Chapter 14

Key Management and Distribution

Key Distribution Technique

- 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 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
- Frequent key changes are desirable to limit the amount of data compromised if an attacker learns the key

Symmetric Key Distribution

Given parties A and B, key distribution can be achieved in a number of ways:

- A can select a key and physically deliver it to B
- A third party can select the key and physically deliver it to A and B
- 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
- 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,
➢ each link encryption device is going to be exchanging data only with its partner on the other end of the link.

For end-to-end encryption over a network, manual delivery is awkward.
➢ Any given host may need to engage in exchanges with many other hosts and terminals over time.
Thus, each device needs a number of keys supplied dynamically.

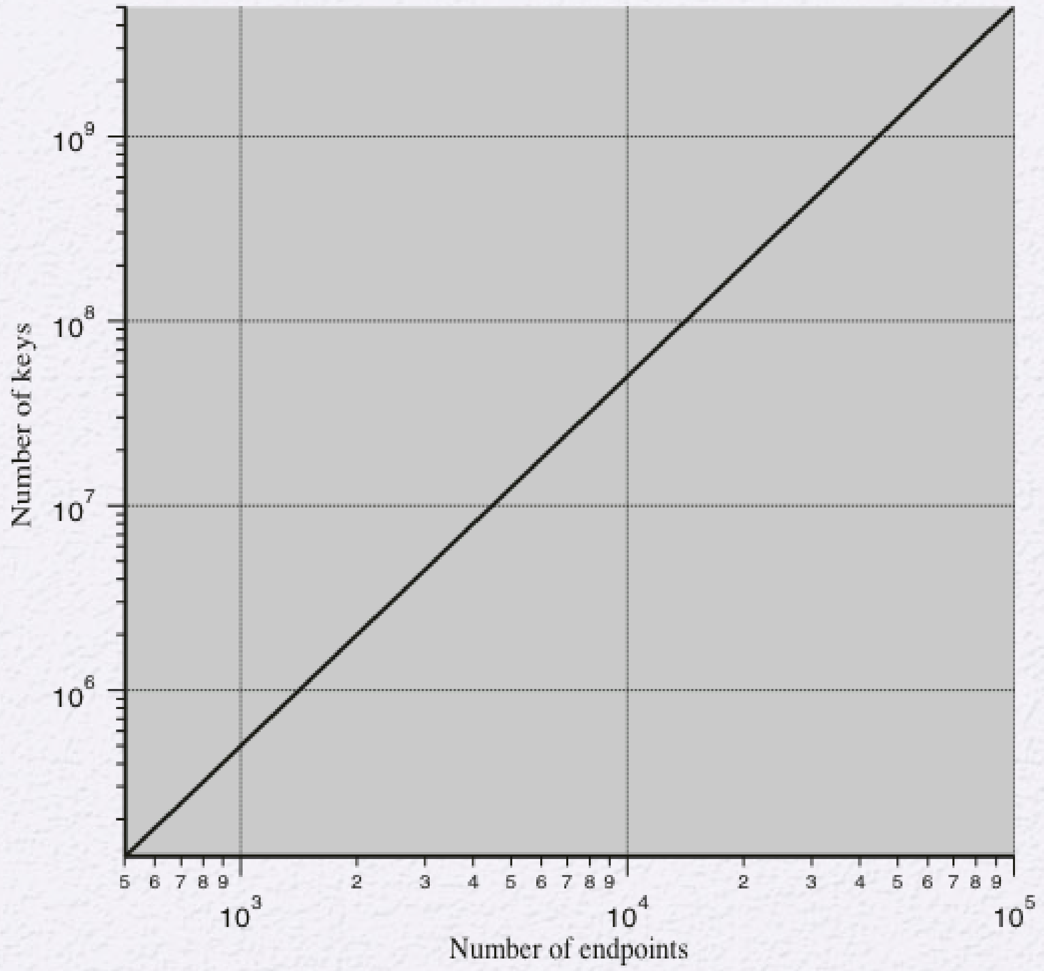


Figure 14.1 Number of Keys Required to Support Arbitrary Connections Between Endpoints

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. Thus, if there are *N* hosts, the number of required keys is *[N(N – 1)]/2*.

If encryption is done at the application level, then a key is needed for every pair of users that require communication. Thus, a network may have hundreds of hosts but thousands of users. Figure 14.1 illustrates the magnitude of the key distribution task for end-to-end encryption.

For end-to-end encryption, some variation on option 4 has been widely adopted.

- a key distribution center is responsible for distributing keys to pairs of users.
- Each user must share a unique key with the key distribution center for purposes of key distribution.

The use of a key distribution center is based on the use of a hierarchy of keys. Two levels of keys are used.

- Communication between end systems is encrypted using a temporary key, called a session key.
- Each session key is obtained from the key distribution center. session keys are transmitted in encrypted form, using a master key that is shared by the key distribution center and an end system or user.

Master keys must be distributed in some fashion. However, the scale of the problem is vastly reduced. If there are N entities that wish to communicate in pairs, then, as was mentioned, as many as [N (N - 1)]/2 session keys are needed at any one time. However, only N master keys are required, one for each entity. Thus, master keys can be distributed in some non-cryptographic way, such as physical delivery.

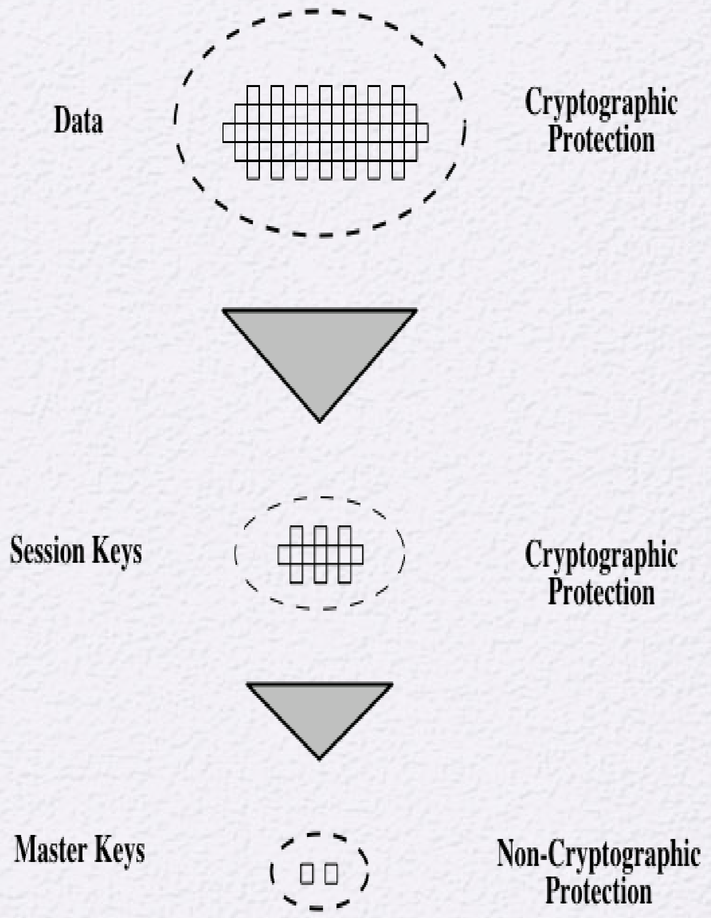


Figure 14.2 The Use of a Key Hierarchy

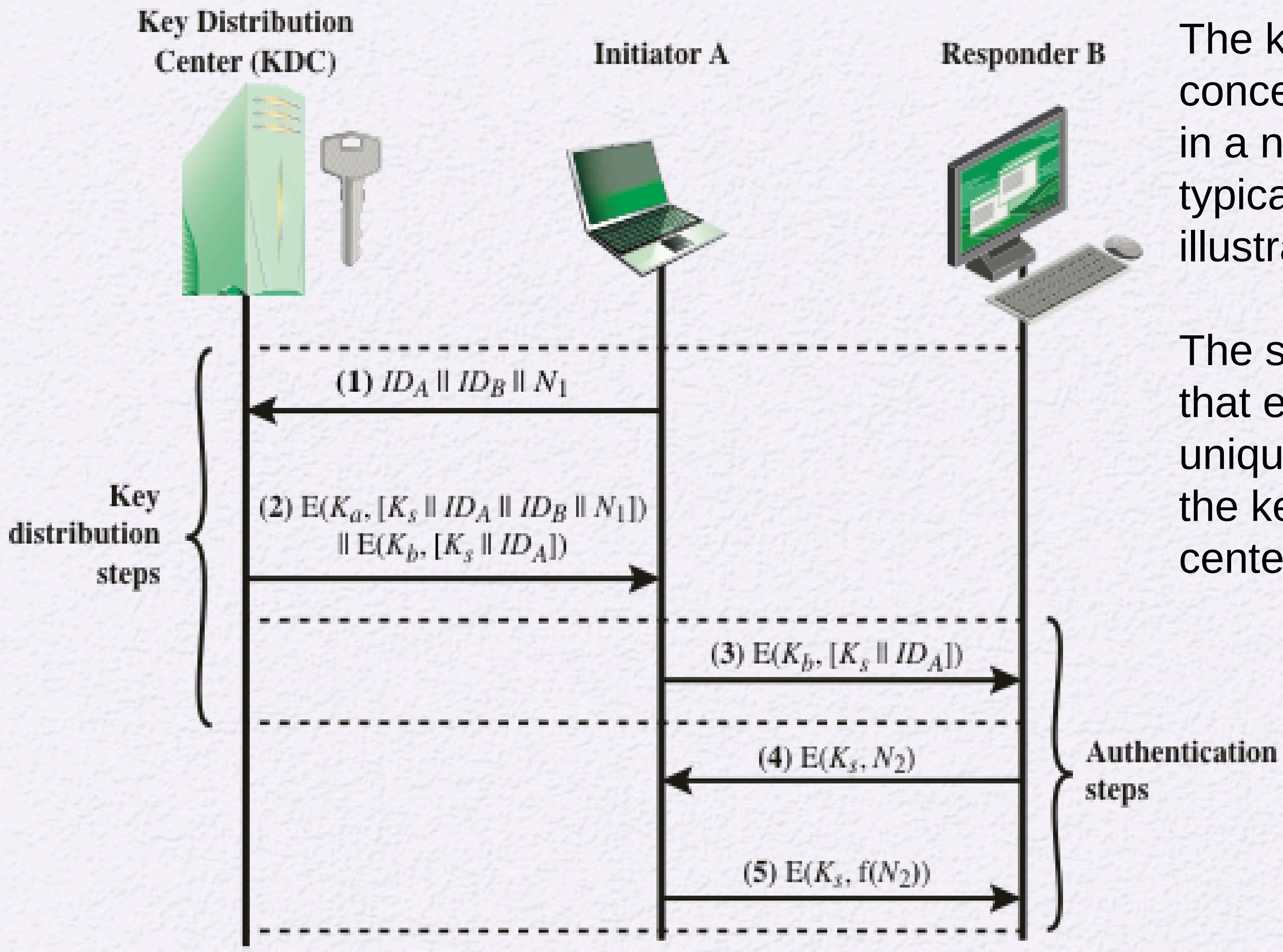

Key Distribution Center (KDC)
Initiator A
Responder B
(1) $ID_A \| ID_B \| N_1$
(2) $E(K_a, [K_s \| ID_A \| ID_B \| N_1]) \| E(K_b, [K_s \| ID_A])$
Key distribution steps
(3) $E(K_b, [K_s \| ID_A])$
(4) $E(K_s, N_2)$
(5) $E(K_s, f(N_2))$
Authentication steps

Figure 14.3 Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 14.3.

The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

Hierarchical Key Control

- For communication among entities within the same local domain, the local KDC is responsible for key distribution
  - If two entities in different domains desire a shared key, then the corresponding local KDC's can communicate through a global KDC
- The hierarchical concept can be extended to three or more layers

Session Key Lifetime

A security manager must balance competing considerations:

- The more frequently session keys are exchanged, the more secure they are, because the attacker has less ciphertext to work with for any given session key
- The distribution of session keys delays the start of any exchange and places a burden on network capacity

The approach suggested in Figure 14.3 has many variations.

- The scheme provides encryption at a network or transport level in a way that is transparent to the end users.
- a session security module (SSM), which may consist of functionality at one protocol layer, that performs encryption and obtains session keys on behalf of its host.

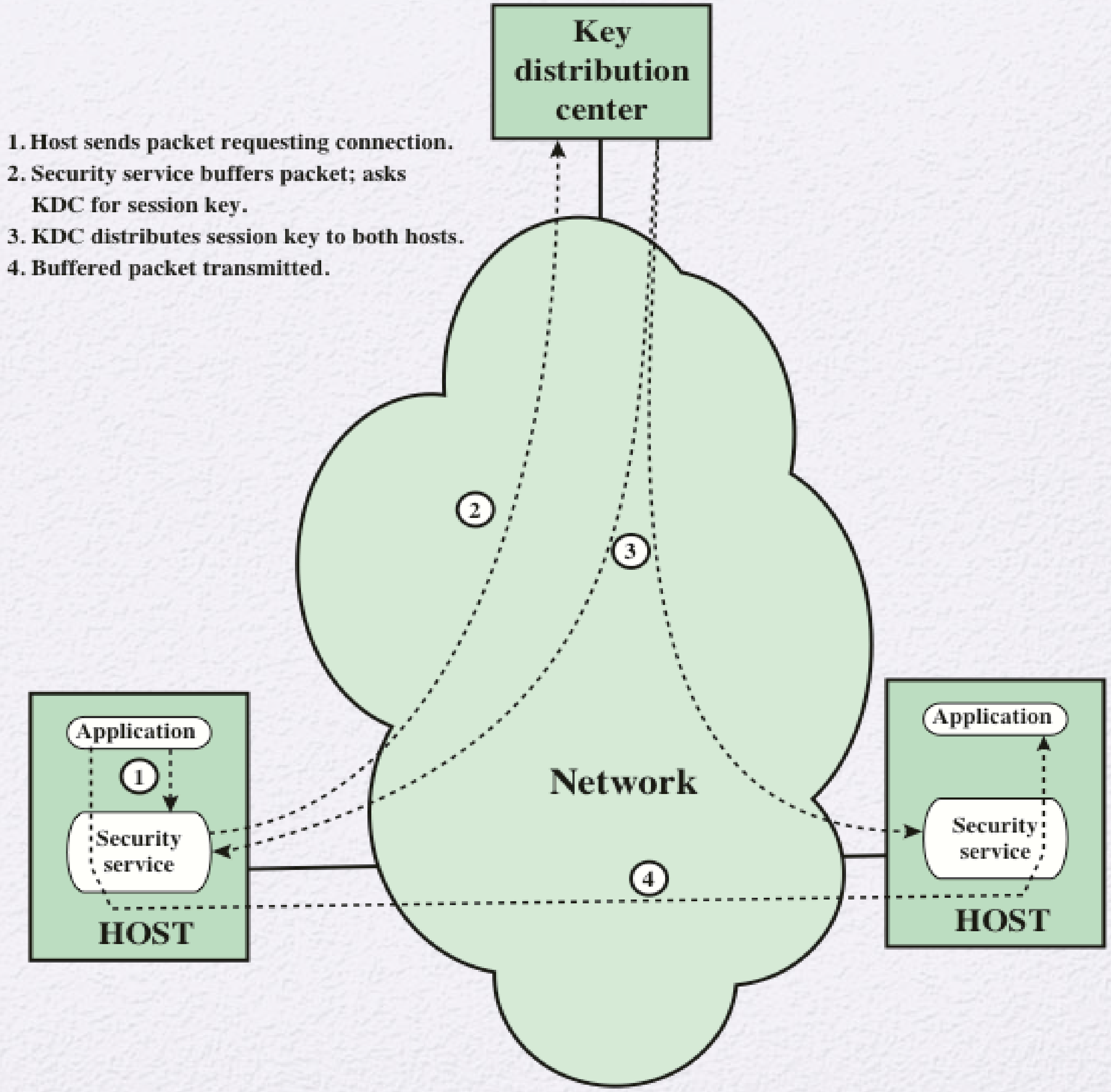

Key distribution center
1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.
Network
Application
1
Security service
HOST
2
3
4
Application
Security service
HOST

Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

➢KDC must be trusted and be protected from attack. This requirement can be avoided if key distribution is fully decentralized.
➢Although full decentralization is not practical for larger networks using symmetric encryption, it may be useful within a local context.
➢A decentralized approach requires that each host be able to communicate in a secure manner with all potential host for purposes of session key distribution. Thus, there may need to be as many as [*n (n - 1)]/2* master keys for *n* hosts.
Here, Km is the shared master key.

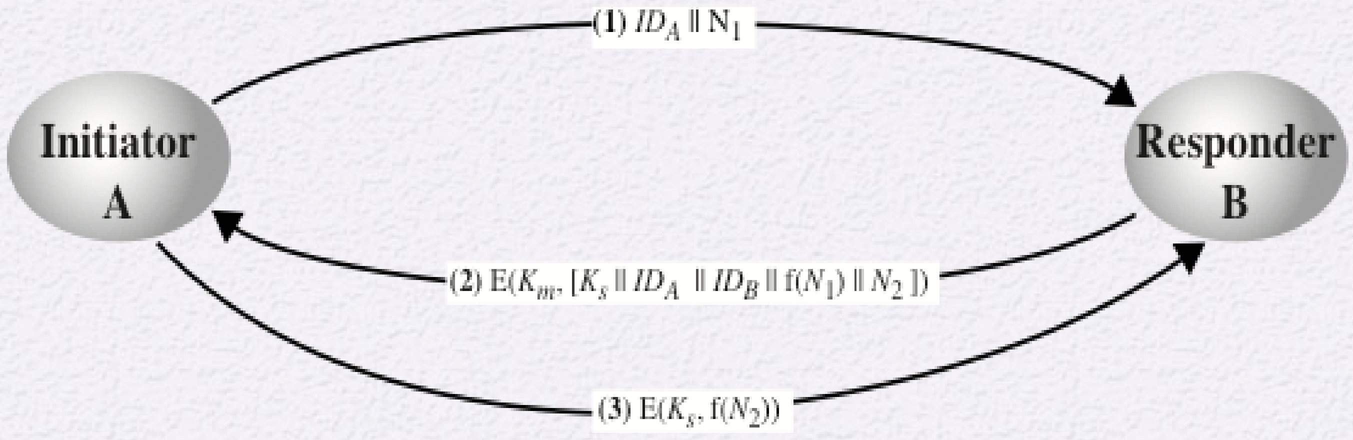

Initiator A
Responder B
(1) $ID_A \parallel N_1$
(2) $E(K_m, [K_s \parallel ID_A \parallel ID_B \parallel f(N_1) \parallel N_2])$
(3) $E(K_s, f(N_2))$

Figure 14.5 Decentralized Key Distribution

Controlling Key Usage

- The concept of a key hierarchy greatly reduce the number of keys that must be manually managed and distributed
- It also may be desirable to impose some control on the way in which automatically distributed keys are used
  - For example, in addition to separating master keys from session keys, we may wish to define different types of session keys on the basis of use

Public key-based Key Distribution

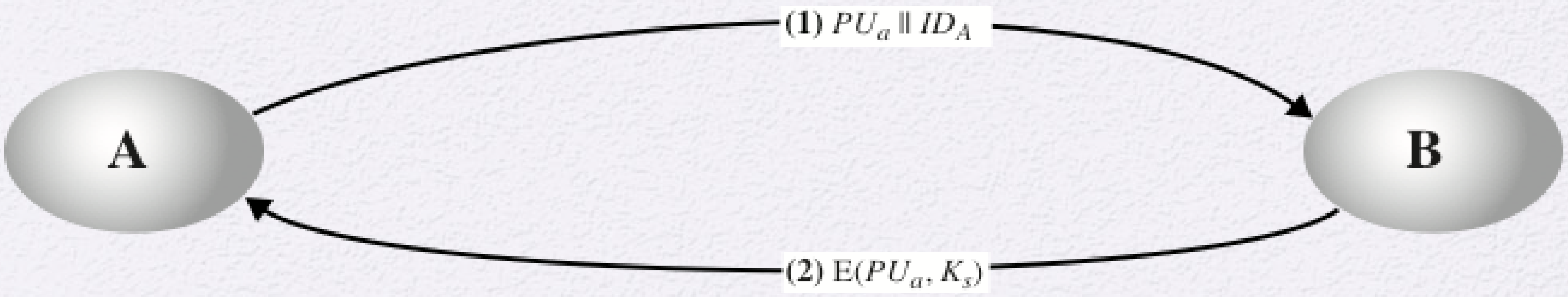


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

No keys exist before the start of the communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping. But it is vulnerable to man in the middle attack.

Man-in-the-Middle Attack

The protocol is insecure against an adversary who 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.

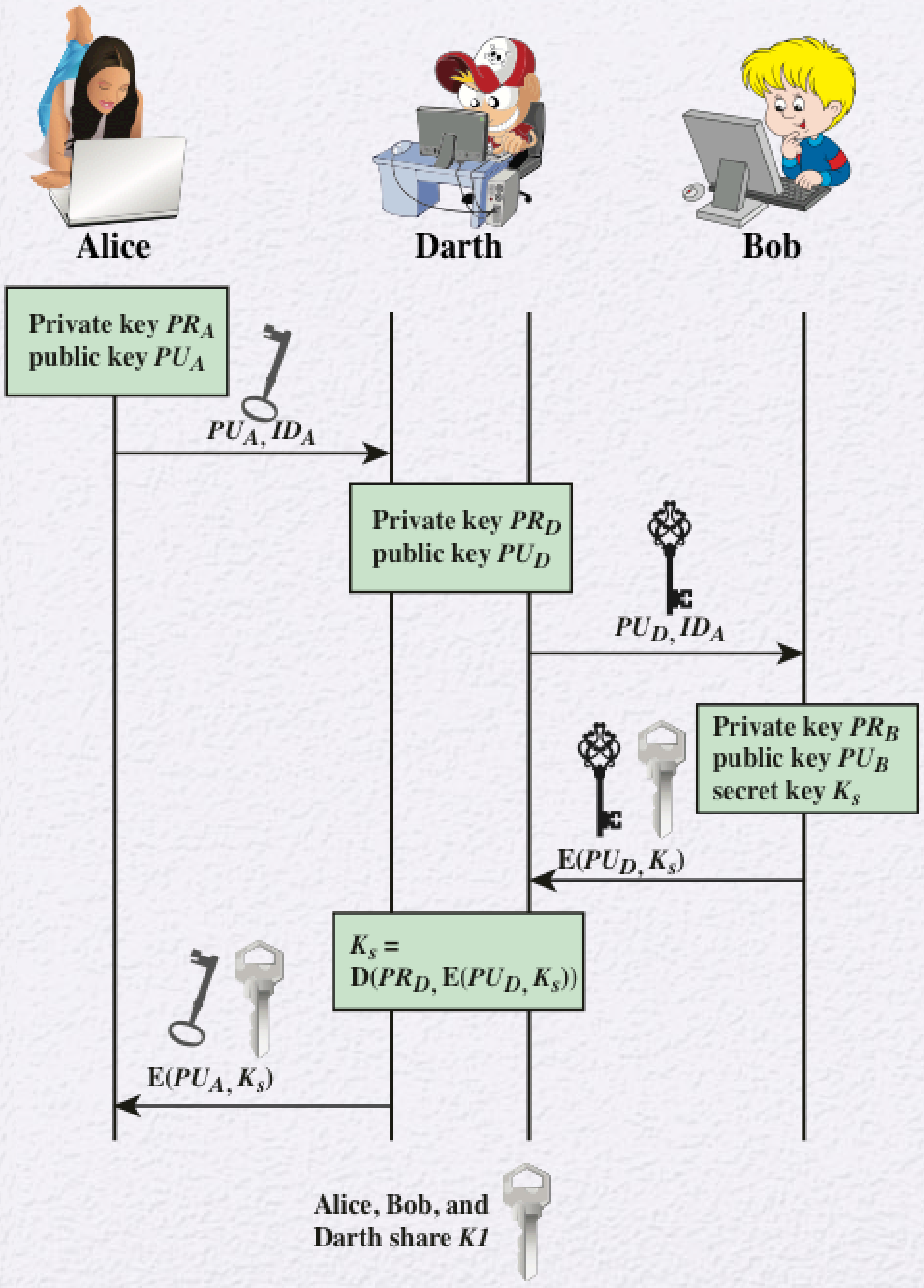


Figure 14.8 Another Man-in-the-Middle Attack

Secret Key Distribution with Confidentiality and Authentication

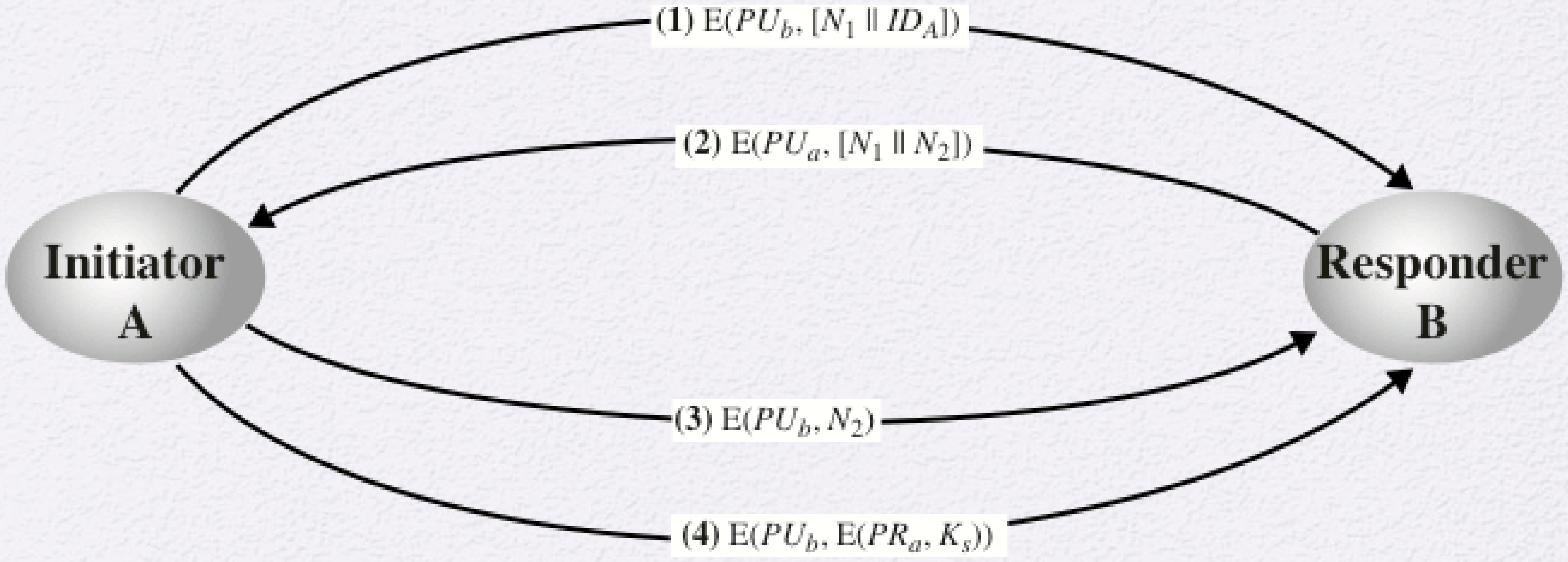

(1) $E(PU_b, [N_1 \| ID_A])$
(2) $E(PU_a, [N_1 \| N_2])$
Initiator A
Responder B
(3) $E(PU_b, N_2)$
(4) $E(PU_b, E(PR_a, K_s))$

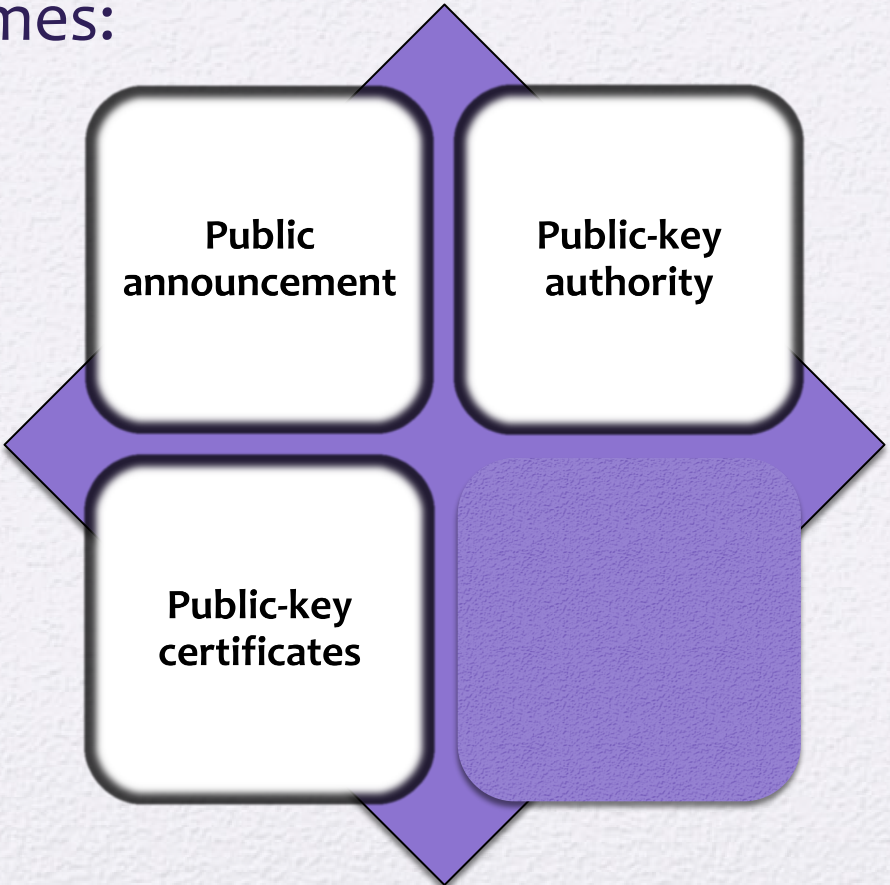
Figure 14.9 Public-Key Distribution of Secret Keys

A Hybrid Scheme

- In use on IBM mainframes
- Retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key
- A public-key scheme is used to distribute the master keys

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

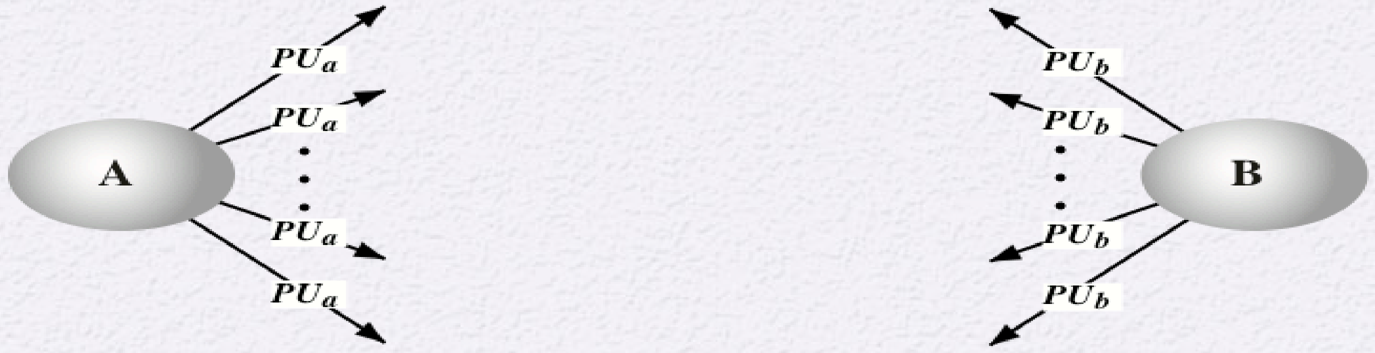
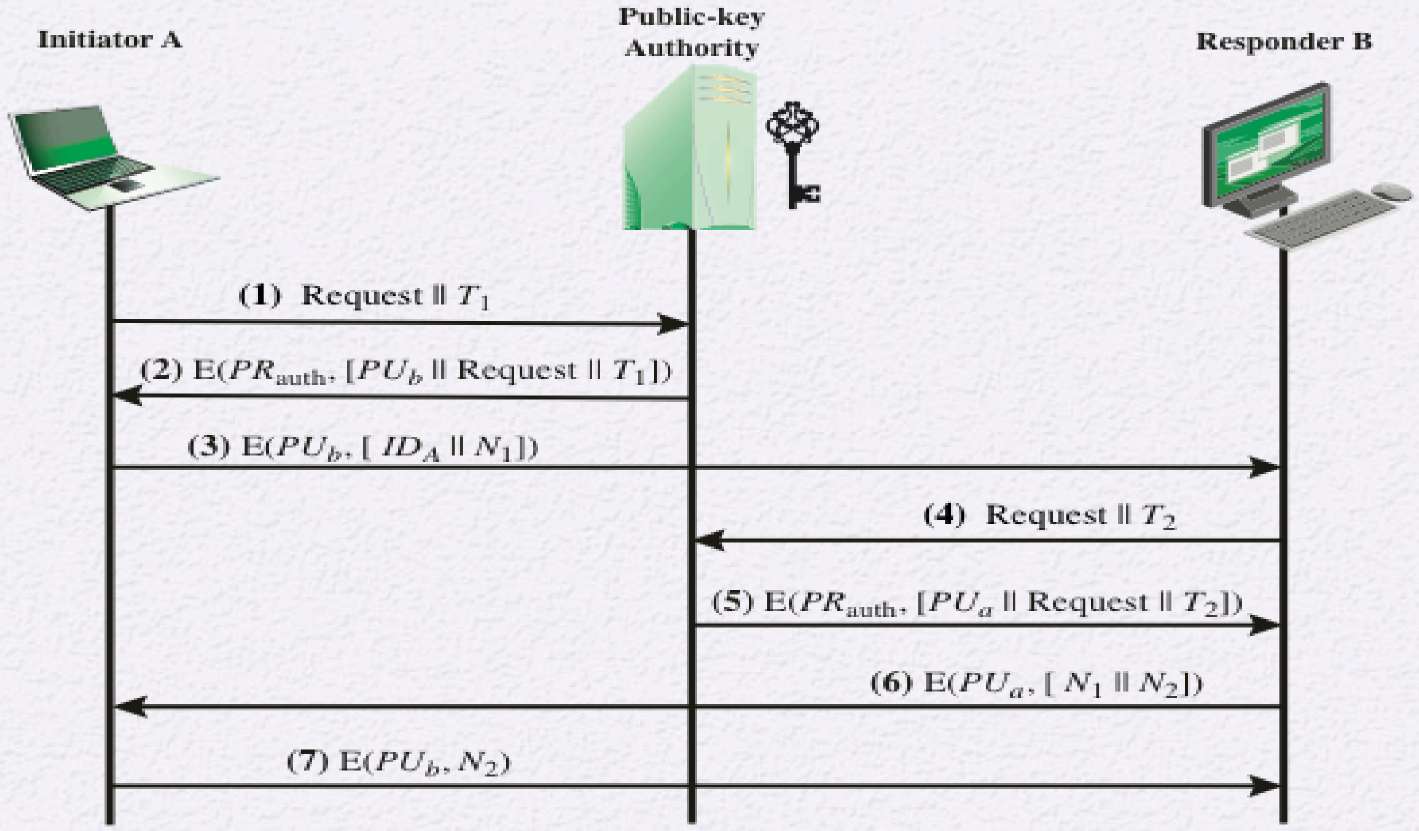


Figure 14.10 Uncontrolled Public Key Distribution

For example, many users have appended their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

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.

Public Key Authority



Initiator A
Public-key Authority
Responder B
(1) Request ‖ T_1
(2) E(PR_{auth}, [PU_b ‖ Request ‖ T_1])
(3) E(PU_b, [ID_A ‖ N_1])
(4) Request ‖ T_2
(5) E(PR_{auth}, [PU_a ‖ Request ‖ T_2])
(6) E(PU_a, [N_1 ‖ N_2])
(7) E(PU_b, N_2)

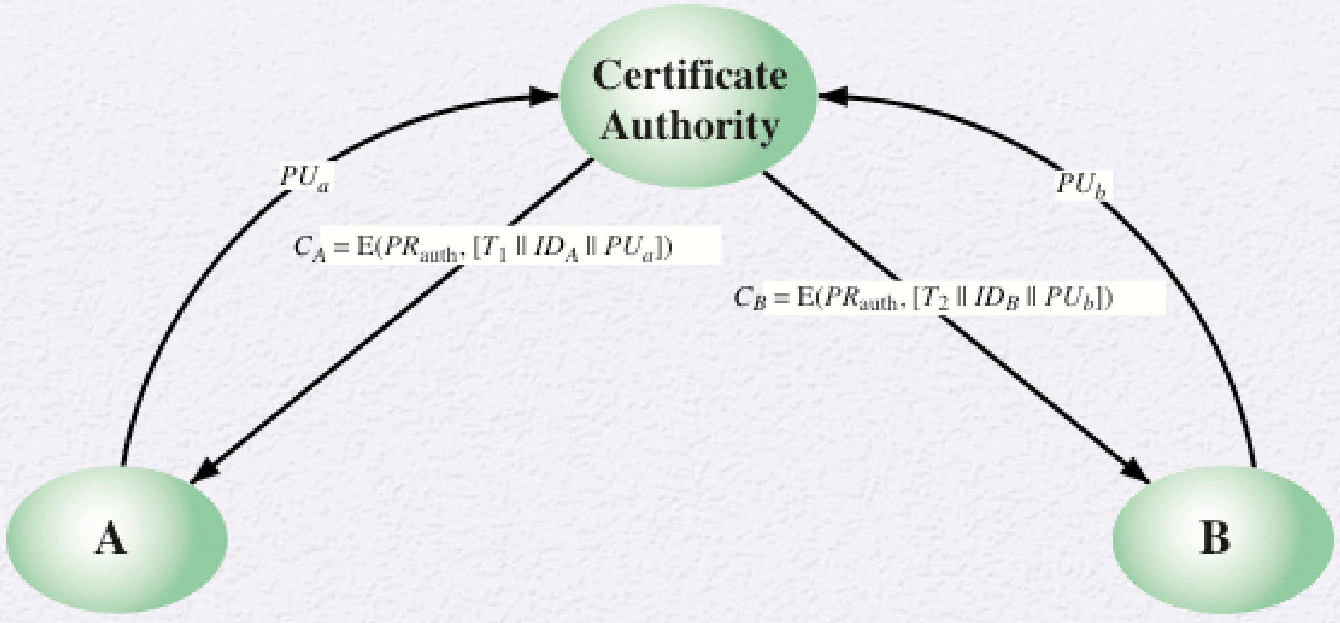
Figure 14.12 Public-Key Distribution Scenario

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 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.

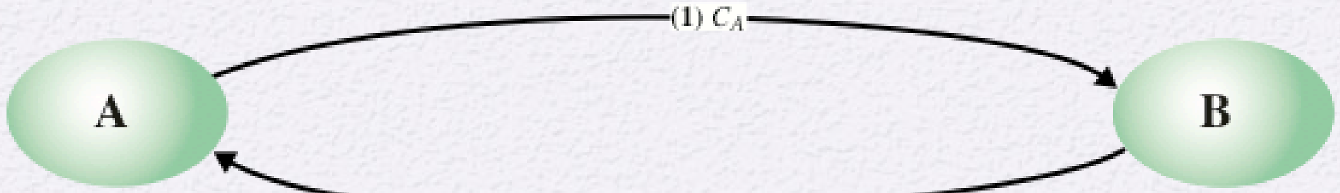
Public Key Certificate

- ➢certificates 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.
- ➢A certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party.
- ➢The third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. Example: Verisign
- ➢A user can present his or her public key to the authority in a secure manner and obtain a 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.

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.


Certificate Authority
PU_a
PU_b
$C_A = \mathrm{E}(PR_{\mathrm{auth}}, [T_1 \| ID_A \| PU_a])$
$C_B = \mathrm{E}(PR_{\mathrm{auth}}, [T_2 \| ID_B \| PU_b])$
A
B

(a) Obtaining certificates from CA

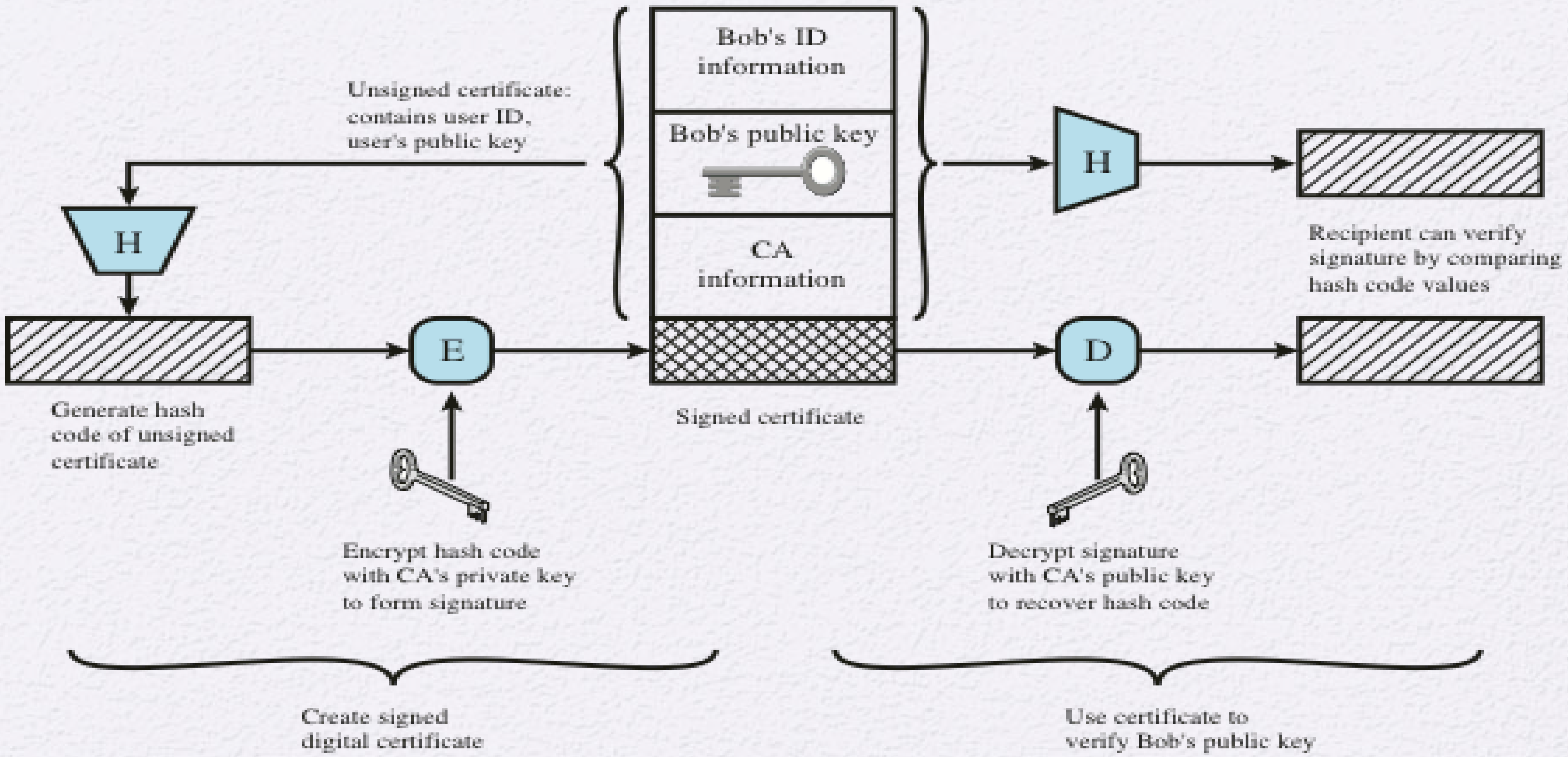


(b) Exchanging certificates

Figure 14.13 Exchange of Public-Key Certificates

X.509 Certificates

- X.509 defines a framework for the provision of authentication services to its users
  - Based on the use of public-key cryptography and digital signatures
  - Does not dictate the use of a specific algorithm but recommends RSA
  - Does not dictate a specific hash algorithm
- Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority
- X.509 defines alternative authentication protocols based on the use of public-key certificates



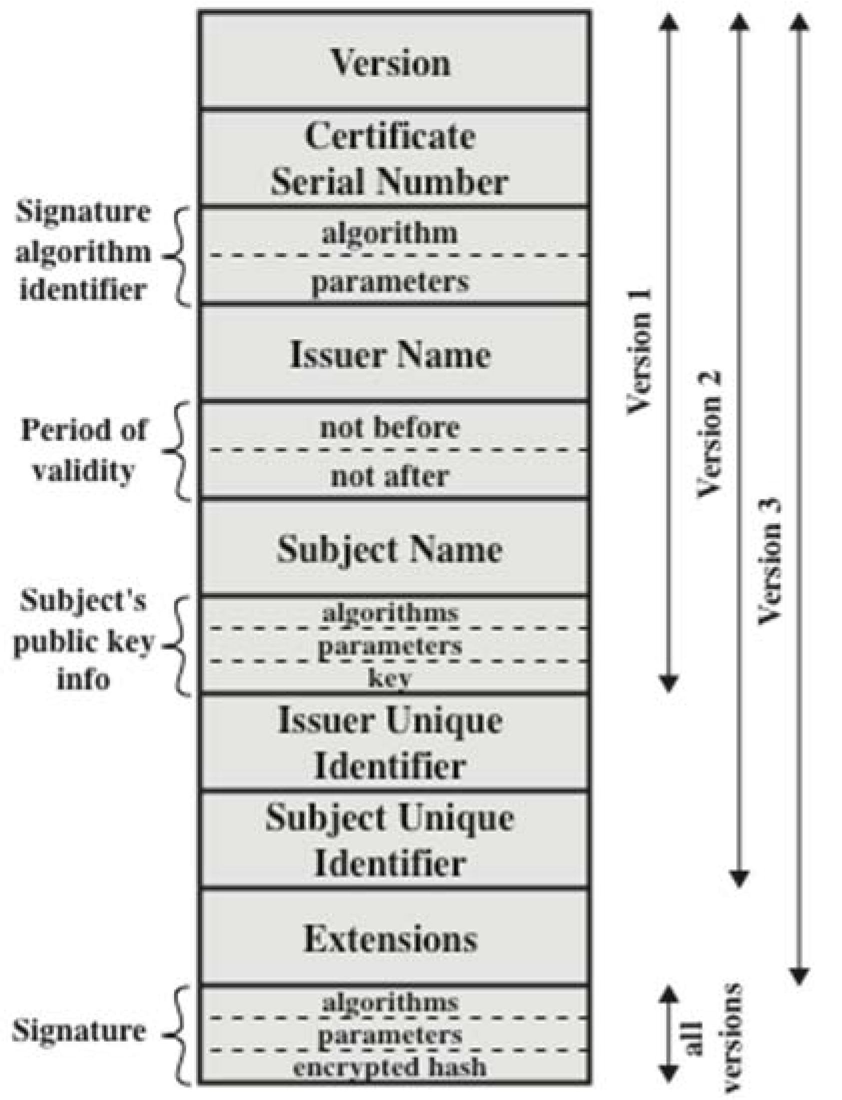
Bob's ID information
Bob's public key
CA information
Unsigned certificate: contains user ID, user's public key
H
H
Recipient can verify signature by comparing hash code values
E
D
Generate hash code of unsigned certificate
Signed certificate
Encrypt hash code with CA's private key to form signature
Decrypt signature with CA's public key to recover hash code
Create signed digital certificate
Use certificate to verify Bob's public key

Figure 14.14 Public-Key Certificate Use

Certificates

Created by a trusted Certification Authority (CA) and have the following elements:

- Version
- Serial number
- Signature algorithm identifier
- Issuer name
- Period of validity
- Subject name
- Subject’s public-key information
- Issuer unique identifier
- Subject unique identifier
- Extensions
- Signature



(a) X.509 Certificate

Figure 14.15 X.509 Formats

Obtaining a Certificate

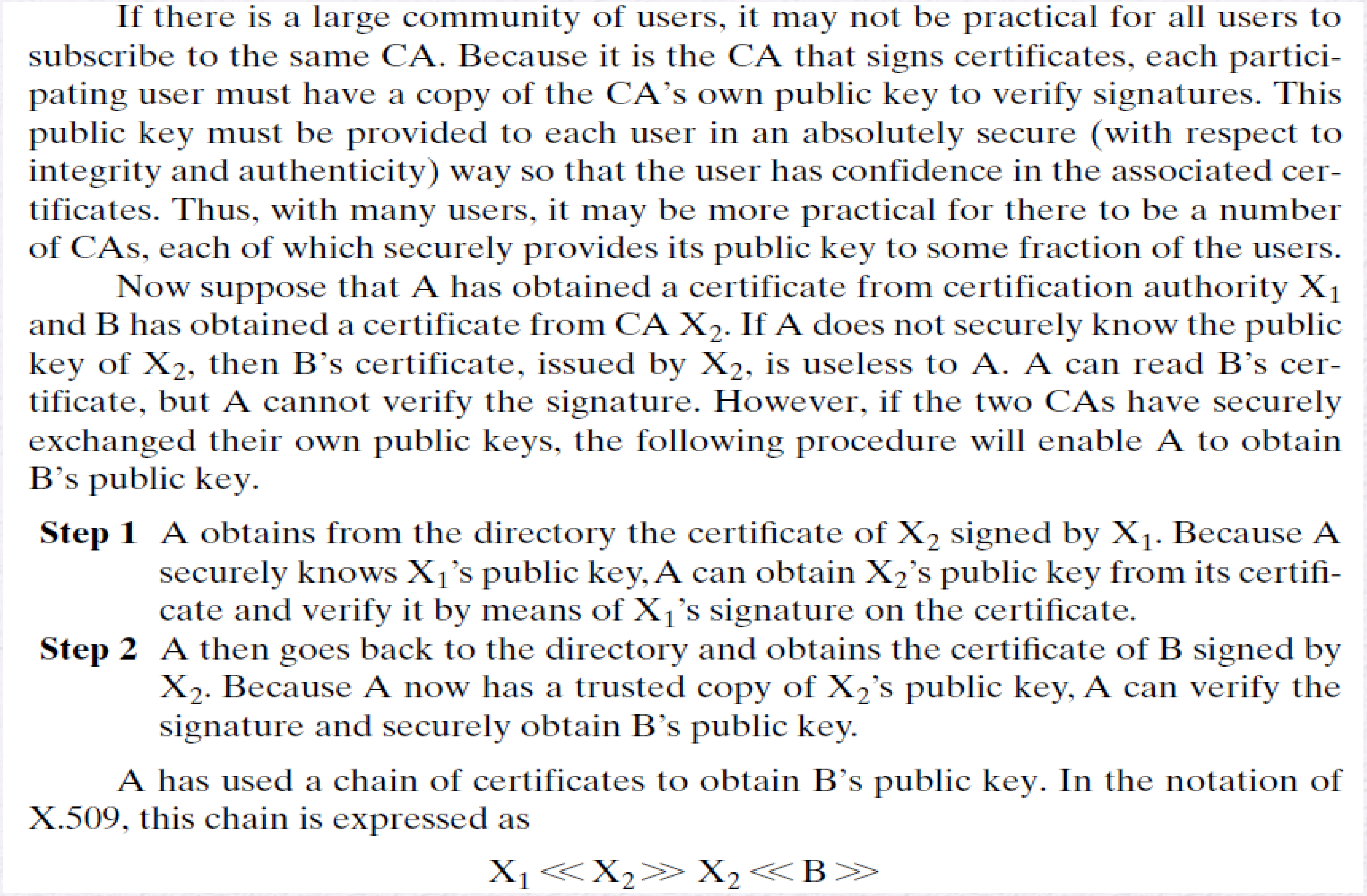
User certificates generated by a CA have the following characteristics:

- 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 without this being detected

- Because certificates are unforgeable, they can be placed in a directory without the need to protect them
  - In addition, a user can transmit his or her certificate directly to other users

- 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 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.

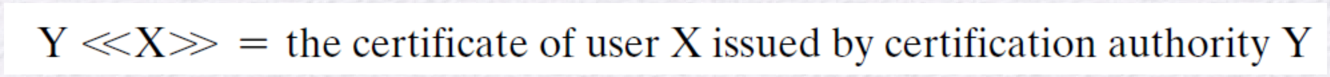
Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, 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

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



$Y \ll X \gg$ = the certificate of user X issued by certification authority Y

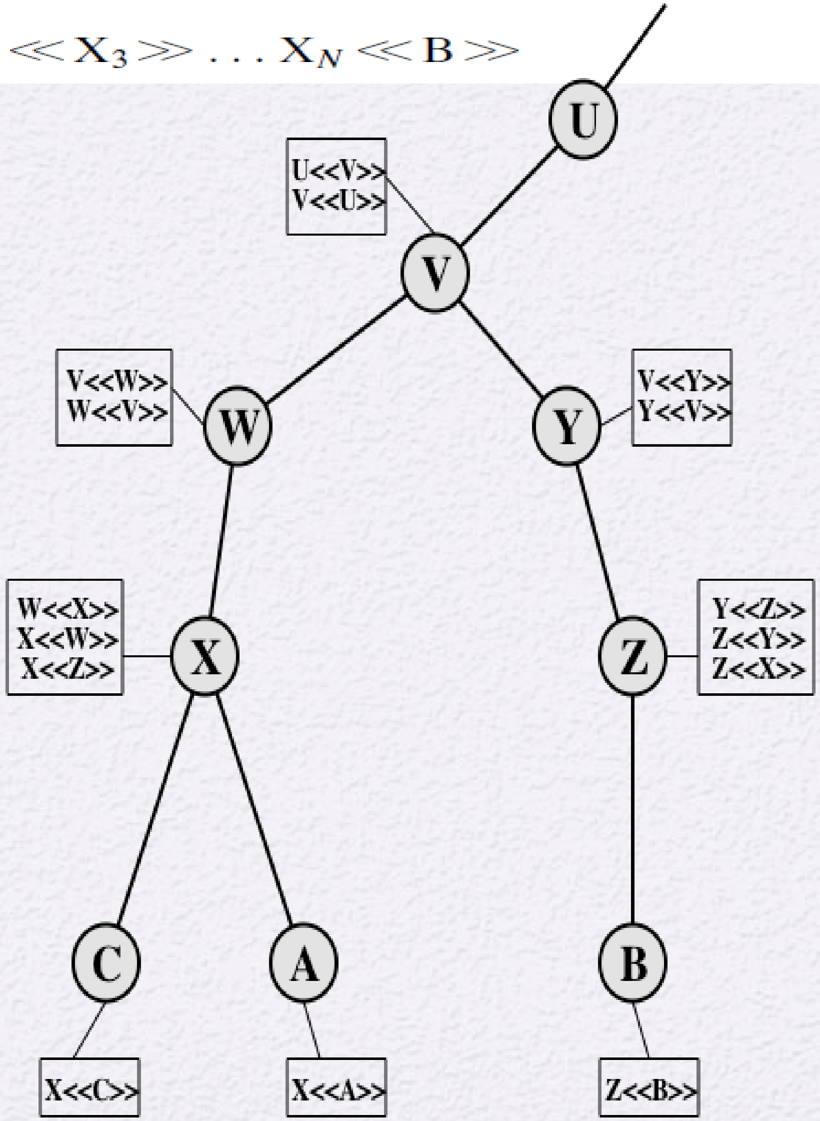
This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as

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

The directory entry for each CA includes two types of certificates:

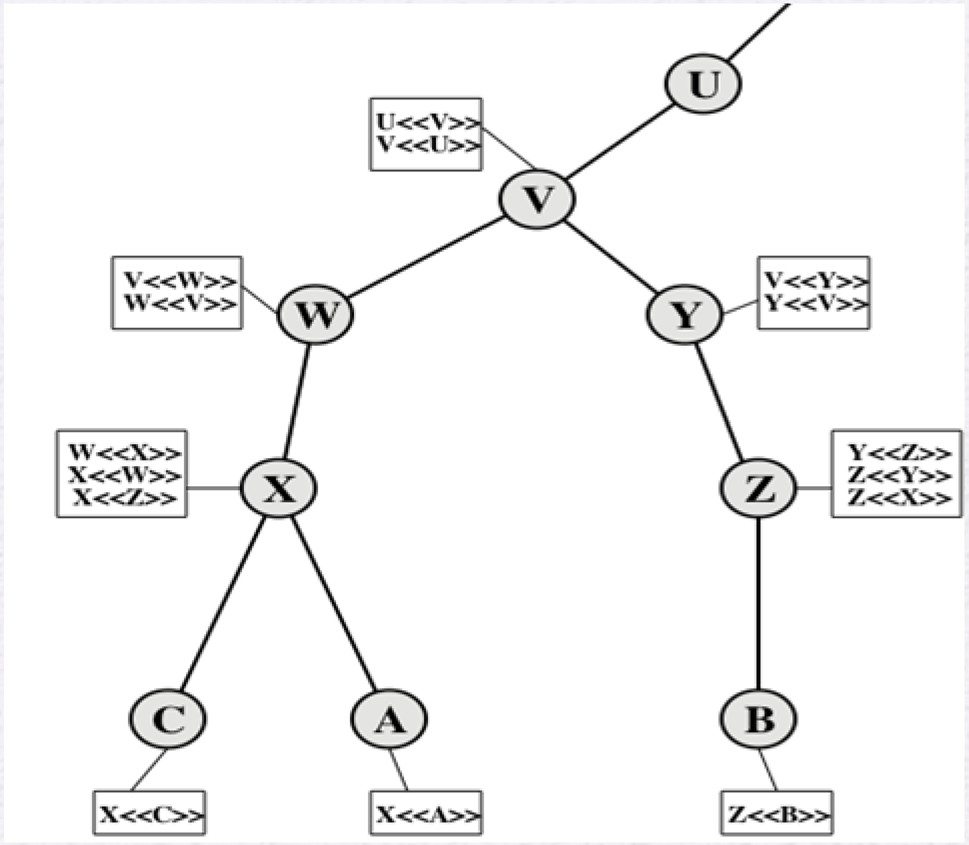
• Forward certificates: Certificates of X generated by other CAs

• Reverse certificates: Certificates generated by X that are the certificates of other CAs



$Y \ll X \gg$ = the certificate of user X issued by certification authority Y

Use CA hierarchy to verify clients certificates.



Y $\ll$X$\gg$ = the certificate of user X issued by certification authority Y

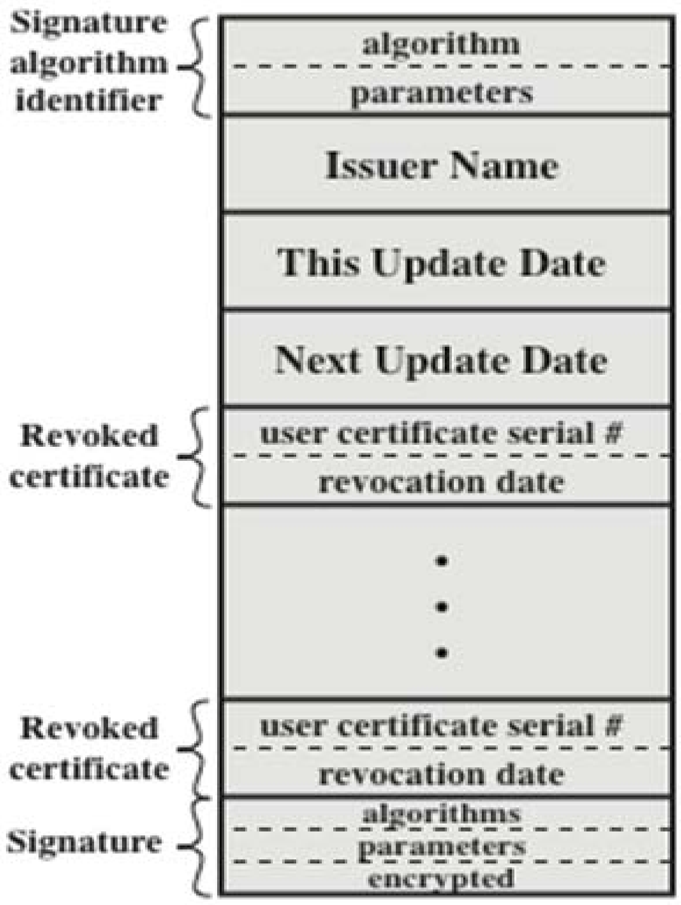
In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

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

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key.

Certificate Revocation

- Each certificate includes a period of validity
    - Typically a new certificate is issued just before the expiration of the old one
- It may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons:
    - The user's private key is assumed to be compromised
    - The user is no longer certified by this CA
- Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA
    - These lists should be posted on the directory



(b) Certificate Revocation List

Summary

- Symmetric key distribution using symmetric encryption
  - Key distribution scenario
  - Hierarchical key control
  - Session key lifetime
  - Transparent key control scheme
  - Decentralized key control
- Symmetric key distribution using asymmetric encryption
  - Simple secret key distribution
  - Secret key distribution with confidentiality and authentication
  - Hybrid scheme

- Distribution of public keys
  - Public announcement of public keys
  - Public-key authority
  - Public-key certificates
- X.509 Certificates