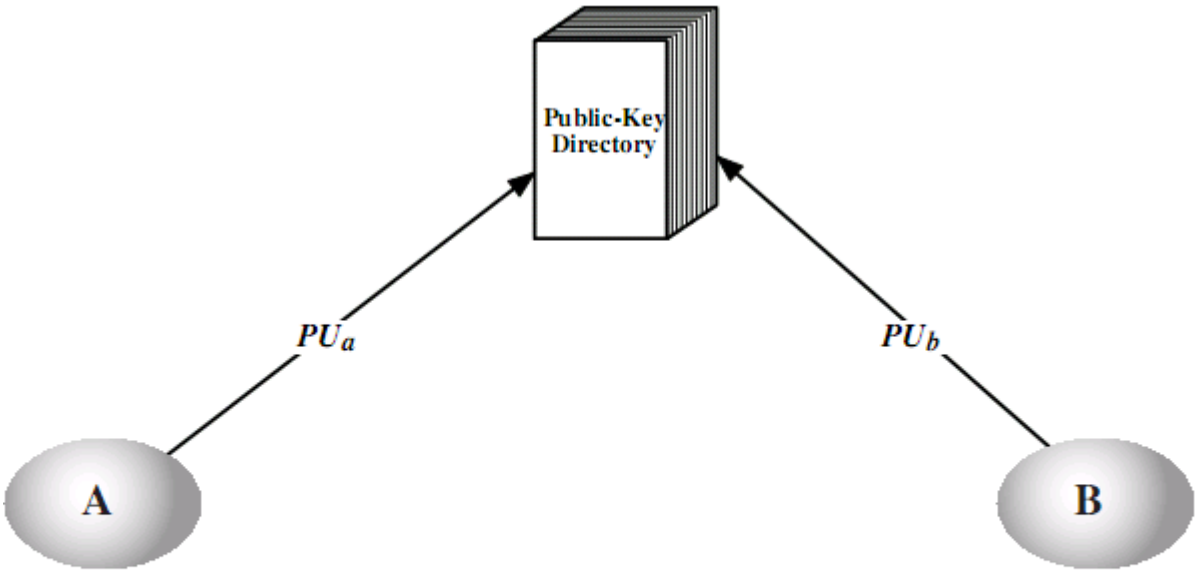


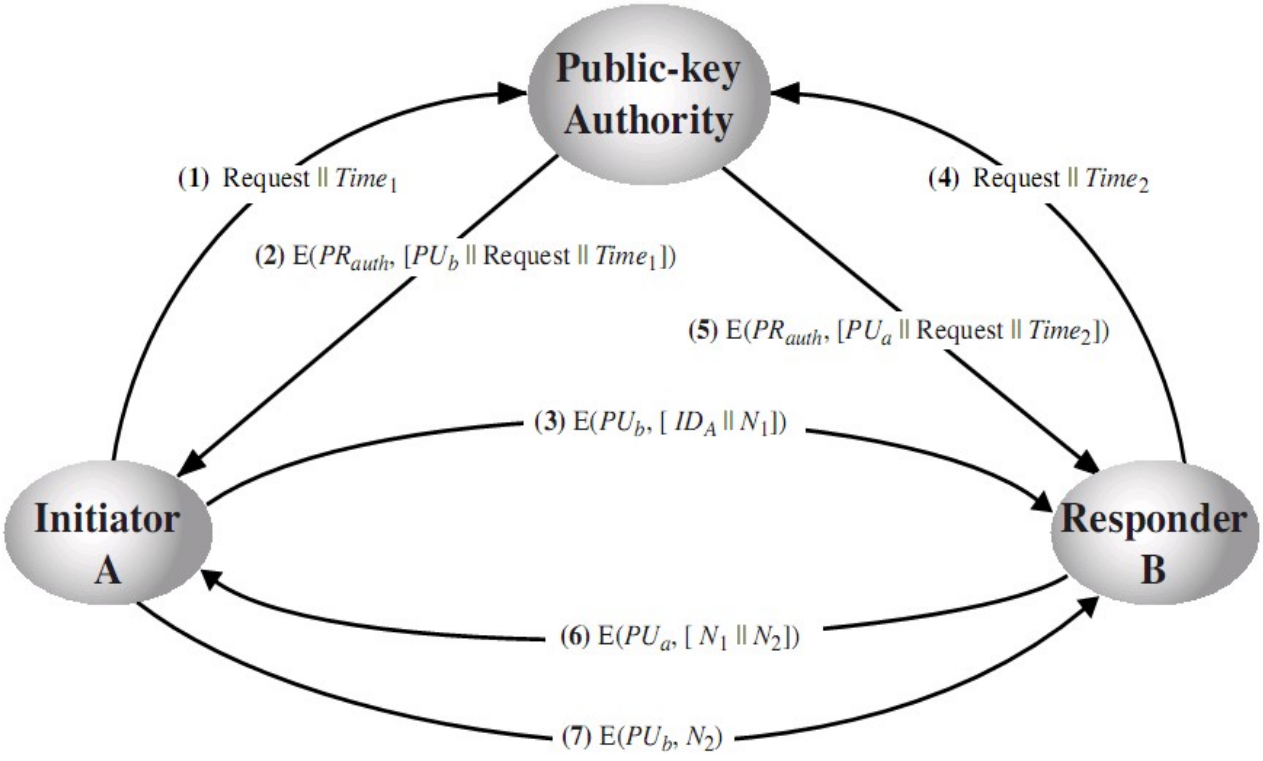
Figure 10.2 Public Key Publication

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.
- It requires users to know the public key for the directory, and that they interact with directory in real-time to obtain any desired public key securely.
- Totally seven messages are required.



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:

    - B's public key, PU_b which A can use to encrypt messages destined for B
    - The original request, 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, 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. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.
5. 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, encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Certificates

- A user must appeal to the authority for a public key for every other user that it wishes to contact and it is vulnerable to tampering too.
- Public key certificates can be used to exchange keys without contacting a public-key authority.
- A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA).
- This can be verified by anyone who knows the public-key authorities public-key.

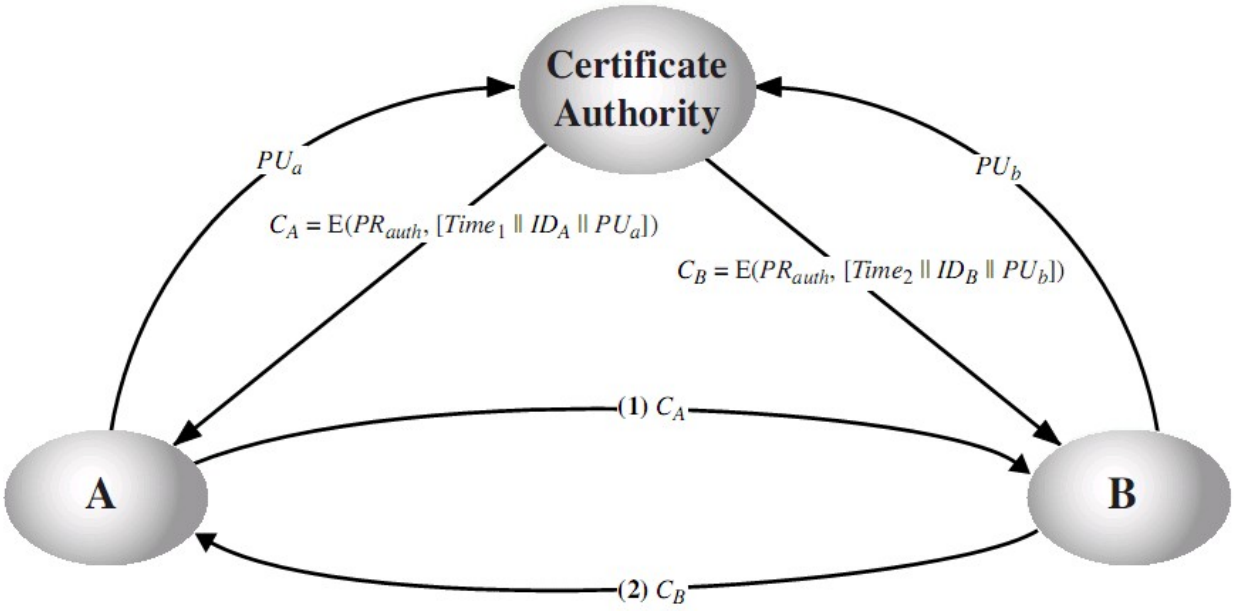
A participant can also convey its key information to another by transmitting its certificate.

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.
4. Any participant can verify the currency of the certificate.

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 transactions (SET), and S/MIME.



X.509 CERTIFICATES

X.509 is part of the X.500 series of recommendations that define a directory service, being a server or distributed set of servers that maintains a database of information about 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 based on the use of public-key cryptography and digital signatures.

The X.509 certificate format is widely used, in for example S/MIME, IP Security and SSL/TLS and SET. X.509 was initially issued in 1988. The standard was ubsequently revised to address some of the security concerns; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

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.

The standard uses the notation for a certificate of:

CA<<A>> where the CA signs the certificate for user A with its private key. In more detail CA<<A>> = CA {V, SN, AI, CA, UCA, A, UA, Ap, TA}.

If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

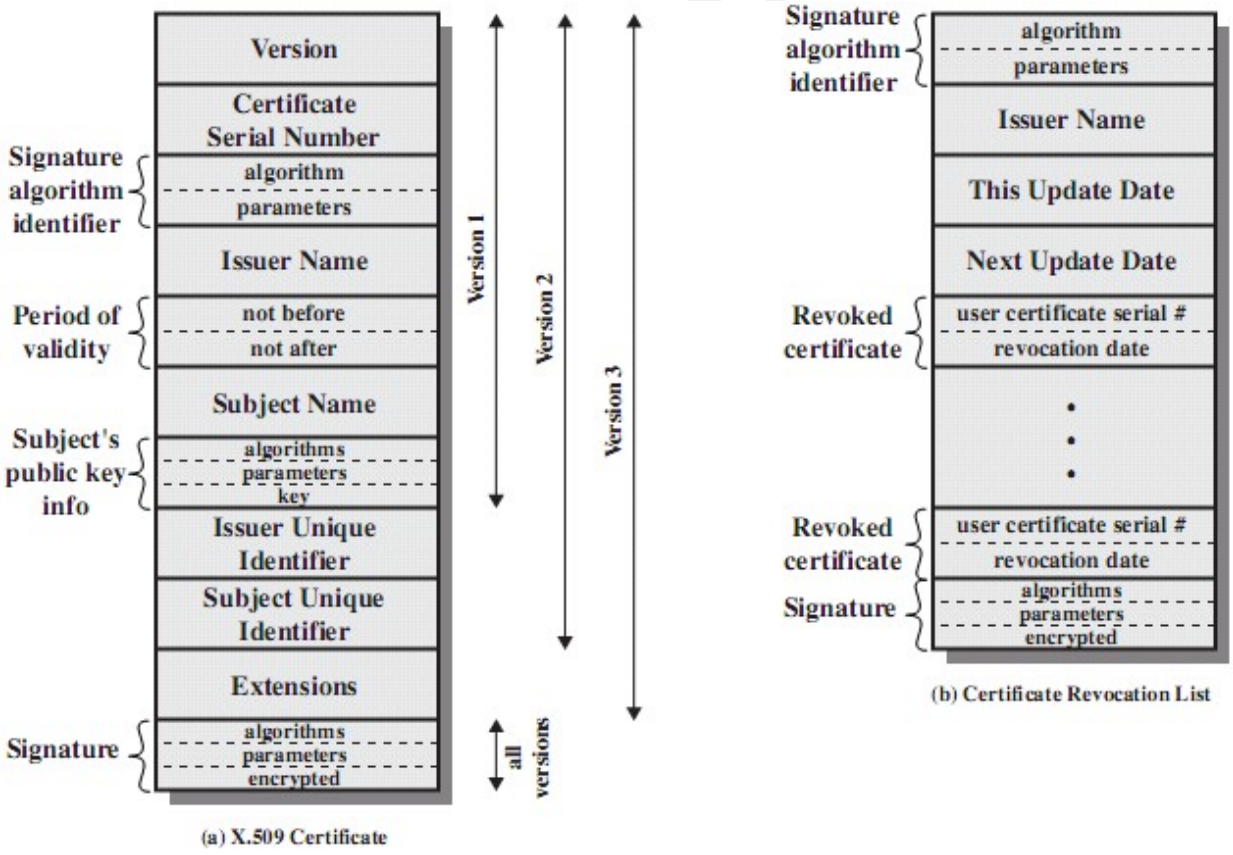


Figure 14.4 X.509 Formats

- **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 is the 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<< X >> = 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

Obtaining a 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 with out 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.

CA Hierarchy:

If both parties use the same CA, they know its public key and can verify others certificates. If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Hence there has to be some means to form a chain of certifications between the CA's used by the two parties, by the use of client and parent certificates. 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. It is assumed that each client trusts its parent's certificates.

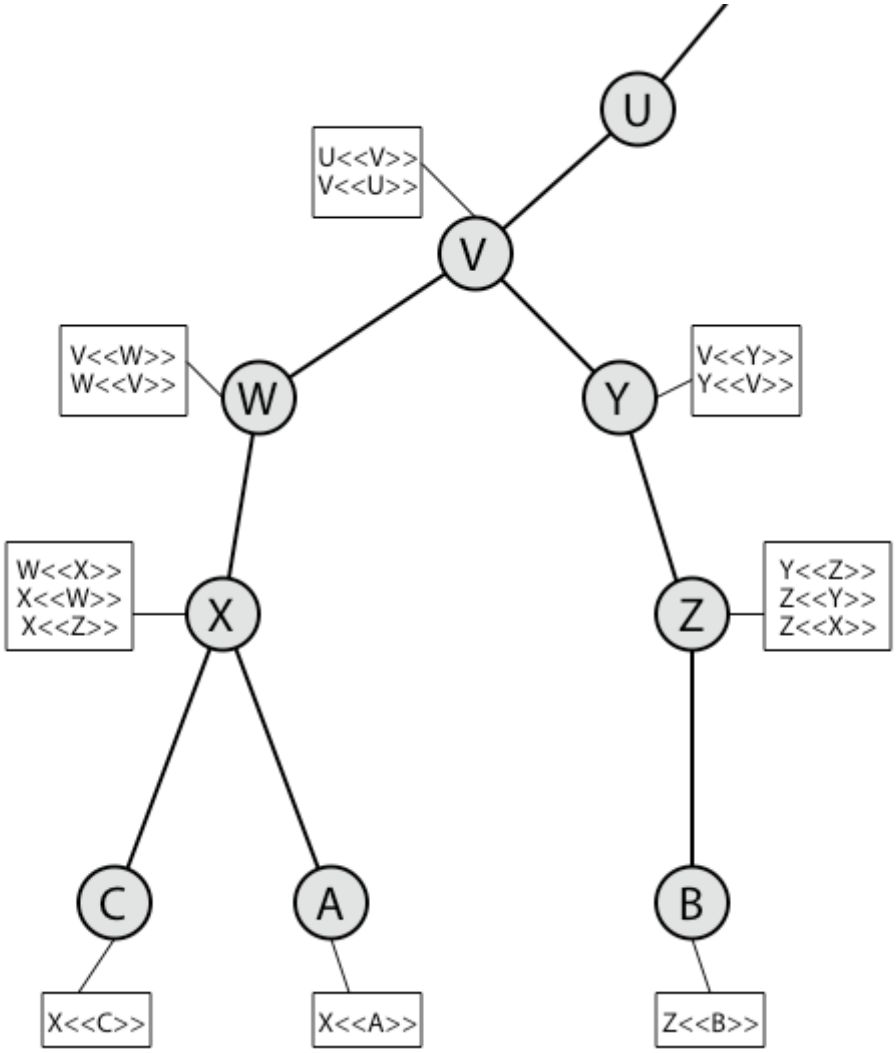


Figure 14.15 illustrates the use of an X.509 hierarchy to mutually verify clients certificates. The connected circles indicate the hierarchical relationship among the CAs; the

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, we can track chains of certificates as follows:

A acquires B certificate using chain

X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

B acquires A certificate using chain:

Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>

Certificate Revocation:

A certificate includes a period of validity. 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 a range of 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.

To support this, each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, known as the certificate revocation list (CRL). Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (as shown in Figure 14.14b previously) 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, by checking the directory CRL each time a certificate is received, this often does not happen in practice.

X.509 Version 3

The X.509 version 2 format does not convey all of the information. Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. X.509 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 or not.

Certificate Extensions

The certificate extensions fall into three main categories:

- **Key and policy information** - 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.
- **Subject and issuer attributes** - support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject; eg. postal address, email address, or picture image
- **Certification path constraints** - allow constraint specifications to be included in certificates issued for CA's by other CA's that may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

****