Chapter 14 :

Key Management and Distribution

- 14.1Symmetric Key Distribution Using Symmetric Encryption
- 14.2 Symmetric Key Distribution Using Asymmetric Encryption
- 14.3 Distribution of Public Keys
- 14.4 X.509 Certificates

Chapter 15 :

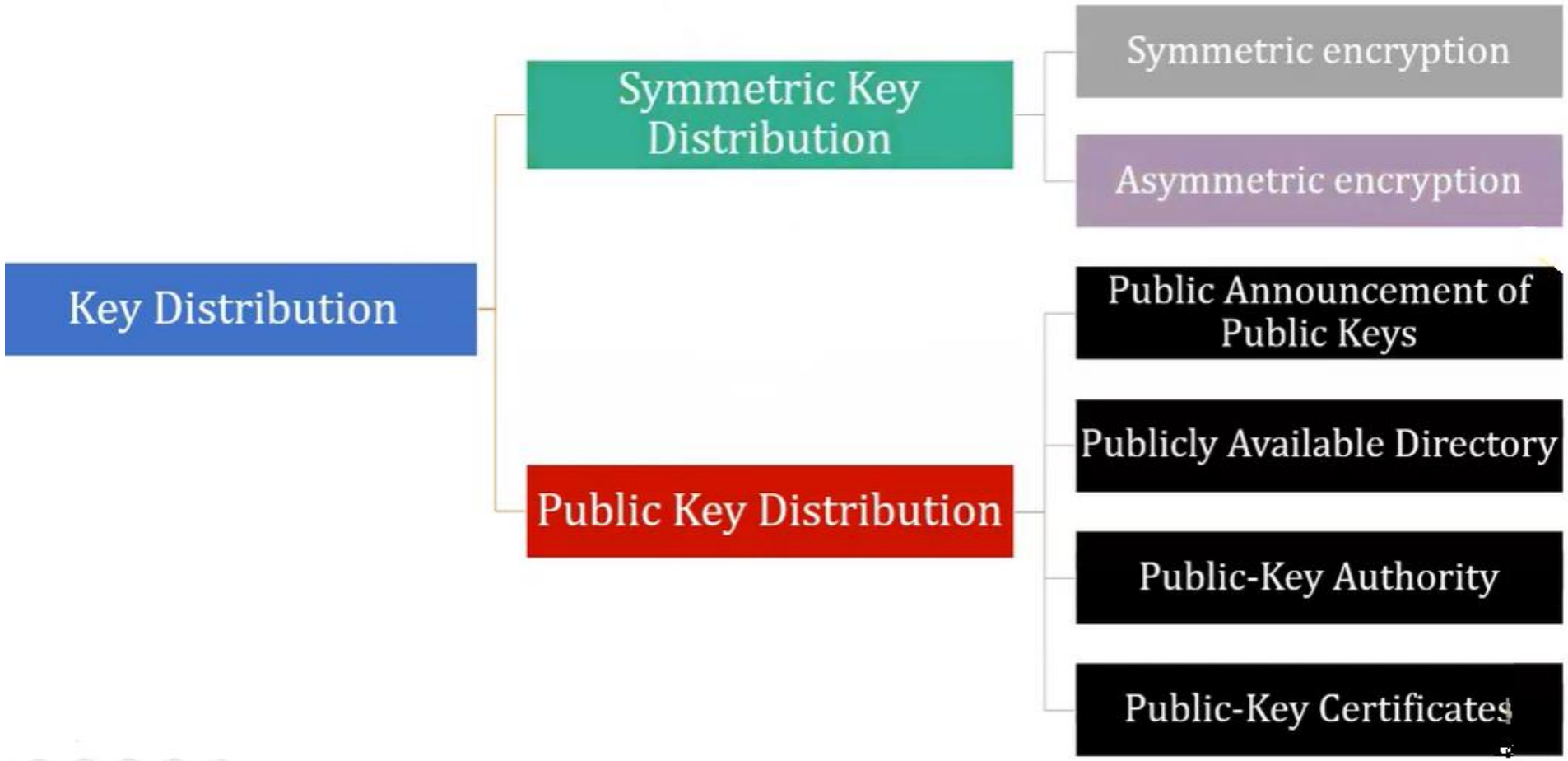
User Authentication

- 15.2Remote User-Authentication Using Symmetric Encryption
- 15.3 Kerberos
- 15.4 Remote User-Authentication Using Asymmetric Encryption

Chapter 17 :

Web Security Considerations

Key Distribution



Symmetric Key Distribution Using Symmetric Encryption

- **When 2 parties share the same key (i.e., symmetric key) that protects from access by others, the process between 2 parties that exchanges the key called as symmetric key distribution**

- 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. – difficult process in WAN
- 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
- 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.
- 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, such as a frame relay connection or transport connection, and then discarded.
- 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.

- For each end system or user, there is a unique master key that it shares with the key distribution center.
- 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.

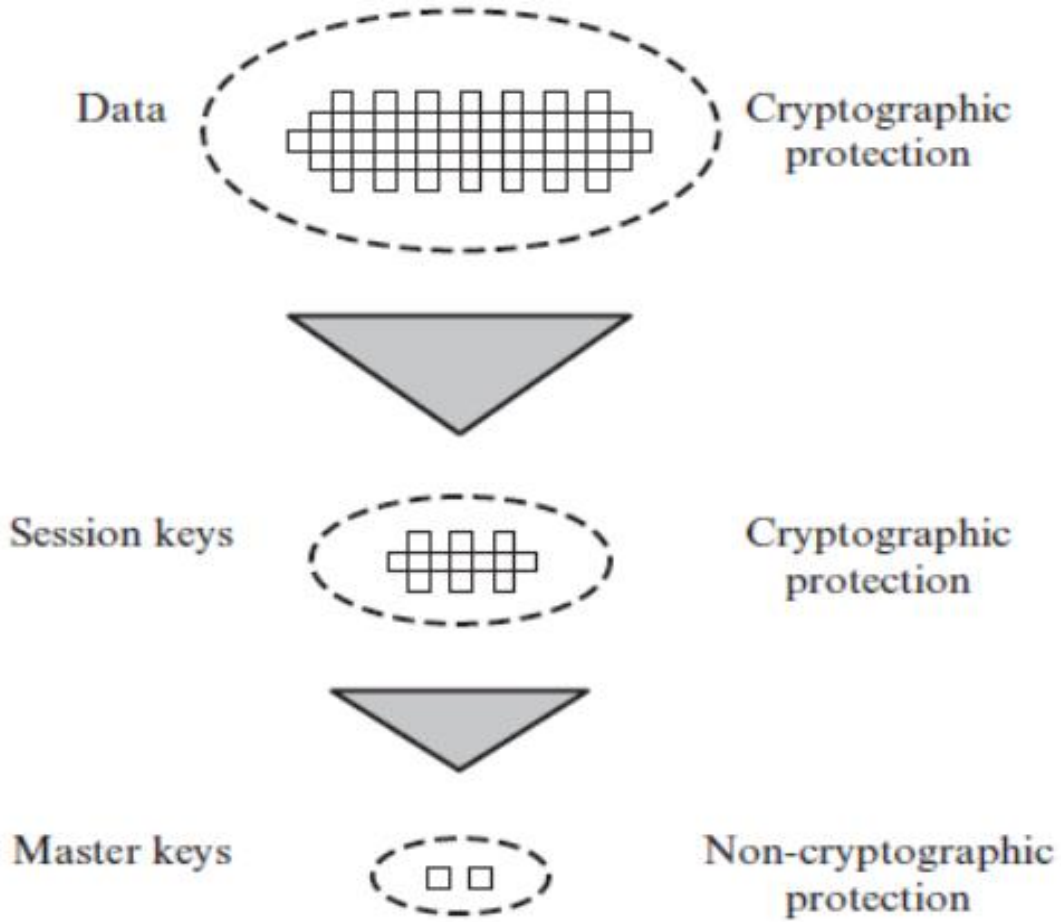


Figure 14.2 The Use of a Key Hierarchy

A Key Distribution Scenario

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

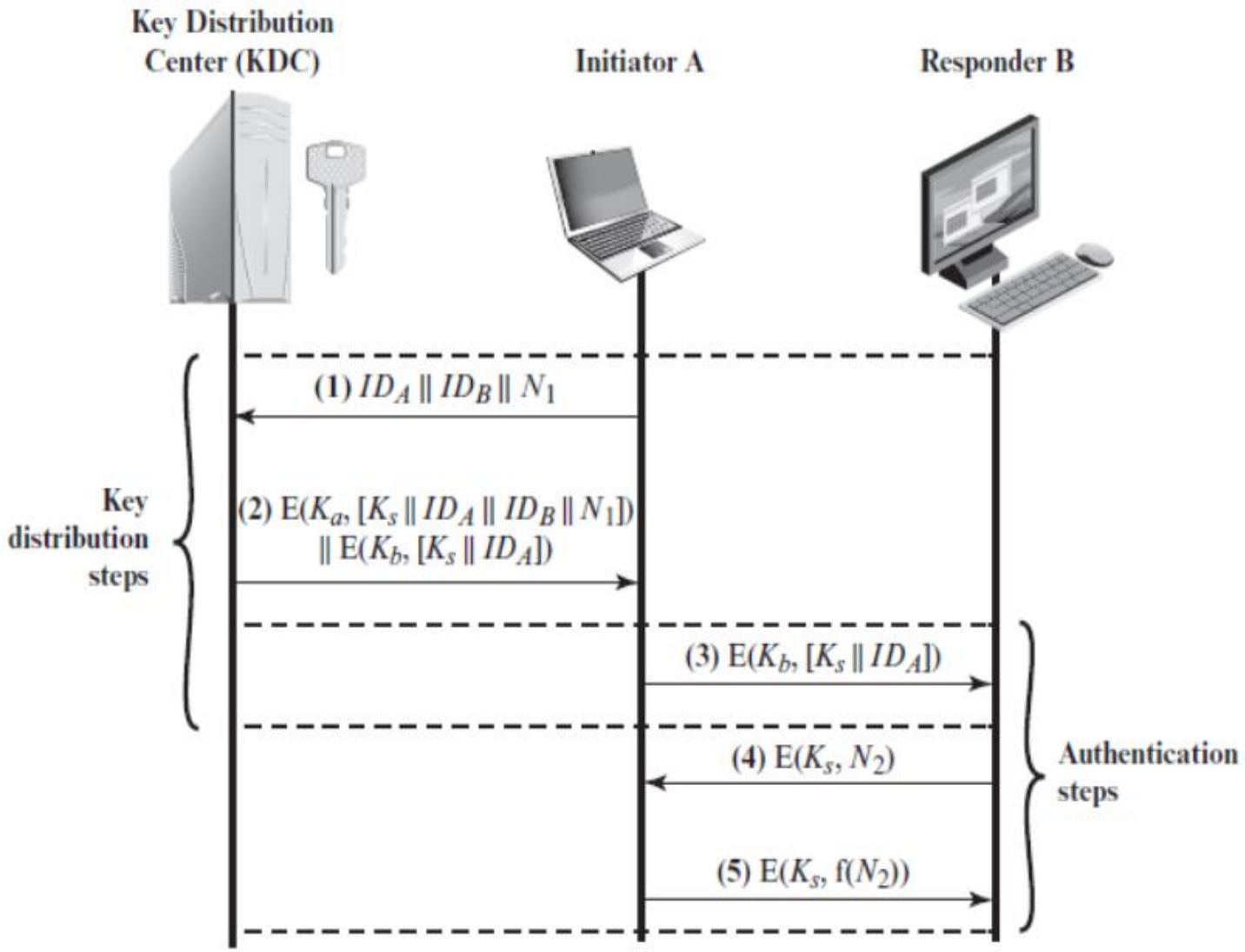


Figure 14.3 Key Distribution Scenario

The following steps occur.

Step 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, referred 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.
- To prevent attack, it should be difficult for an opponent to guess the nonce.
- Thus, a **random number is a good choice for a nonce.**

Step 2 : The KDC responds with a message encrypted using Ka

- A is the only one who can successfully read the message
- 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
- 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*

Step 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(Kb,[Ks }IDA]).

- B now knows the session key (*Ks*),
- knows that the other party is A (from *IDA*), and
- knows that the information originated at the KDC

Step 4: Using the newly minted session key for encryption, B sends a nonce, N2, to A.

Step 5 : Also, using Ks, A responds with f(N2), where f is a function that performs some transformation on N2 (e.g., adding one).

Hierarchical Key Control

- It is not necessary to limit the key distribution function to a single KDC.
- For very large networks, 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.
- 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 KDCs can communicate through a **global KDC**
- In this case, any one of the three KDCs involved can actually select the key
- 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.

Advatanges :

- A hierarchical scheme minimizes the effort involved in master key distribution, because most master keys are shared by a local KDC with its local entities.
- limits the damage of a faulty or subverted KDC to its local area only.

Session Key Lifetime

- The more frequently session keys are exchanged, the more secure they are because the opponent has less ciphertext to work with for any given session key.
- 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 necessary to change the session key periodically,
- 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. – **increases overhead and delay**
- 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

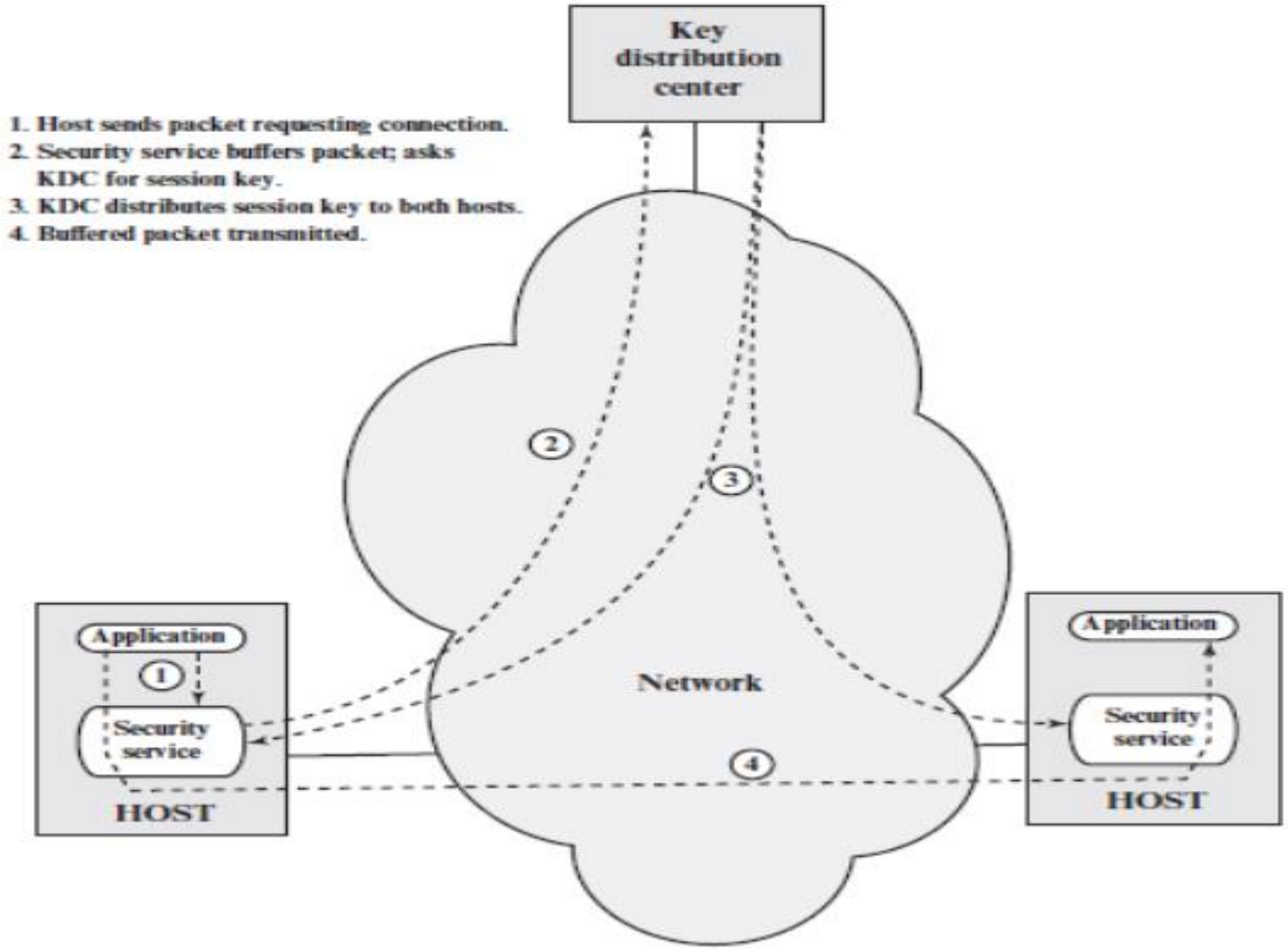


Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

- The scheme is 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.**
- **Session Security Module (SSM),** that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

Step 1 : When one host wishes to set up a connection to another host, it transmits a connection request packet to SSM

Step 2 : The SSM saves that packet and applies to the KDC for permission to establish the connection

- The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC.

Step 3 : 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

Step 4 : The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems

- All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

Decentralized Key Control

- Key distribution is fully decentralized. **Uses shared master key**
- Each node must maintain **at most (n - 1) master keys**
- **Session key is exchanged without the help of KDC**
- full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context

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, N_1.
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.

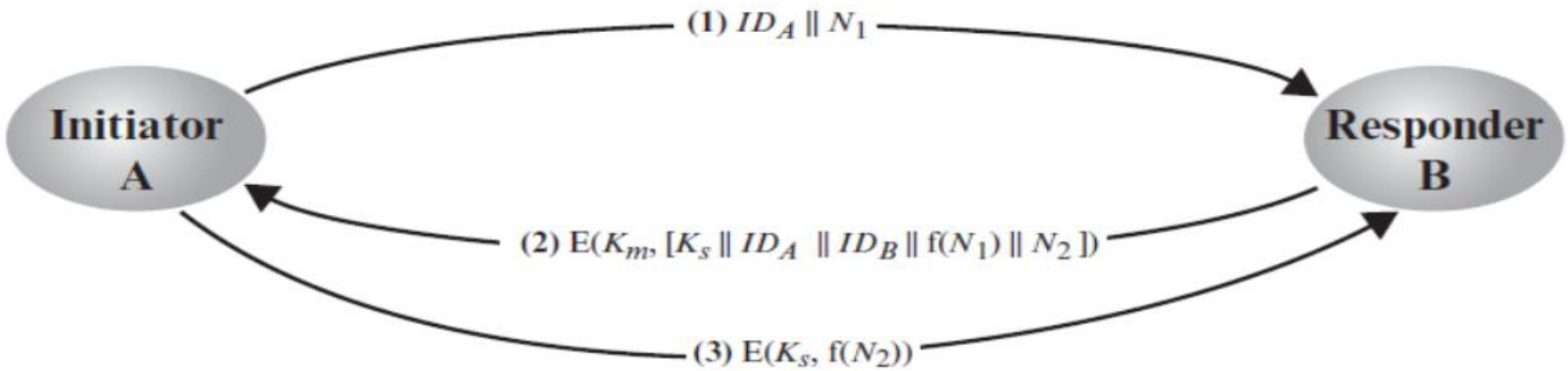


Figure 14.5 Decentralized Key Distribution

Controlling Key Usage

- The concept of a **key hierarchy** and the use of **automated key distribution** techniques greatly reduce the number of keys that must be manually managed and distributed.

Different types of session keys

- **Data-encrypting key** : for general communication across a network
- **PIN-encrypting key** : for personal identification numbers (PINs) used in electronic funds transfer.
- **File-encrypting key** : encrypting files stored in publicly accessible locations

- If a master key is treated as a session key, it may be possible for an unauthorized application to obtain plaintext of session keys encrypted with that master key.
- Thus, Limit the ways in which keys are used, based on characteristics associated with those keys.
- **Solution : associate a tag with each key**
- The proposed technique is for use with DES and makes use of the extra 8 bits in each 64-bit DES key
  - One bit indicates whether the key is a session key or a master key
  - One bit indicates whether the key can be used for encryption
  - One bit indicates whether the key can be used for decryption
  - The remaining bits are spares for future use

Advantage :

- Tag is embedded in the key, it is encrypted along with the key when that key is distributed, thus providing protection.

Drawbacks :

- The tag length is limited to 8 bits, limiting its flexibility and functionality.
- Because the tag is not transmitted in clear form, it can be used only at the point of decryption, limiting the ways in which key use can be controlled.

The control vector is cryptographically coupled with the key at the time of key generation at the KDC

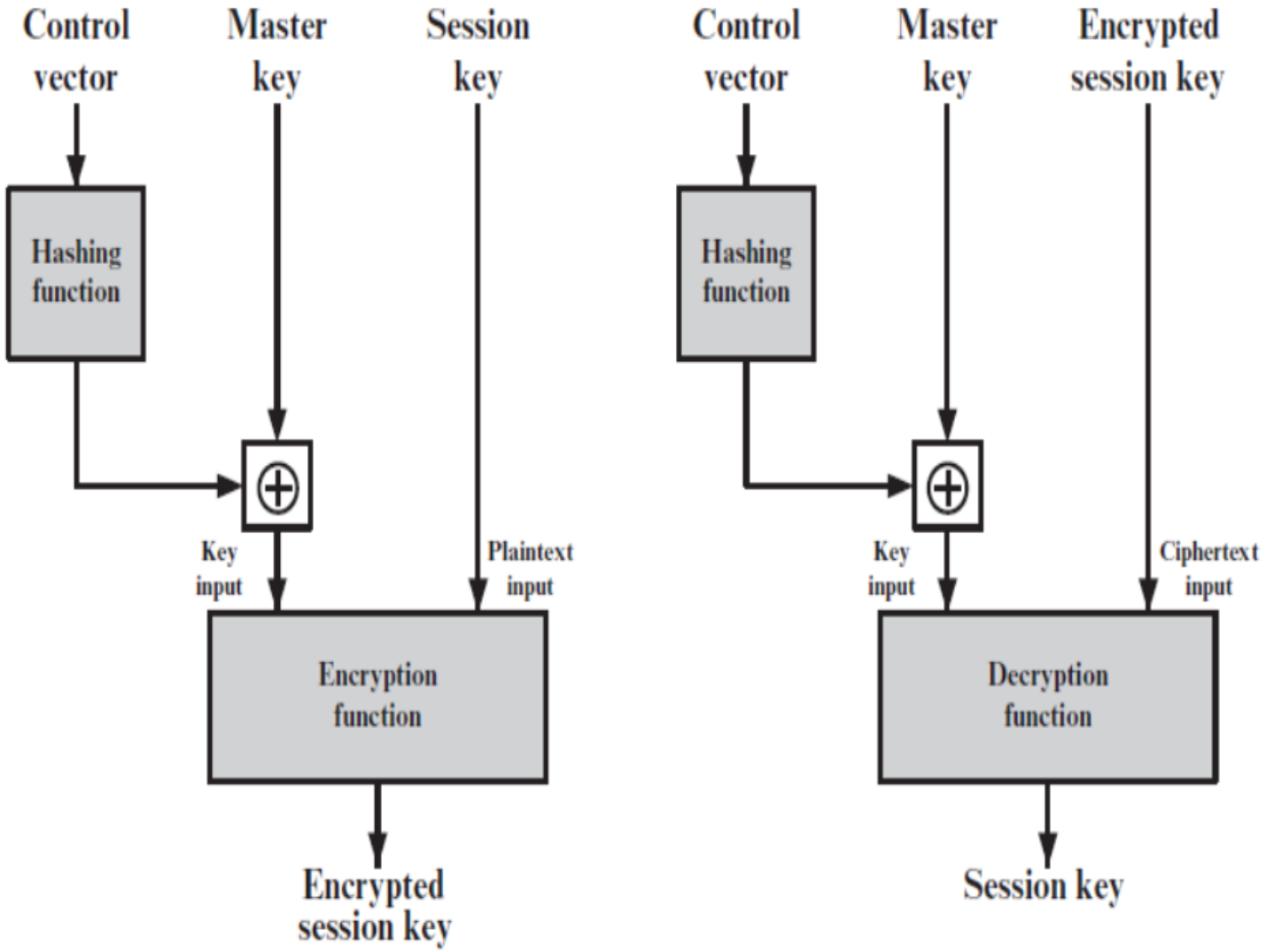


Figure 14.6 Control Vector Encryption and Decryption

The hash value is then XORed with the master key to produce an output that is used as the key input for encrypting the session key. Thus,

$$\text{Hash value} = H = \text{h(CV)}$$
$$\text{Key input} = K_m \oplus H$$
$$\text{Ciphertext} = \text{E}([K_m \oplus H], K_s)$$

where K_m is the master key and K_s is the session key. The session key is recovered in plaintext by the reverse operation:

$$\text{D}([K_m \oplus H], \text{E}([K_m \oplus H], K_s))$$

When a session key is delivered to a user from the KDC, it is accompanied by the control vector in clear form. The session key can be recovered only by using both the master key that the user shares with the KDC and the control vector. Thus, the linkage between the session key and its control vector is maintained.

Use of the control vector has two advantages

1. There is no restriction on length of the control vector
2. The control vector is available in form at all stages of operation.

Symmetric Key Distribution Using Asymmetric Encryption

There are two approaches:

1. Simple Secret Key Distribution
2. Secret Key Distribution with Confidentiality and Authentication

1. Simple Secret Key Distribution

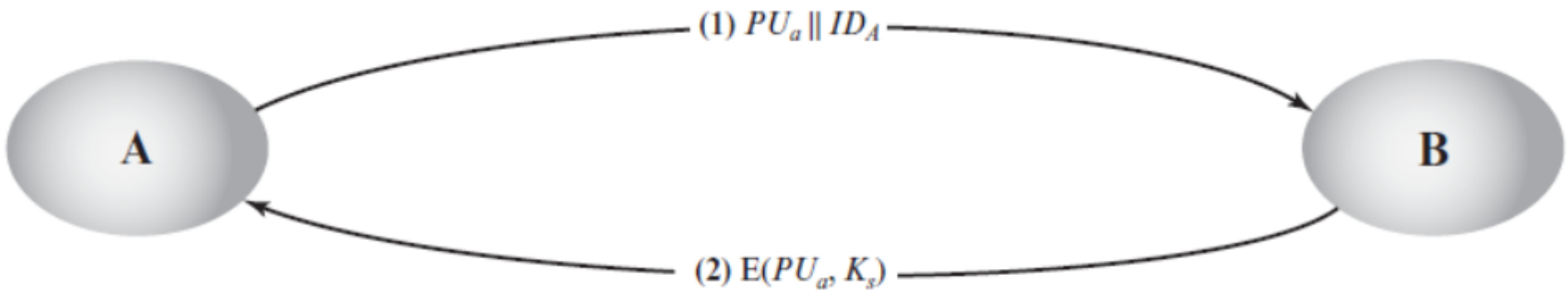


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

- 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 *Ks*. At the completion of the exchange, both A and B discard *Ks*.
- 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 is insecure against an adversary who 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**

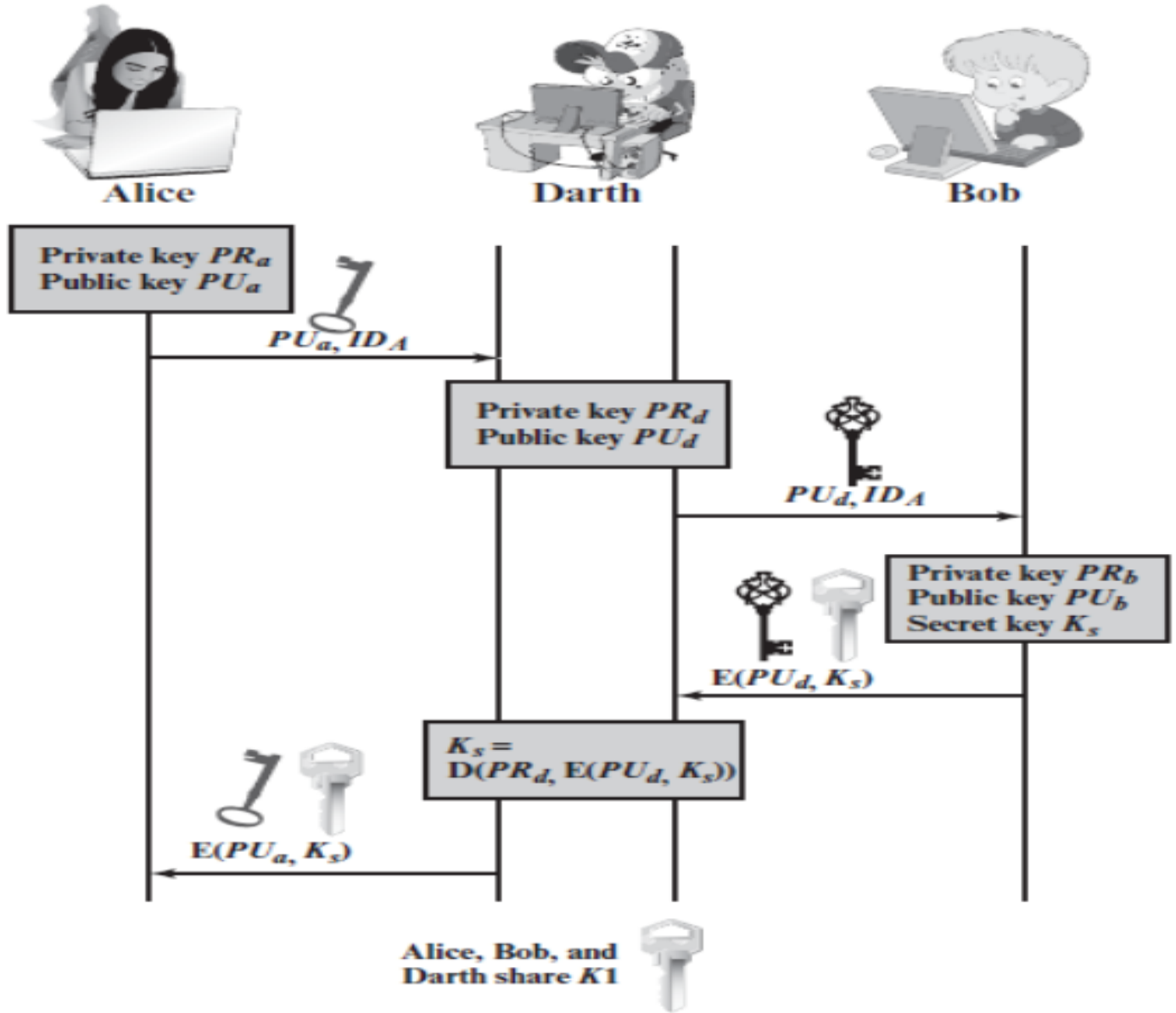


Figure 14.8 Another Man-in-the-Middle Attack

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.

2. Secret Key Distribution with Confidentiality and Authentication

- Provides protection against both active and passive attacks.

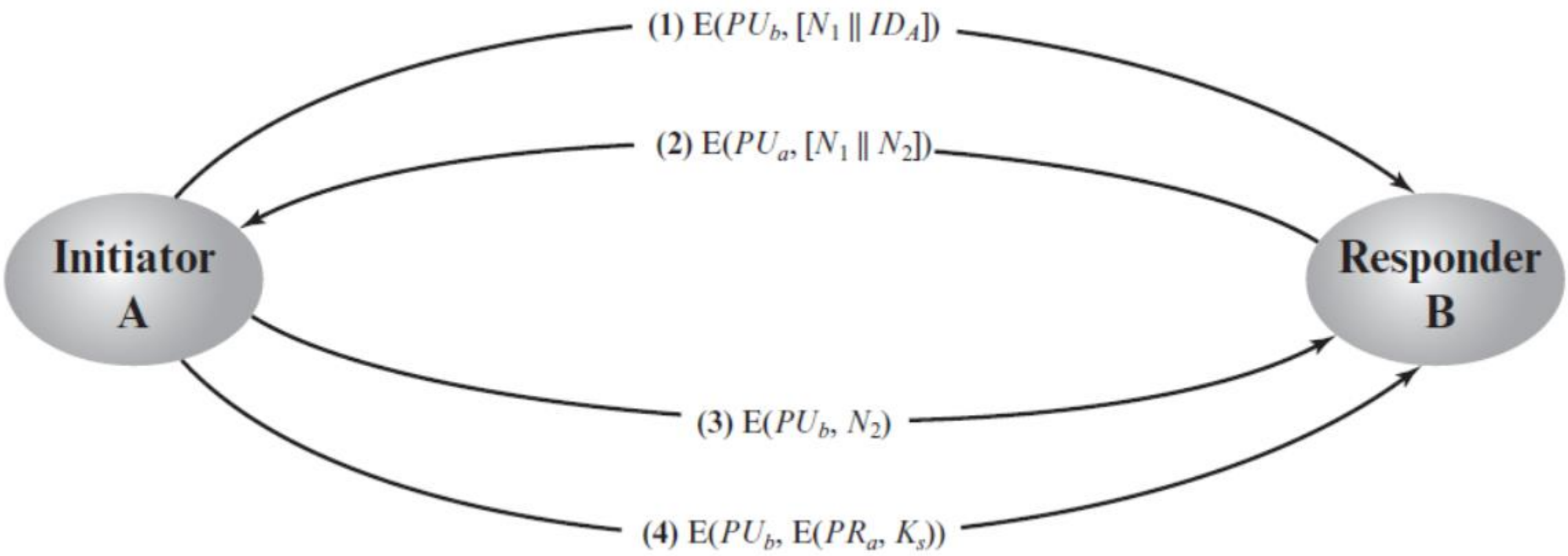


Figure 14.9 Public-Key Distribution of Secret Keys

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 $M = \mathrm{E}(PU_b, \mathrm{E}(PR_a, K_s))$ 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 $\mathrm{D}(PU_a, \mathrm{D}(PR_b, M))$ to recover the secret key.

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

A Hybrid Scheme

- Yet another way to use **public-key encryption to distribute secret keys is a hybrid approach** in use on IBM mainframes
- This scheme 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.

Performance:

- There are many applications, especially transaction-oriented applications, in which the session keys change frequently.
- Distribution of session keys by public-key encryption could degrade overall system performance because of the relatively high computational load of public-key encryption and decryption.
- With a three-level hierarchy, public-key encryption is used only occasionally to update the master key between a user and the KDC.

Backward compatibility:

- The hybrid scheme is easily overlaid on an existing KDC scheme with minimal disruption or software changes.

The addition of a public-key layer provides a **secure**, **efficient** means of distributing master keys.

This is an advantage in a configuration in which a **single KDC serves a widely distributed set of users**.

DISTRIBUTION OF PUBLIC KEYS

Several techniques have been proposed for the distribution of public keys.

1. **Public announcement**
2. **Publicly available directory**
3. **Public-key authority**
4. **Public-key certificates**

1. Public Announcement of Public Keys :

Participant can send his or her public key to **any other participant** or **broadcast** the key to the community at large

Figure 14.10 Uncontrolled Public-Key Distribution

- **Weakness :**
  - Anyone can forge such a public announcement.
  - 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.
  - forged keys can also be used for authentication

2. 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
- 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 authenticate communication.

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.

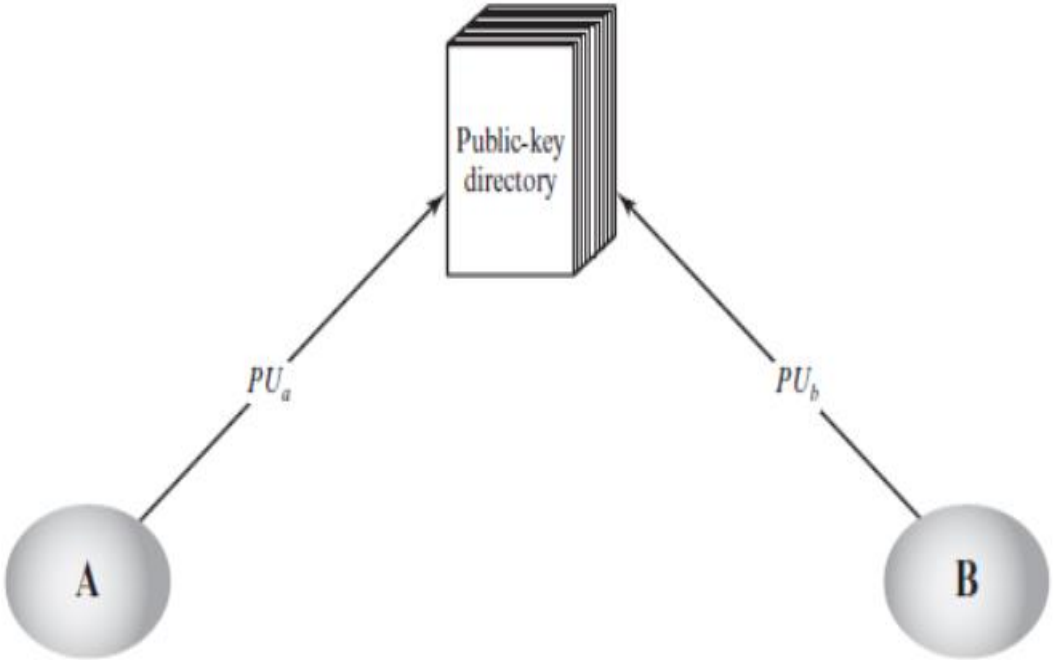


Figure 14.11 Public-Key Publication

- 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,
  - Eavesdrop on messages sent to any participant.
  - counterfeit (imitate) public keys
  - impersonate any participant
- Adversary can tamper the records kept by the authority.

3. 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.**
- Central authority maintains a dynamic directory of public keys of all participants.
- Each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key

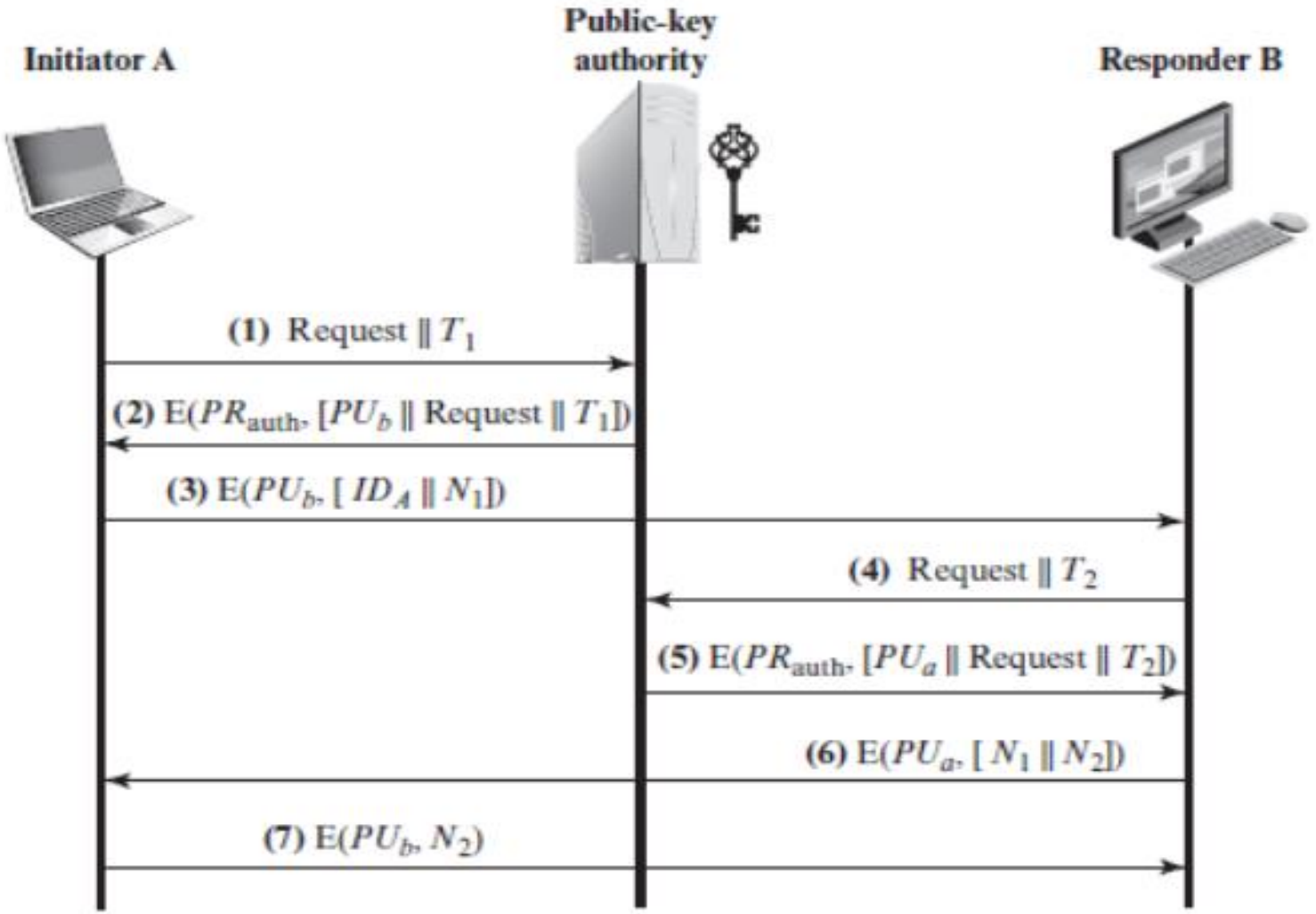


Figure 14.12 Public-Key Distribution Scenario

Drawbacks:

- The public-key authority could be somewhat of a bottleneck in the system
- user must appeal to the authority for a public key for every other user that it wishes to contact.
- The directory of names and public keys maintained by the authority is vulnerable to tampering.

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, *PUb*, 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 uses it to encrypt a message to B containing an identifier of A (*IDA*) 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 *PUa* 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.

4. Public-Key Certificates

- **Certificates** can be used by participants to **exchange keys without contacting a public-key authority.**
- Reliable as if the keys were obtained directly from a public-key authority.
- A certificate consists
  - A public key,
  - An identifier of the key owner
- The whole block signed by a trusted third party. (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.

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.

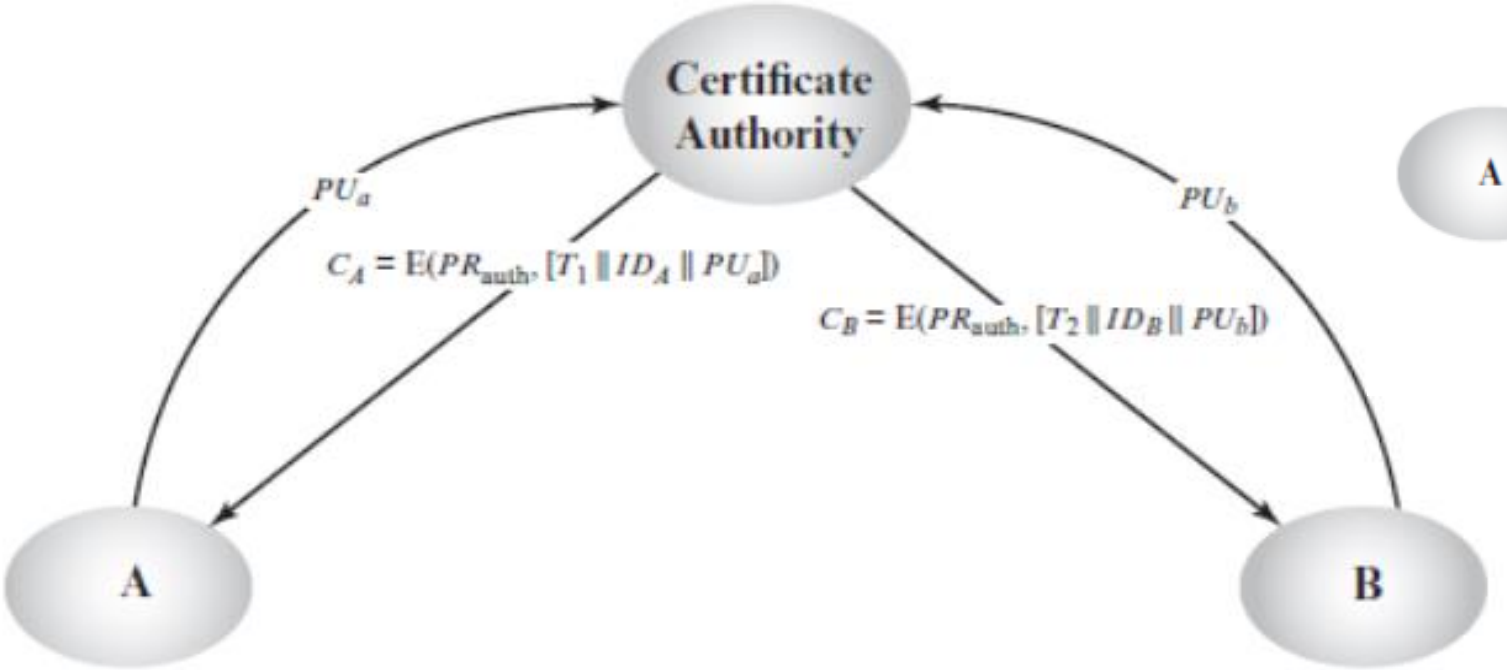
3. Only the certificate authority can create and update certificates.

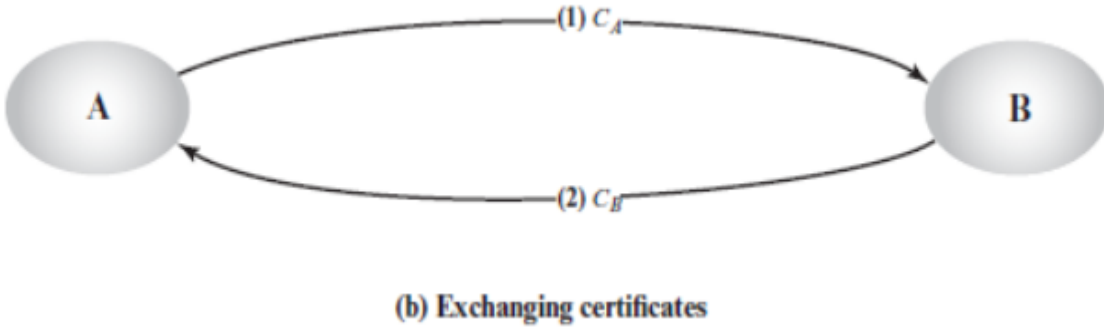
4. Any participant can verify the time validity of the certificate.

- Each participant applies to the certificate authority, supplying a public key and requesting a certificate.

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

who reads and verifies the





(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** by the X.500 directory to its users.
- 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.
- X.509 is an important standard because the **certificate structure and authentication protocol**s defined in X.509 are used in a variety of contexts.
- X.509 certificate format is used in
  - S/MIME
  - IP Security
  - SSL/TLS
- X.509 is based on the use of public-key cryptography and digital signatures.

X.509 scheme for generation of a public-key certificate.

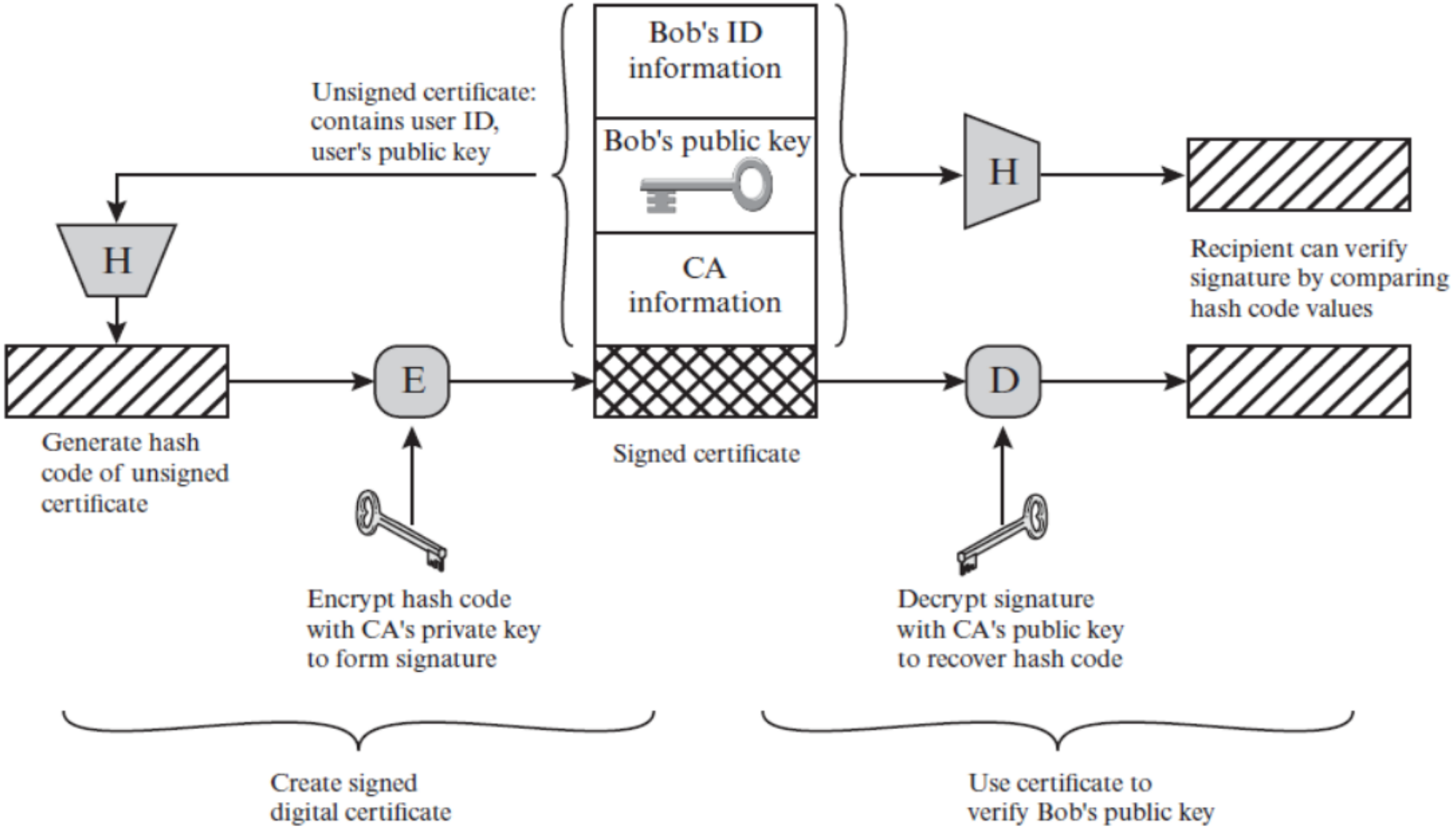


Figure 14.14 X.509 Public-Key Certificate Use

Certificates :

- user certificates are assumed to be created by some trusted **certification authority** (CA).
- These certificates are 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.

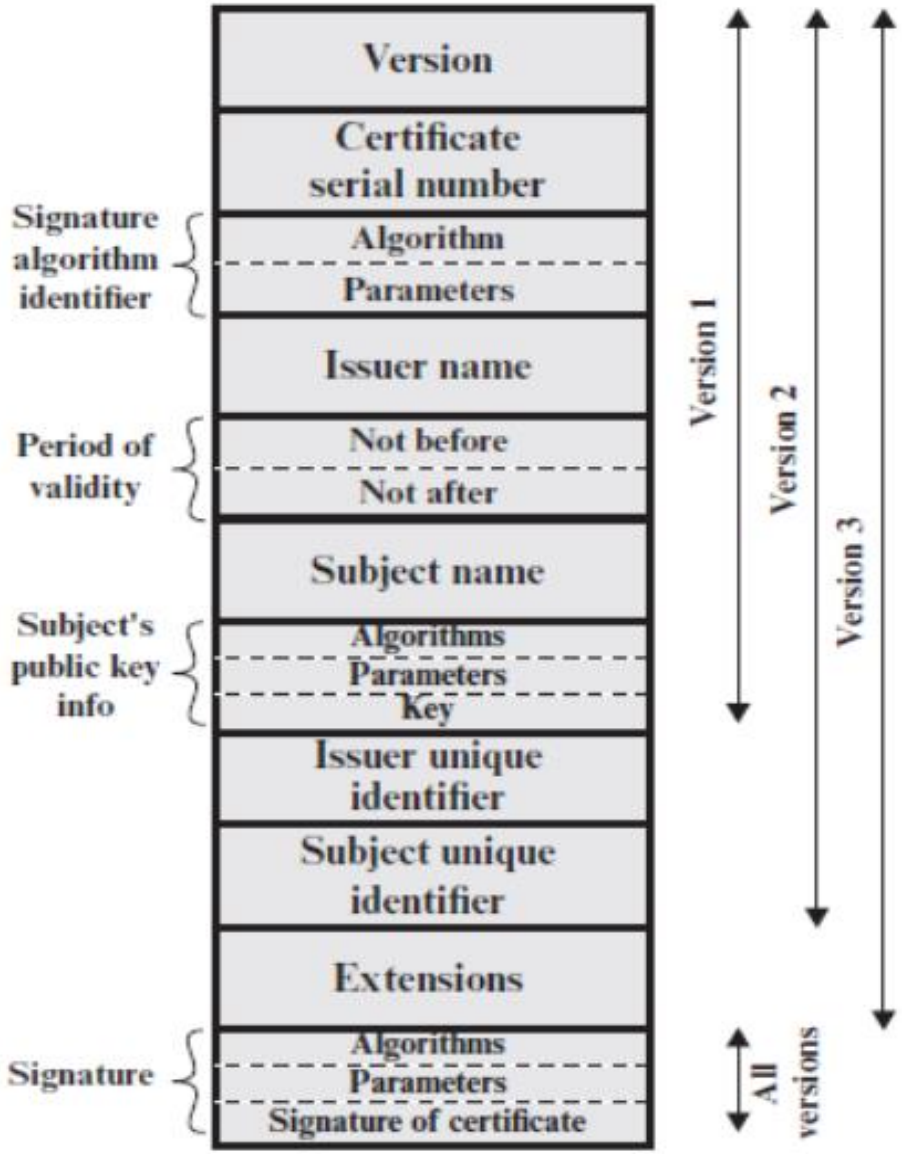
General format of a certificate includes the following elements.

1. Version:

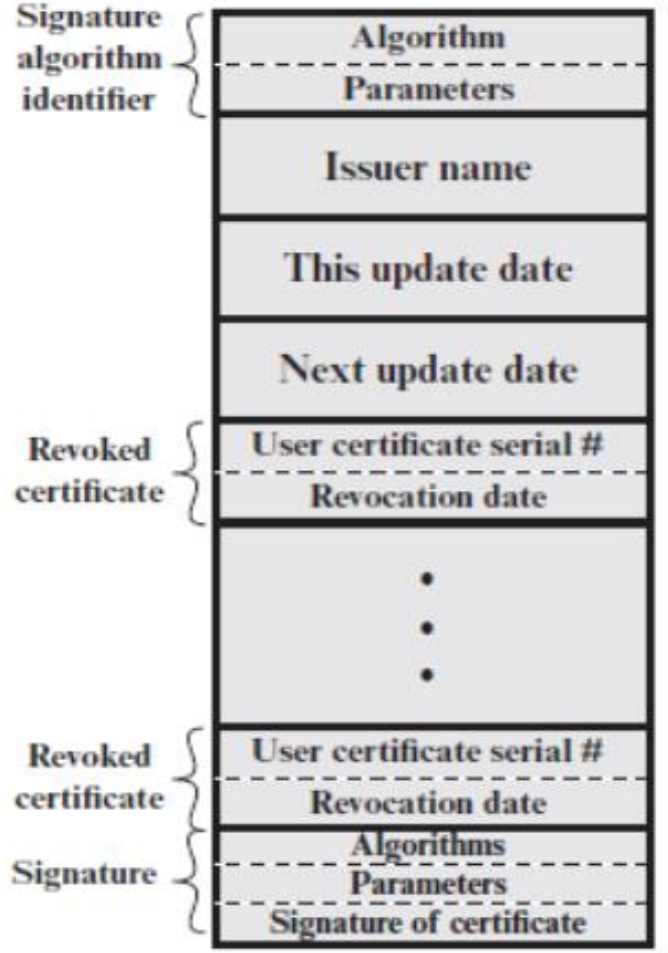
- Differentiates among successive versions of the certificate format. default is version 1
- If the *issuer unique identifier* or *subject unique identifier* are present - version 2
- If one or more extensions are present – version 3
- Current version is 7. No more changes are done to the field after version 3.

2. Serial number: Unique number associated with the certificate

3. Signature algorithm identifier: The algorithm **used to sign the certificate** together with any associated parameters.



(a) X.509 certificate



(b) Certificate revocation list

Figure 14.15 X.509 Formats

4. Issuer name: X.500 name of the CA that created and signed this certificate.

5. Period of validity: Consists of two dates: the first and last on which the certificate is valid.

6. 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.

7. 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.

8. 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.

9. 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.

10. Extensions: A set of one or more extension fields. Extensions were added in version 3

11. 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 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.

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

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

- Small group of people : common CA can be maintained
- Large group of people : Multiple CA's can be used.

Scenario :

- A has obtained a certificate from certification authority X1
- B has obtained a certificate from CA X2.
- If A does not securely know the public key of X2, then B's certificate, issued by X2, is useless to A. A can read B's certificate, but A cannot verify the signature.

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

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

Each CA includes two types of certificates

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

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

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

- Using B's public key, A can send encrypted messages to B.
- If B requires A's public key, which can be obtained from the following certification path:

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

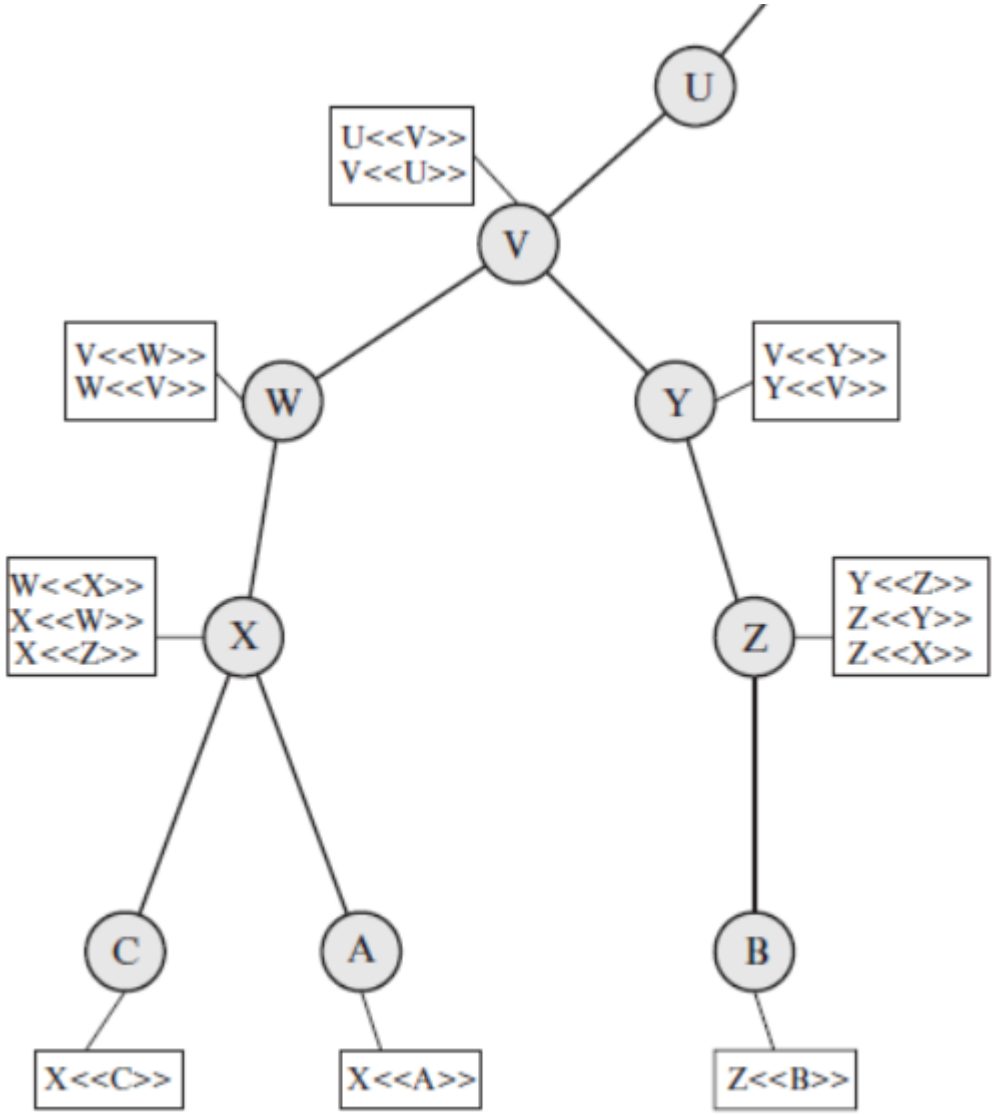


Figure 14.16 X.509 Hierarchy: A Hypothetical Example

REVOCATION OF CERTIFICATES :

- Each certificate includes a period of validity, much like a credit card.
- A new certificate is issued just before the expiration of the old one.
- 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 **certificate revocation list (CRL)** posted to the directory is signed by the issuer.
- Includes 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.
- Serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

X.509 Version 3 :

Requirements not satisfied by version 2.

- The subject field is inadequate to convey the identity of a key owner and also for many applications
- There was a need to indicate security policy information
- There is a need to limit the damage that can result from a faulty or malicious CA
- It is important to be able to identify different keys used by the same owner at different times

- version 3 includes a number of optional extensions
- The certificate extensions fall into three main categories:
  - key and policy information
  - subject and issuer attributes
  - certification path constraints

Key and policy information:

- This extension includes subject and issuer key and a certificate policy.
- Certificate policy is set of rules that indicates the applicability of a certificate with security requirements.
- Includes:
  - **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate
  - **Subject key identifier:** Identifies the public key being certified.
  - **Key usage:** Indicates a restriction imposed on key ussage
  - **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key
  - **Certificate policies:** Certificates may be used in environments where multiple policies apply
  - **Policy mappings:** 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, additional information about the certificate subject to increase a certificate user's confidence
- 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.
- 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.

User Authentication

15.2 Remote User-Authentication Using Symmetric Encryption

15.3 Kerberos

15.4 Remote User-Authentication Using Asymmetric Encryption

Remote User-Authentication Using Symmetric Encryption :

- **Mutual Authentication**
- **One-Way Authentication**

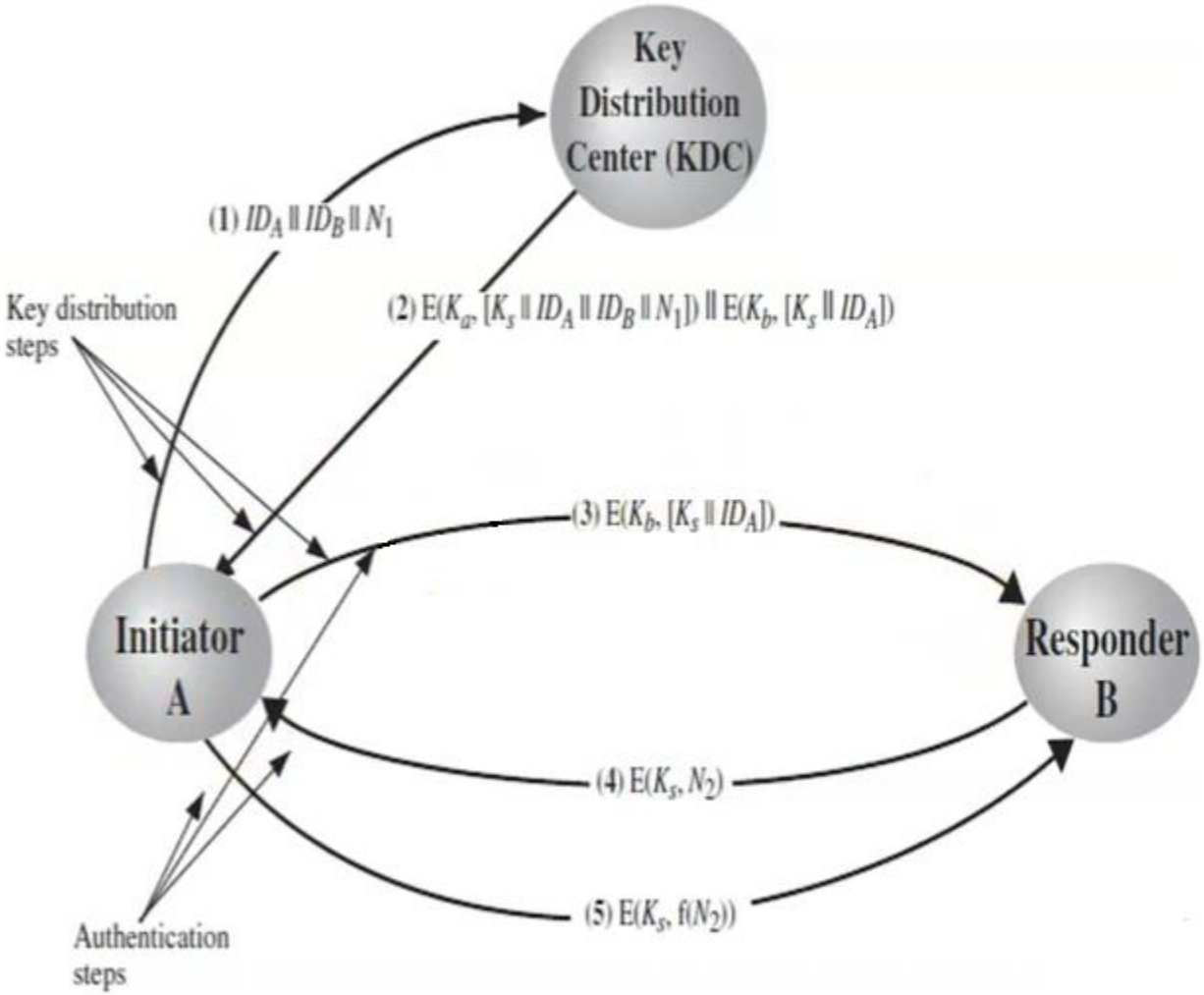
Mutual Authentication :

❑Needham – Schroeder Protocol

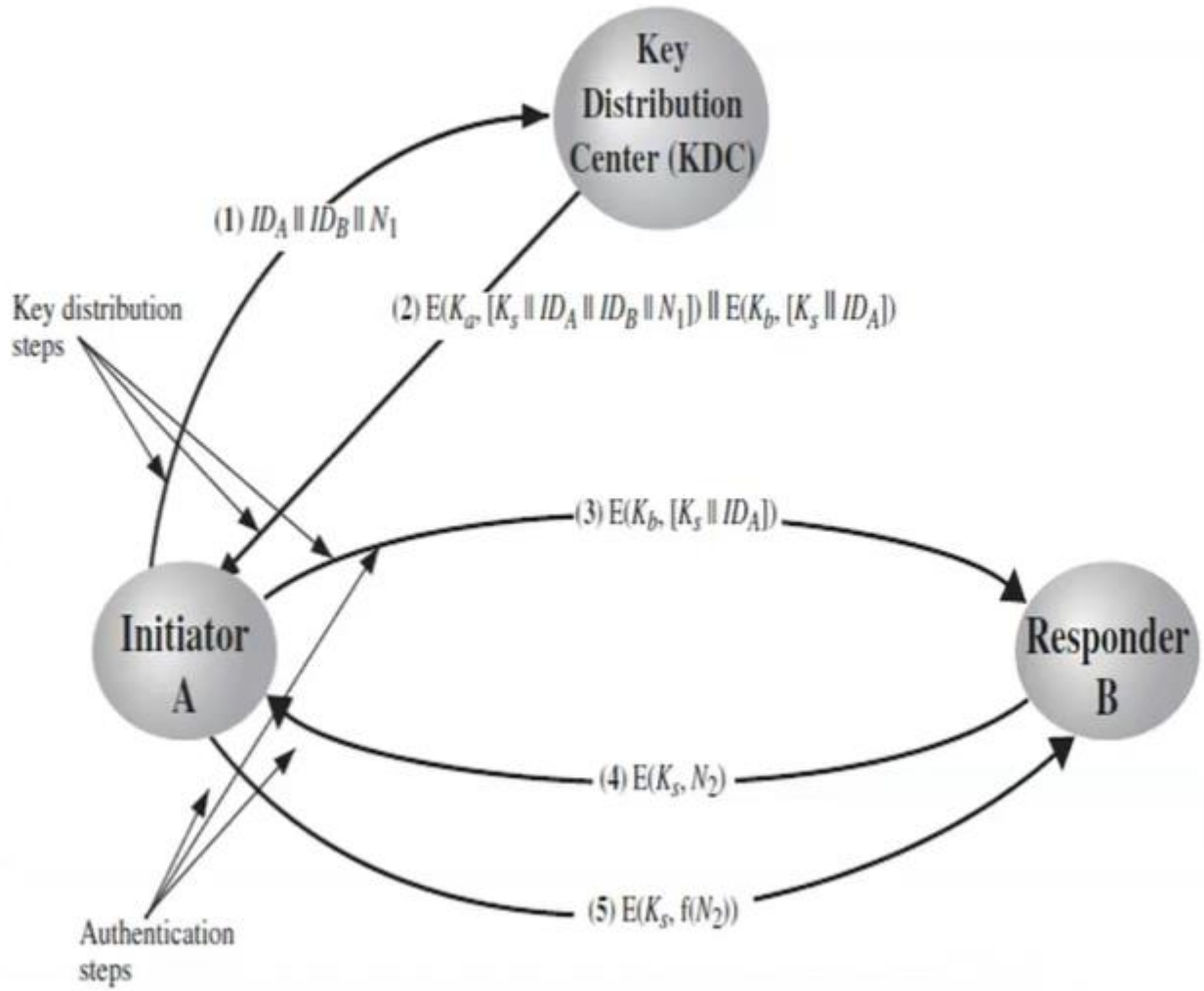
- The protocol can be summarized as follows.

1. $A \rightarrow KDC: ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B: E(K_b, [K_s \| ID_A])$
4. $B \rightarrow A: E(K_s, N_2)$
5. $A \rightarrow B: E(K_s, f(N_2))$

- The protocol is still vulnerable to a form of replay attack.
- Suppose that an opponent, X, has been able to compromise an old session key.
- X can impersonate A and trick B into using the old key by simply replaying step 3.



- Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay.
- If X can intercept the handshake message in step 4, then it can impersonate A's response in step 5.
- From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.
- Denning proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3.

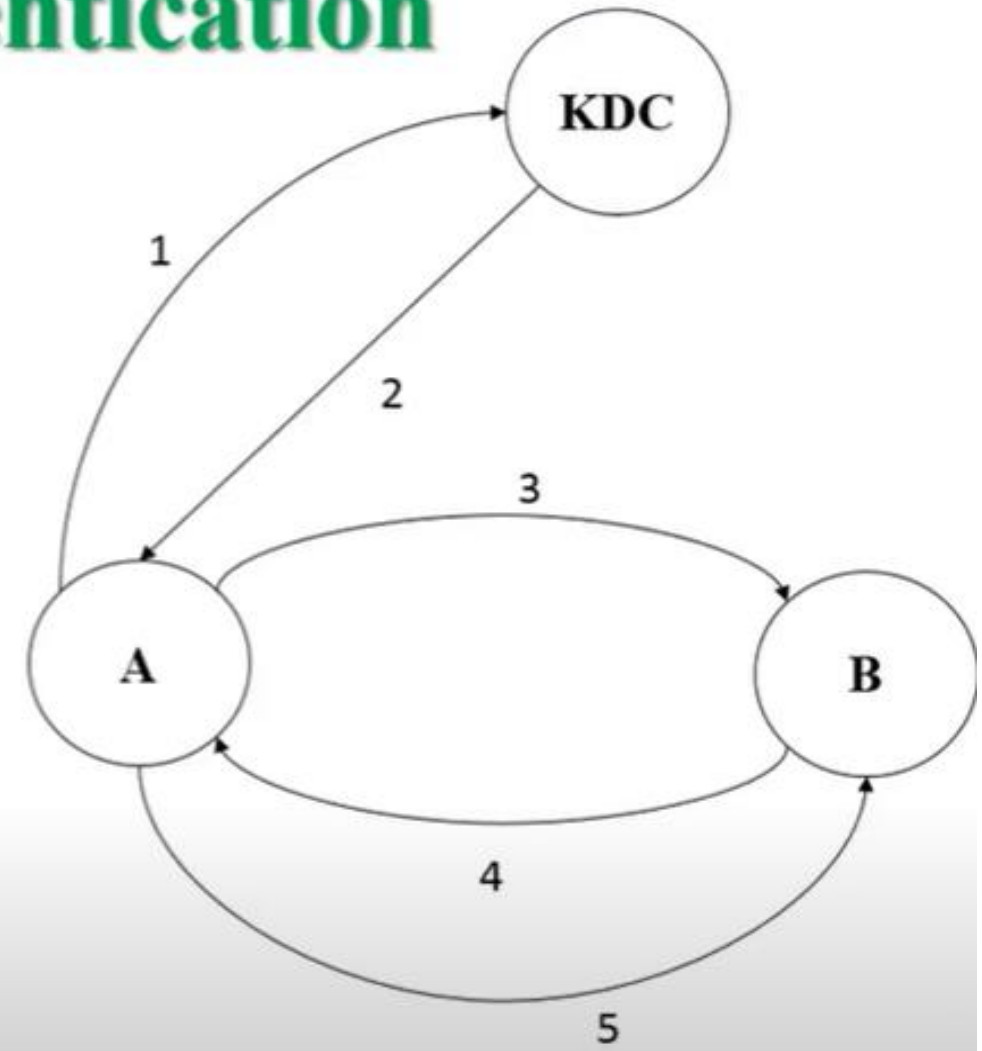


Mutual Authentication

❑Solution by Denning

- Her proposal assumes that the master keys, K_a and K_b, are secure, and it consists of the following steps.

1. $A \rightarrow KDC: ID_A || ID_B$
2. $KDC \rightarrow A: E(K_a, [K_s || ID_B || T || E(K_b, [K_s || ID_A || T])])$
3. $A \rightarrow B: E(K_b, [K_s || ID_A || T])$
4. $B \rightarrow A: E(K_s, N_1)$
5. $A \rightarrow B: E(K_s, f(N_1))$



- The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised:
- **Clocks need to be synchronized throughout the network.**
- The problem occurs when a sender's clock is ahead of the intended recipient's clock.
- In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site.
- This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.
- One way to handle this attack is parties regularly check their clocks against the KDC's clock.
- Another way avoids the need for clock synchronization, is to rely on handshaking protocols using nonces - **not vulnerable to a suppress-replay attack**
- **Solution to suppress-replay attack**

1. $A \rightarrow B$: $ID_A \| N_a$
2. $B \rightarrow KDC$: $ID_B \| N_b \| E(K_b, [ID_A \| N_a \| T_b])$
3. $KDC \rightarrow A$: $E(K_a, [ID_B \| N_a \| K_s \| T_b]) \| E(K_b, [ID_A \| K_s \| T_b]) \| N_b$
4. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

- This protocol provides an effective, secure means for A and B to establish a session with a secure session key.
- The protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly.
- **Within the time limit established by the protocol, if A desires a new session with B**

1. $A \rightarrow B: \quad E(K_b, [ID_A \| K_s \| T_b]) \| N'_a$
2. $B \rightarrow A: \quad N'_b \| E(K_s, N'_a)$
3. $A \rightarrow B: \quad E(K_s, N'_b)$

- When B receives the message in step 1, it verifies that the ticket has not expired.
- The newly generated nonces *Na* = and *Nb* = assure each party that there is no replay attack.
- In all the foregoing, the time specified in *Tb* is a time relative to B's clock.
- Thus, this timestamp does not require synchronized clocks, because B checks only self-generated timestamps.

One-Way Authentication:

- Using symmetric encryption, the decentralized key distribution scenario, requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.
- **To avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated.**
- For a message with content *M*, the sequence is as follows:

1. $A \rightarrow KDC: \quad ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

- This approach guarantees that only the intended recipient of a message will be able to read it.
- It also provides a level of authentication that the sender is A.
- As specified, the protocol does not protect against replays.
- Some measure of defense could be provided by including a timestamp with the message.
- However, because of the potential delays in the email process, such timestamps may have limited usefulness.

Kerberos

- Kerberos provides a centralized authentication server
- It’s function is to authenticate users to servers and servers to users.
- Kerberos is a network authentication protocol that works on the basis of tickets to allow nodes communicating over non secure network to prove their identity to one another in a secure manner.
- Two versions of Kerberos are in common use.
  - Version 4
  - Version 5

The first published report on Kerberos [STEI88] listed the following requirements.

■ **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user.

■ **Reliable:**

- Kerberos should be highly reliable
- should employ a distributed server architecture with one system able to **back up** another.

■ **Transparent:** The user should not be aware that authentication is taking place beyond the requirement to enter a password.

■ **Scalable:** The system should be capable of supporting large numbers of clients and servers.

Kerberos Version 4 using authentication server (AS) :

- Version 4 of Kerberos makes use of DES

(1) C → AS: $ID_C \| P_C \| ID_V$

(2) AS → C: $Ticket$

(3) C → V: $ID_C \| Ticket$

$Ticket = E(K_v, [ID_C \| AD_C \| ID_V])$

where

C = client

AS = authentication server

V = server

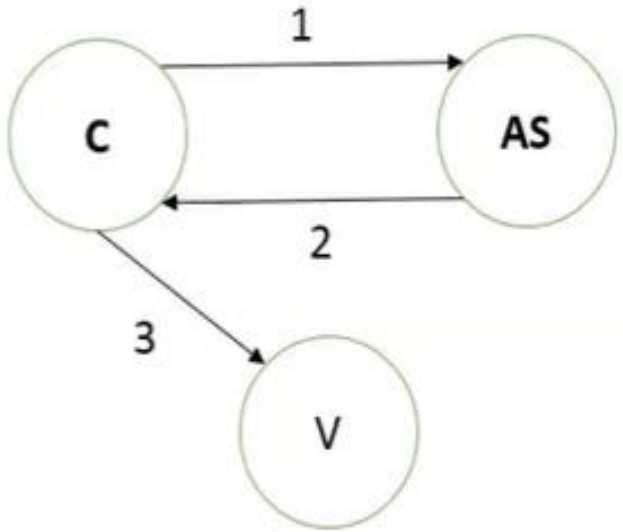
ID_C = identifier of user on C

ID_V = identifier of V

P_C = password of user on C

AD_C = network address of C

K_v = secret encryption key shared by AS and V



Limitation :

- user would need a new ticket for every different service
- Password is transmitted without encryption

Solution :

- **Ticket-Granting Server (TGS)**.

Kerberos Version 4 using Ticket-Granting Server (TGS) :

Once per user logon session:

(1) C → AS: $ID_C \| ID_{tgs}$

(2) AS → C: $E(K_c, Ticket_{tgs})$

Once per type of service:

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

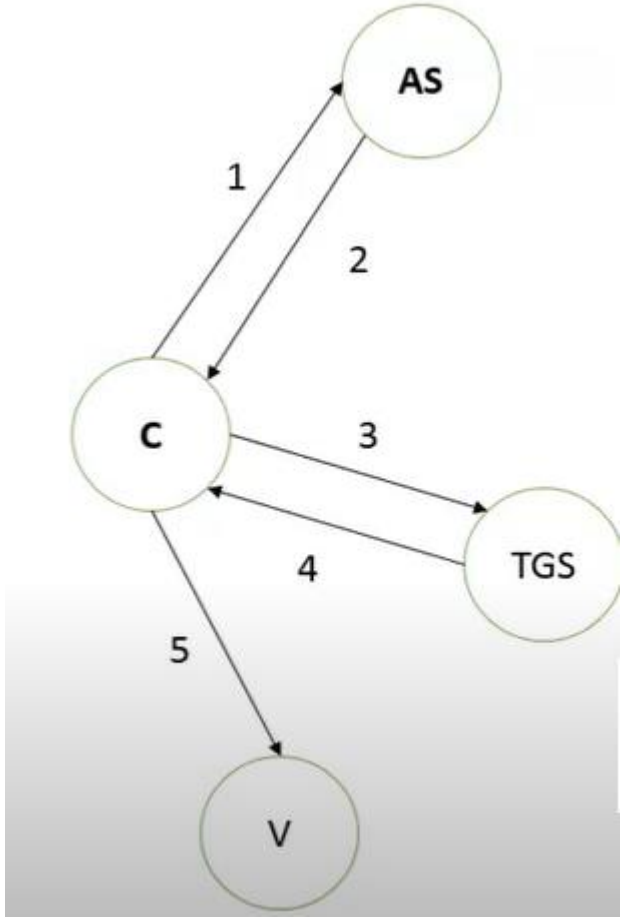
(4) $TGS \rightarrow C$: $Ticket_v$

Once per service session:

(5) C → V: $ID_C \| Ticket_v$

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

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



K_c = key that is derived from user password

K_{tgs} = key shared only by the AS and the TGS

K_v = key shared between server and TGS

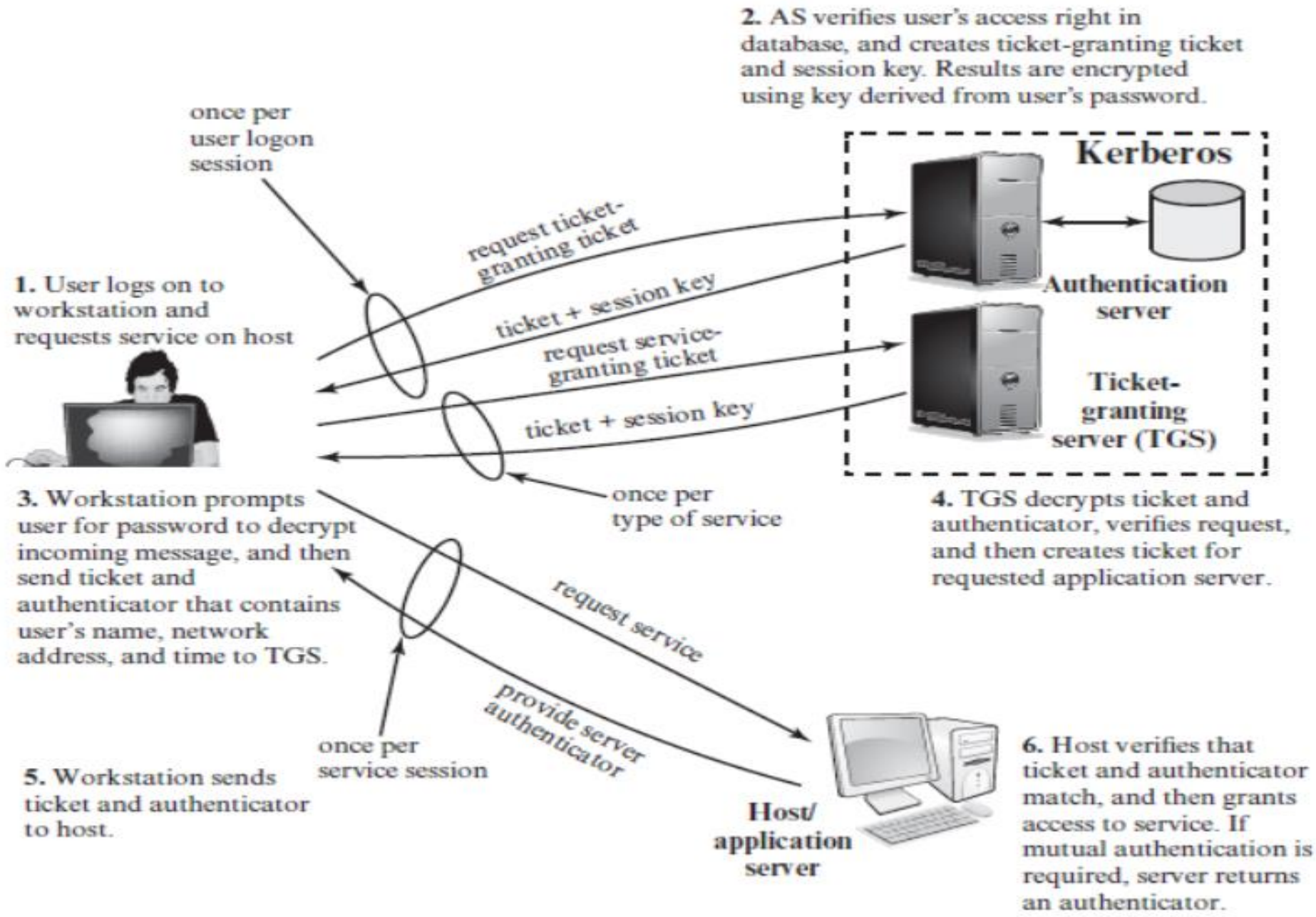


Figure 15.2 Overview of Kerberos

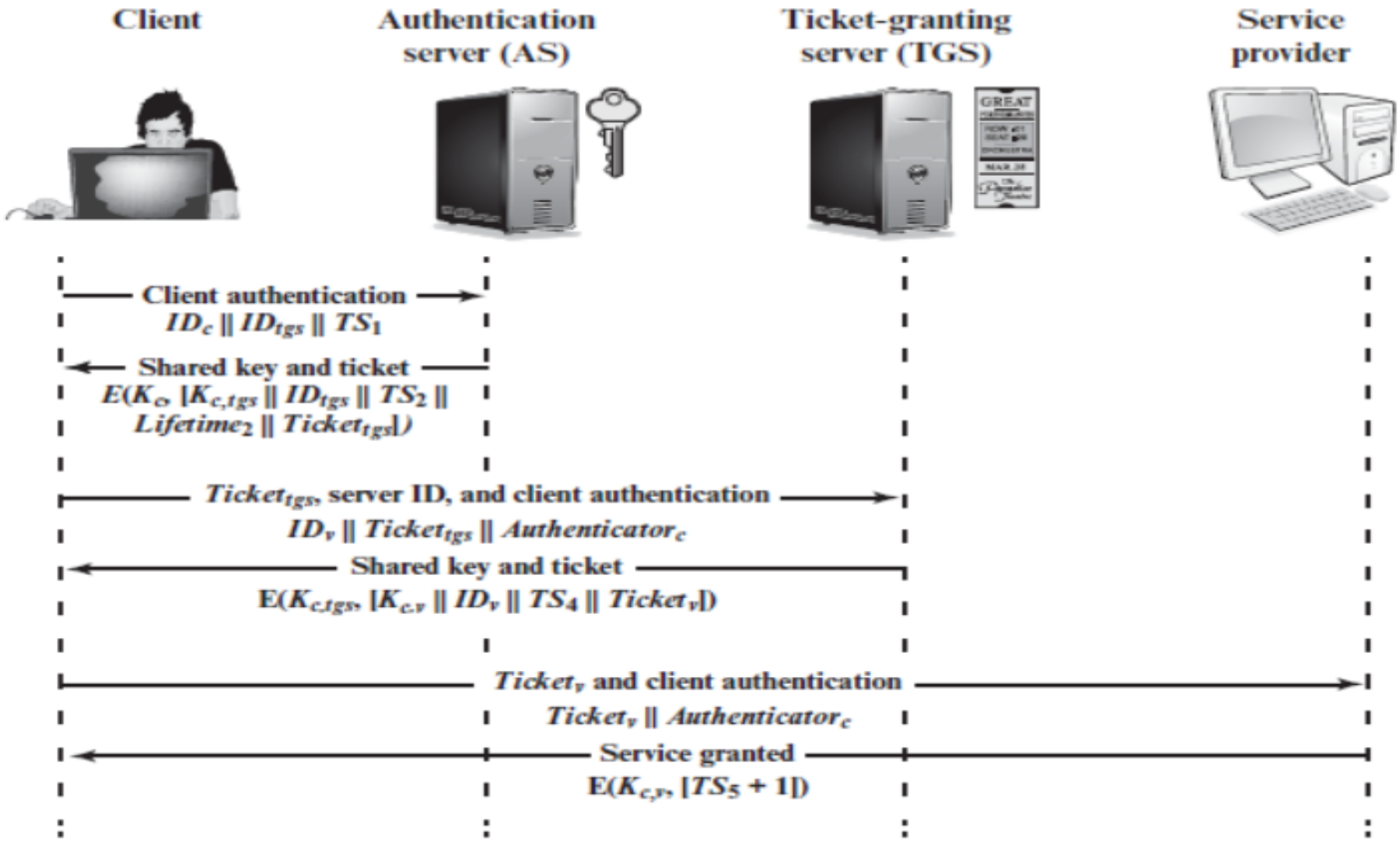


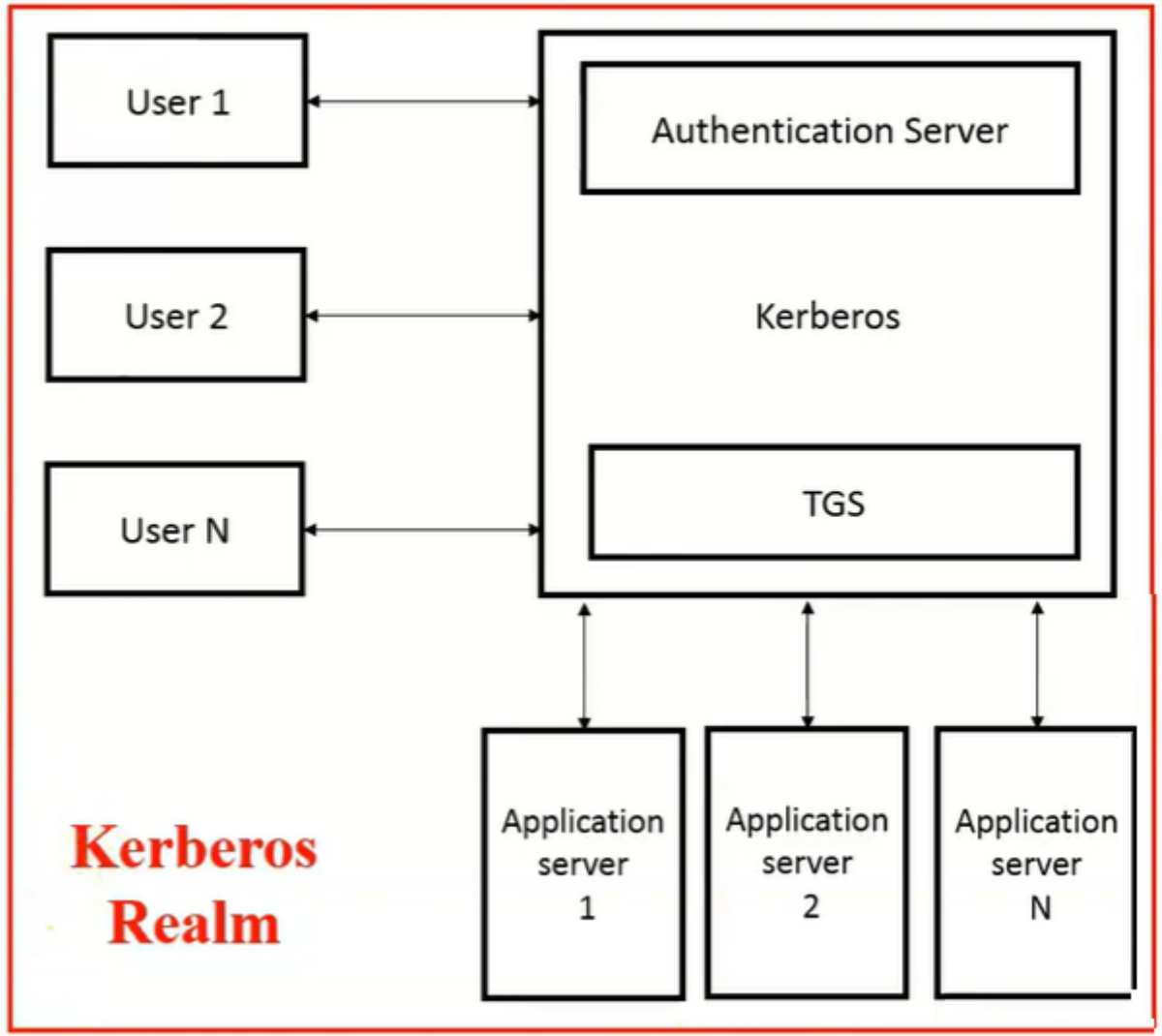
Figure 15.3 Kerberos Exchanges

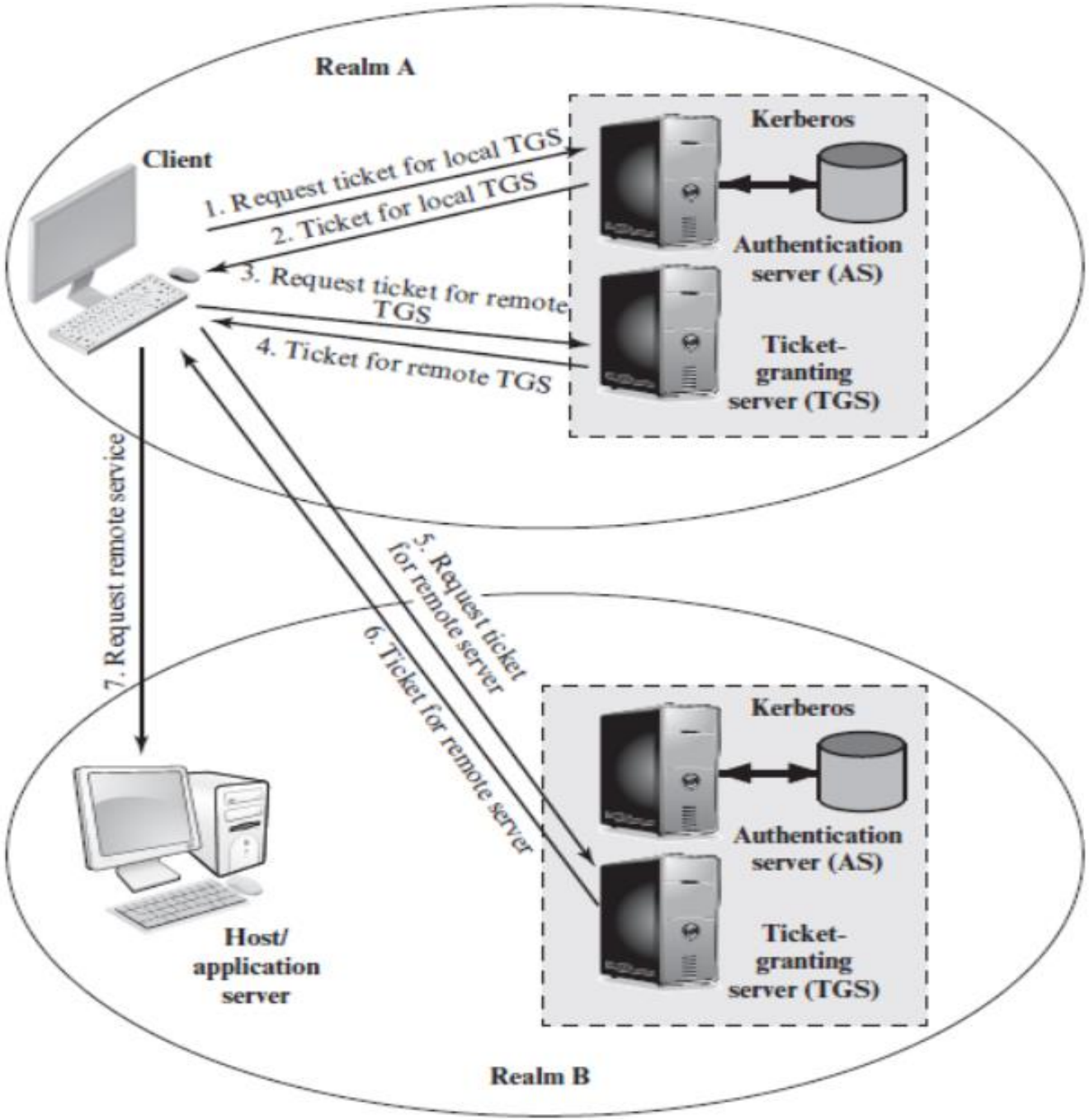
KERBEROS REALM

❑ Kerberos Realm

- A full-service Kerberos environment consists of
  - a Kerberos server
  - a number of clients, all are registered with Kerberos server
  - a number of application servers, all are sharing keys with Kerberos server

- Such an environment is referred to as a **Kerberos realm.**





(1) $C \rightarrow AS$: $ID_c \| ID_{tgs} \| TS_1$

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

(3) $C \rightarrow TGS$: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

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

(5) $C \rightarrow TGS_{rem}$: $ID_{vrem} \| Ticket_{tgsrem} \| Authenticator_c$

(6) $TGS_{rem} \rightarrow C$: $E(K_{c,tgsrem}, [K_{c,vrem} \| ID_{vrem} \| TS_6 \| Ticket_{vrem}])$

(7) $C \rightarrow V_{rem}$: $Ticket_{vrem} \| Authenticator_c$

Figure 15.4 Request for Service in Another Realm

Shortcomings of Kerberos Version 4:

Encryption system dependence:

version 4 : Encryption algorithm used is DES

version 5 : Encryption algorithm used is AES

Internet protocol dependence:

version 4 : requires use of IP address only

version 5 : any network address type eg : ISO network address

Message byte ordering:

version 4 : its own byte ordering is used

version 5 : ASN.1 rules and BER (Basic encoding rules) used to provide unambiguous byte order

Ticket lifetime:

version 4 : 21 hours

version 5 : arbitrary (based on random choice)

Authentication forwarding:

version 4 : not supported

version 5 : supported

Interrealm authentication:

version 4 : N^2 relationship (N is the number of Realm)

version 5 : fewer relationships

Technical Limitations :

Double encryption:

- Tickets provided to clients are encrypted twice—once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

PCBC encryption:

- **propagating cipher block chaining (PCBC)**. this mode is vulnerable to an attack involving the interchange of ciphertext blocks.
- In version 5 a checksum or hash code is attached to the message prior to encryption using CBC.

Session keys:

- Each ticket includes a session key that is used by the client to encrypt the authenticator sent to the service associated with that ticket.
- same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages

Password attacks:

- Both versions are vulnerable to a password attack
- Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

Kerberos Version 5:

- Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4

  - **Realm:** Indicates realm of user
  - **Options:** Used to request that certain flags be set in the returned ticket
  - **Times:** Used by the client to request the following time settings in the ticket:
    - —**from**: the desired start time for the requested ticket
    - —**till**: the requested expiration time for the requested ticket
    - —**rtime**: requested renew-till time
  - **Nonce:** A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

  - **Subkey:** The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket ($K_{c,v}$) is used.
  - **Sequence number:** An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

Summary of Kerberos Version 4 Message Exchanges

1. Authentication Service Exchange to obtain ticket-granting ticket

(1) $C \rightarrow AS \quad ID_c \| ID_{tgs} \| TS_1$

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

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

2. Ticket-Granting Service Exchange to obtain service-granting ticket

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

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

$$Ticket_{tgs} = E(K_{tgs}, [K_{c,tgs} \| ID_C \| AD_C \| ID_{tgs} \| TS_2 \| Lifetime_2])$$

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,tgs}, [ID_C \| AD_C \| TS_3])$$

3. Client/Server Authentication Exchange to obtain service

(5) $C \rightarrow V \quad Ticket_v \| Authenticator_c$

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

$$Ticket_v = E(K_v, [K_{c,v} \| ID_C \| AD_C \| ID_v \| TS_4 \| Lifetime_4])$$

$$Authenticator_c = E(K_{c,v}, [ID_C \| AD_C \| TS_5])$$

Summary of Kerberos Version 5 Message Exchanges

1. Authentication Service Exchange to obtain ticket-granting ticket

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

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

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

2. Ticket-Granting Service Exchange to obtain service-granting ticket

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

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

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

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

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

3. Client/Server Authentication Exchange to obtain service

(5) $C \rightarrow V \quad Options \| Ticket_v \| Authenticator_c$

(6) $V \rightarrow C \quad E_{K_{c,v}}[TS_2 \| Subkey \| Seq\ \#]$

$$Ticket_v = E(K_v, [Flag \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$$

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

Table 15.4 Kerberos Version 5 Flags

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

Remote User-Authentication Using Asymmetric Encryption

Mutual Authentication

A protocol using timestamps is provided in [DENN81]:

1. $A \rightarrow AS: \quad ID_A \| ID_B$
2. $AS \rightarrow A: \quad E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T])$
3. $A \rightarrow B: \quad E(PR_{as}, [ID_A \| PU_a \| T]) \| E(PR_{as}, [ID_B \| PU_b \| T]) \| E(PU_b, E(PR_a, [K_s \| T]))$

Advantages :

- The session key is chosen and encrypted by A.
- The timestamps protect against replays of compromised keys.

Limitations :

- This protocol is compact but, as before, requires the synchronization of clocks.

- Another approach, proposed by Woo and Lam [WOO92a], makes use of **nonces.** The protocol consists of the following steps.
- This is secure protocol that takes into account the various attacks

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$
4. B → KDC: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_B]))$
6. B → A: $E(PU_a, [E(PR_{auth}, [(N_a \| K_s \| ID_B)]) \| N_b])$
7. A → B: $E(K_s, N_b)$

Revised one :

1. A → KDC: $ID_A \| ID_B$
2. KDC → A: $E(PR_{auth}, [ID_B \| PU_b])$
3. A → B: $E(PU_b, [N_a \| ID_A])$
4. B → KDC: $ID_A \| ID_B \| E(PU_{auth}, N_a)$
5. KDC → B: $E(PR_{auth}, [ID_A \| PU_a]) \| E(PU_b, E(PR_{auth}, [N_a \| K_s \| ID_A \| ID_B]))$
6. B → A: $E(PU_a, [N_b \| E(PR_{auth}, [N_a \| K_s \| ID_A \| ID_B])])$
7. A → B: $E(K_s, N_b)$

One-Way Authentication

Public-key algorithm must be applied once or twice to what may be a long message.

1. If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B: \quad E(PU_b, K_s) \| E(K_s, M)$$

- In this case, the message is encrypted with a **one-time secret key** than simply encrypting the entire message with B's public key. **Session key is encrypted using B's public key**

2. If authentication is the primary concern,

$$A \rightarrow B: \quad M \| E(PR_a, H(M))$$

- This method guarantees that **A cannot later deny having sent the message.**
- This technique is open for another kind of fraud

- Eg :**Bob** composes a message to his boss **Alice** that contains an idea that will save the company money. **Bob** appends his digital signature and sends it into the email system. Eventually, the message will get delivered to **Alice's** mailbox.
- But suppose that **Max** has heard of Bob's idea and gains access to the mail queue before delivery. He finds **Bob's** message, strips off his signature, appends Max's digital signature, and requeues the message to be delivered to Alice. **Max gets credit for Bob's idea.**

- To counter such a scheme, **both the message and signature can be encrypted with the recipient's public key:**

$$\mathrm{A} \rightarrow \mathrm{B}: \quad \mathrm{E}(PU_b, [M \| \mathrm{E}(PR_a, H(M))])$$

- The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate.

$$\mathrm{A} \rightarrow \mathrm{B}: \quad M \| \mathrm{E}(PR_a, \mathrm{H}(M)) \| \mathrm{E}(PR_{as}, [T \| ID_A \| PU_a])$$

- A sends B the signature encrypted with A's private key.
- A's certificate encrypted with the private key of the authentication server.
- The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself.

Chapter 17 :

Web Security Considerations

- World Wide Web is fundamentally a client/server application running over the Internet and TCP/IP intranets.
- Web browsers are very easy to use
- Web servers are relatively easy to configure and manage
- Web content is easy to develop but the **underlying software is extraordinarily complex** and vulnerable to a variety of security attacks.
- A **Web server can be exploited as a launching pad**, an attacker may be able to gain access to data and systems
- **Casual and untrained (in security matters) users** are common clients for Web based services. Such users are not necessarily aware of the security risks that exist and do not have the tools or knowledge to take effective countermeasures.

Web Security Threats can be classified as

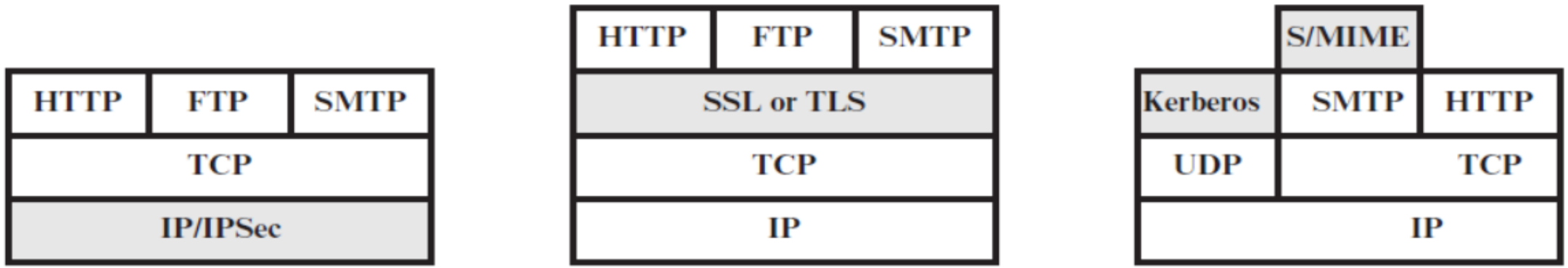
- **Passive attack :** include eavesdropping on network traffic between browser and server and gaining access to information on a Web site that is supposed to be restricted.
- **Active attack :** include impersonating another user, altering messages in transit between client and server, and altering information on a Web site.

Web Security Threats

	Threats	Consequences	Countermeasures
Integrity	• Modification of user data • Trojan horse browser • Modification of memory • Modification of message traffic in transit	• Loss of information • Compromise of machine • Vulnerability to all other threats	Cryptographic checksums
Confidentiality	• Eavesdropping on the net • Theft of info from server • Theft of data from client • Info about network configuration • Info about which client talks to server	• Loss of information • Loss of privacy	Encryption, Web proxies
Denial of Service	• Killing of user threads • Flooding machine with bogus requests • Filling up disk or memory • Isolating machine by DNS attacks	• Disruptive • Annoying • Prevent user from getting work done	Difficult to prevent
Authentication	• Impersonation of legitimate users • Data forgery	• Misrepresentation of user • Belief that false information is valid	Cryptographic techniques

Web Traffic Security Approaches

- A number of approaches to providing Web security is possible
- Approaches differ with respect to their scope of applicability and location within the TCP/IP protocol stack.



(a) Network level (b) Transport level (c) Application level

Figure 17.1 Relative Location of Security Facilities in the TCP/IP Protocol Stack

Web Traffic Security Approaches

1. One way to provide Web security is to use IP security (IPsec)

Advantage :

- It is transparent to end users and applications and provides a general-purpose solution.
- IPsec includes a filtering capability so that only selected traffics are processed

2. Implement security just above TCP. Example : Sockets Layer (SSL) also known as Transport Layer Security (TLS).

Two Implementation choices :

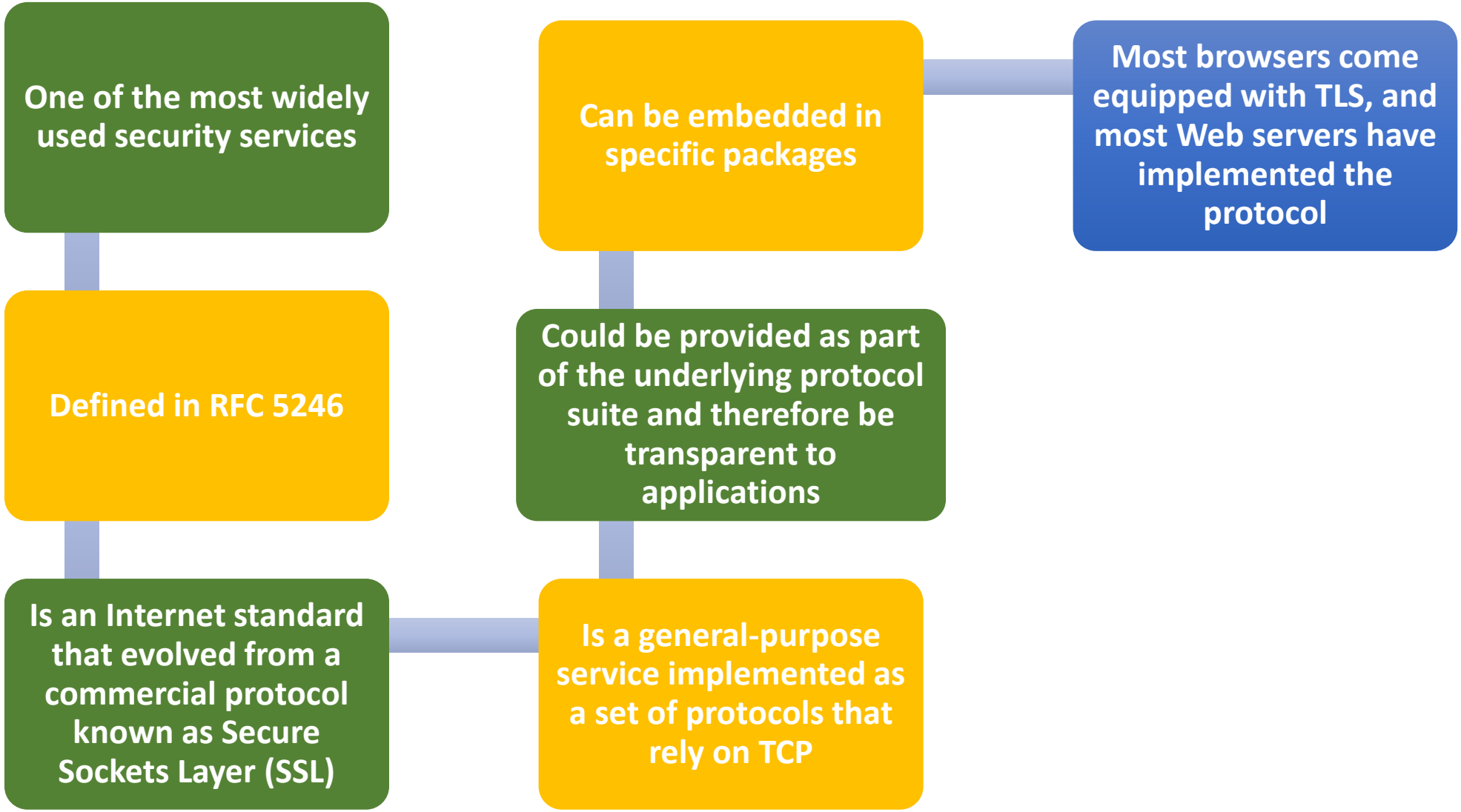
- SSL (or TLS) could be provided as part of the underlying protocol suite and therefore be transparent to applications.
- TLS can be embedded in specific packages.

3. Application-specific security services are embedded within the application.

Advantages :

- service can be tailored to the specific needs of a given application.

Transport Layer Security



TLS Architecture

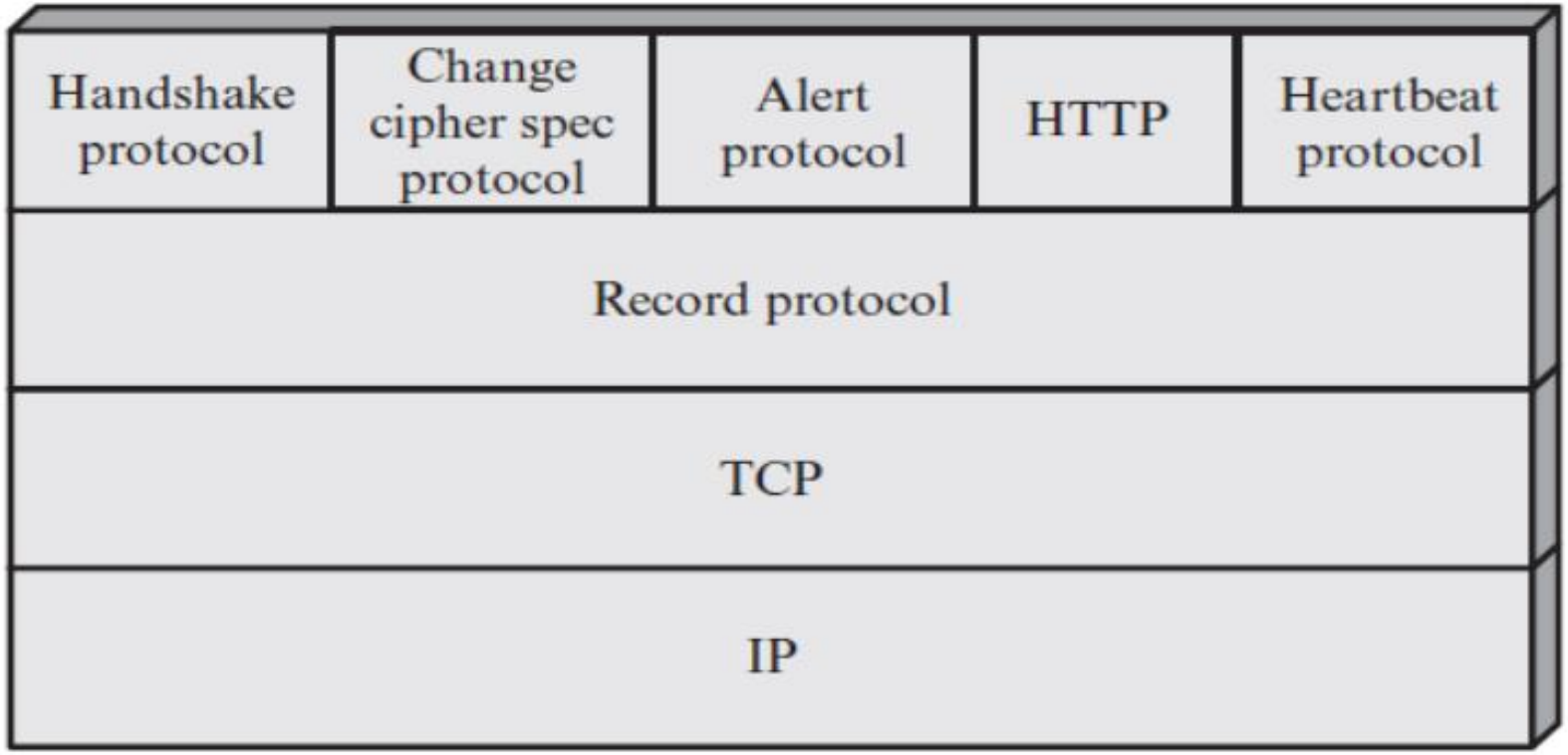


Figure 17.2 TLS Protocol Stack

- TLS is designed to make use of TCP to provide a reliable end-to-end secure service.
- TLS is not a single protocol but rather two layers of protocols.
- The TLS Record Protocol provides basic security services to various higher layer protocols.
  - Hypertext Transfer Protocol (HTTP): provides the transfer service for Web client/server interaction, can operate on top of TLS.
- Three higher-layer protocols are defined as part of TLS:
  - Handshake Protocol
  - Change Cipher Spec Protocol
  - Alert Protocol.
- These TLS specific protocols are used in the management of TLS exchanges
  - A fourth protocol, the Heartbeat Protocol, is defined in a separate RFC

Two important TLS concepts are the TLS session and the TLS connection

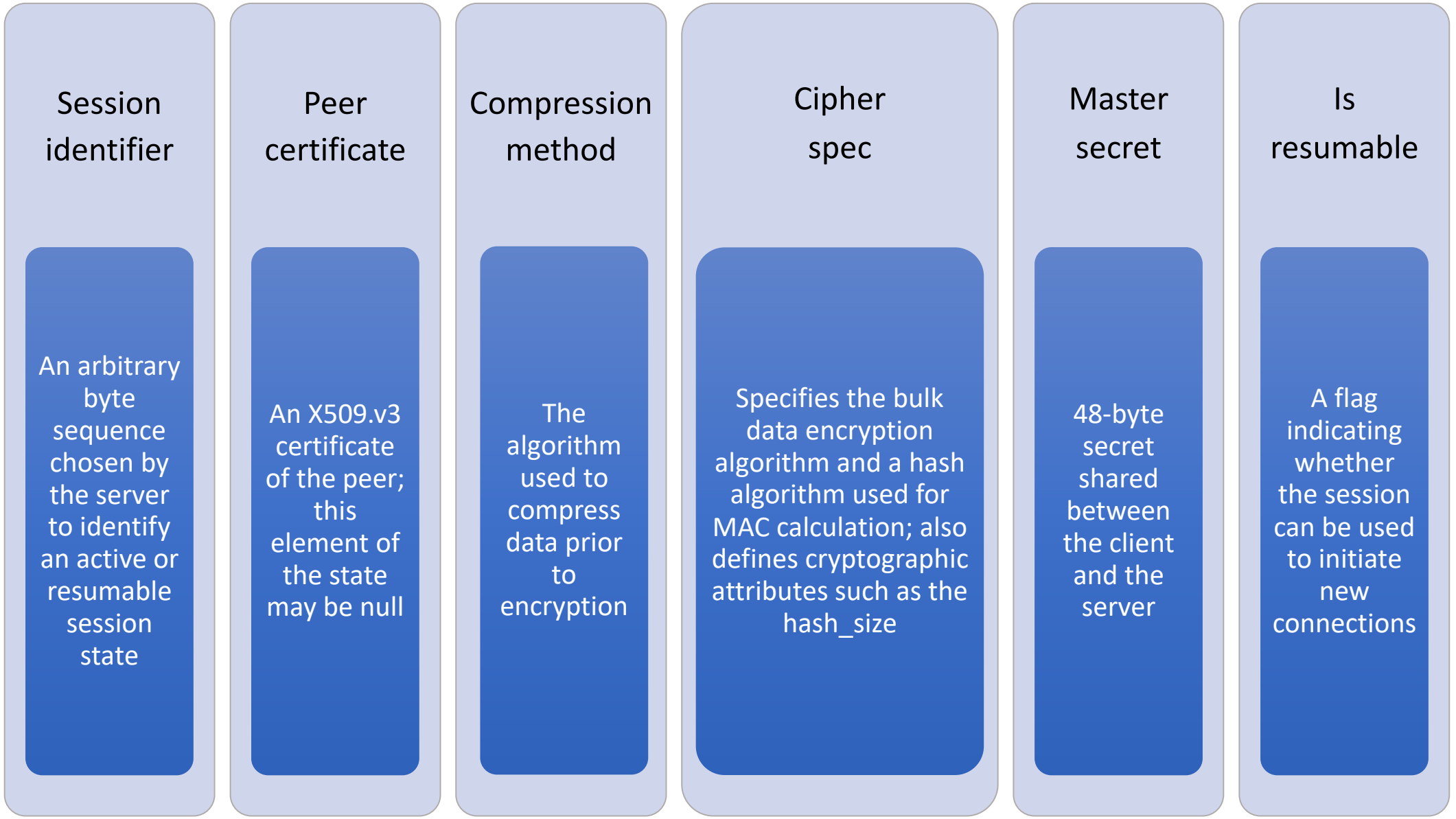
TLS connection

- A transport that provides a suitable type of service
- For TLS such connections are peer-to-peer relationships
- Connections are transient (Short term connections)
- Every connection is associated with one session

TLS session

- An association between a client and a server
- Created by the Handshake Protocol
- Define a set of cryptographic security parameters which can be shared among multiple connections
- Are used to avoid the expensive negotiation of new security parameters for each connection

- A session state is defined by the following parameters:



- A connection state is defined by the following parameters:

Server and client random
- Byte sequences that are chosen by the server and client for each connection

Server write MAC secret
- The secret key used in MAC operations on data sent by the server

Client write MAC secret
- The secret key used in MAC operations on data sent by the client

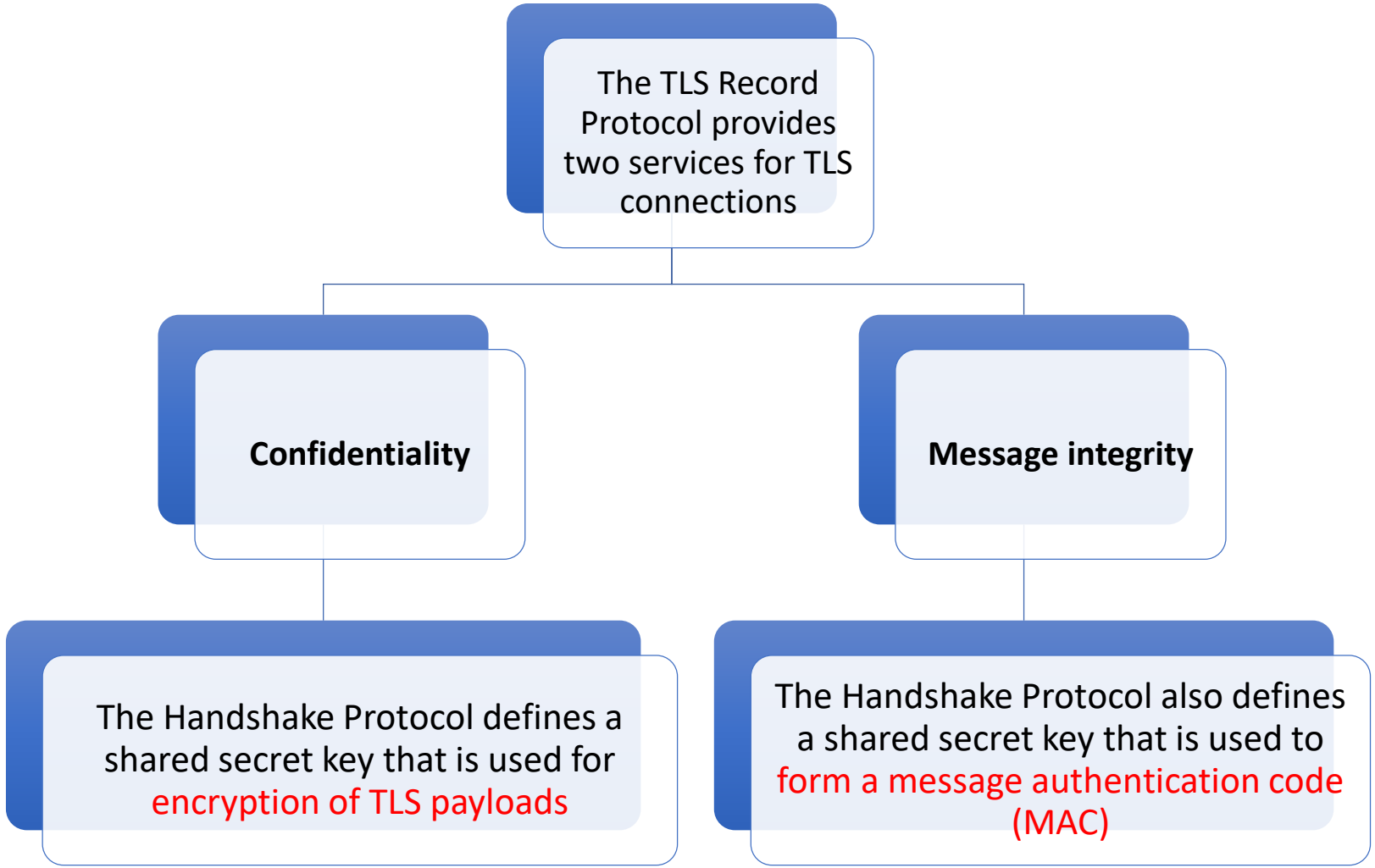
Server write key
- The secret encryption key for data encrypted by the server and decrypted by the client

Client write key
- The symmetric encryption key for data encrypted by the client and decrypted by the server

Initialization vectors
- When a block cipher in CBC mode is used, an initialization vector (IV) is maintained for each key
- This field is first initialized by the TLS Handshake Protocol
- The final ciphertext block from each record is preserved for use as the IV with the following record

Sequence numbers
- Each party maintains separate sequence numbers for transmitted and received messages for each connection
- When a party sends or receives a change cipher spec message, the appropriate sequence number is set to zero
- Sequence numbers may not exceed $2^{64} - 1$

TLS Record Protocol



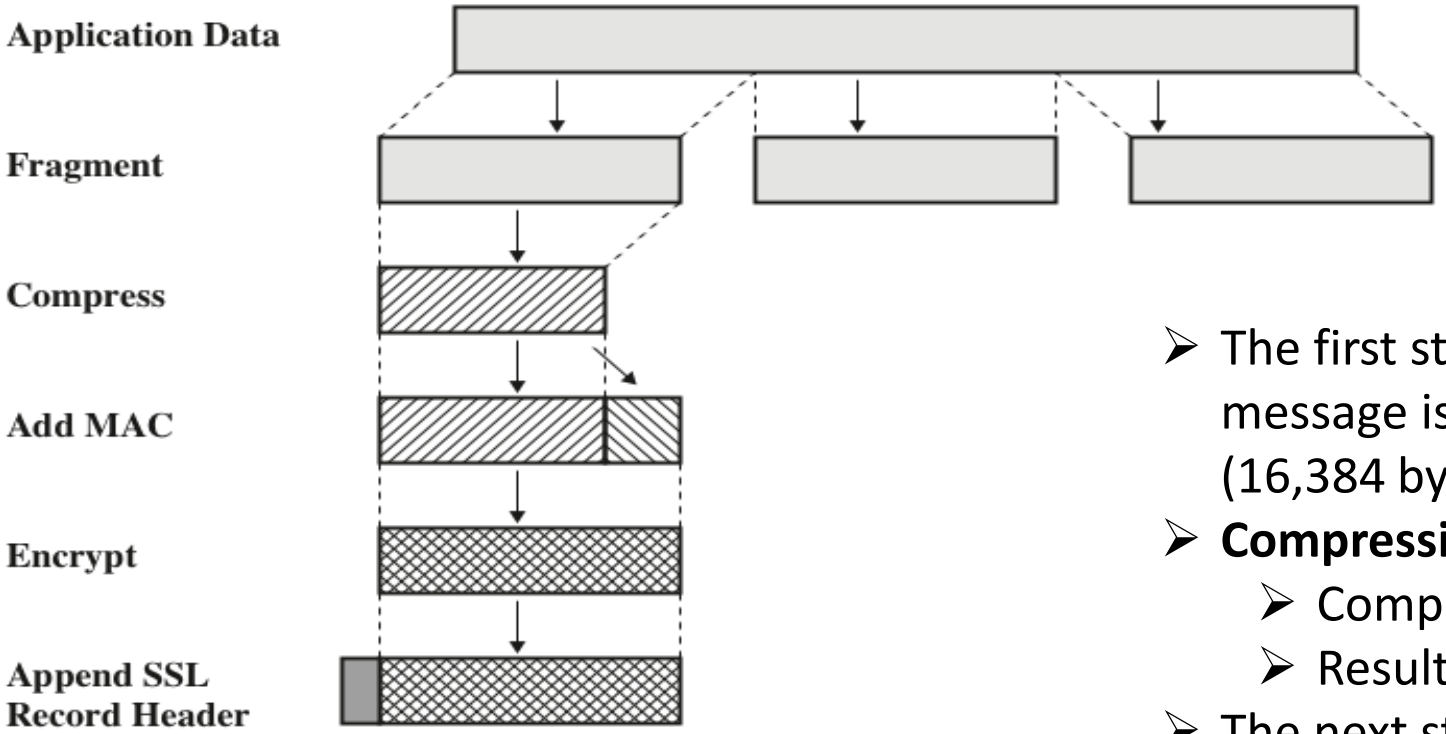


Figure 17.3 SSL Record Protocol Operation

- The first step is **fragmentation**. Each upper-layer message is fragmented into blocks of 214 bytes (16,384 bytes) or less
- **Compression** is optionally applied.
  - Compression must be lossless.
  - Resulting length < 1024 bytes
- The next step is to compute a **message authentication code** over the compressed data.
- Next, the compressed message plus the MAC are **encrypted** using symmetric encryption.
- Final step is to append record header

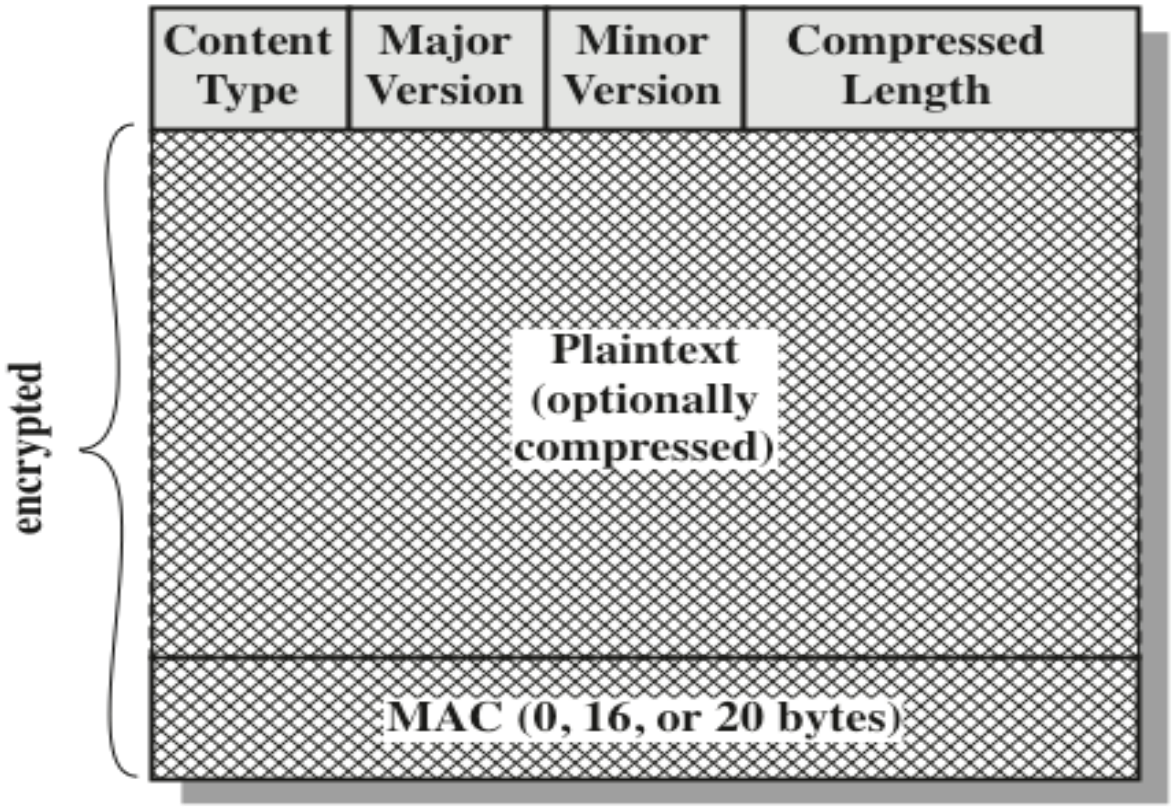


Figure 17.4 SSL Record Format

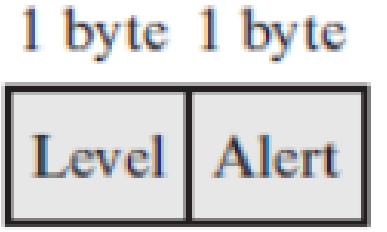
- **Content Type (8 bits)**: The higher-layer protocol used to process the enclosed fragment.
- **Major Version (8 bits):** Indicates major version of TLS in use. For TLSv2, the value is 3.
- **Minor Version (8 bits):** Indicates minor version in use. For TLSv2, the value is 1.
- **Compressed Length (16 bits):** The length in bytes of the plaintext fragment (or compressed fragment if compression is used).
  - The maximum value is 2^14 + 2048.

Change Cipher Spec Protocol

- The Change Cipher Spec Protocol is one of the four TLS-specific protocols that use the TLS Record Protocol
- It is simplest protocol.
- This protocol consists of a single message which consists of a **single byte with the value 1.**
- The sole purpose of this message is to cause the **pending state to be copied into the current state**, which updates the cipher suite to be used on this connection.

(a) Change Cipher Spec Protocol

Alert Protocol



(b) Alert Protocol

- The Alert Protocol **is used to convey TLS-related alerts to the peer entity**
- Alert messages are compressed and encrypted.
- Each message in this protocol consists of two bytes
- **The first byte takes the value warning (1) or fatal (2) to convey the severity of the message.**
  - if the level is fatal, TLS immediately terminates the connection.
  - Other connections on the same session may continue, but no new connections on this session may be established.
- **The second byte contains a code that indicates the specific alert.**
- Some of the alert messages:

  - **unexpected_message:** An inappropriate message was received.
  - **bad_record_mac:** An incorrect MAC was received.
  - **decompression_failure:** The decompression function received improper input
  - **handshake_failure:** Sender was unable to negotiate an acceptable set of security parameters given the options available.
  - **illegal_parameter:** A field in a handshake message was out of range or inconsistent with other fields.

- **decryption_failed:** A ciphertext decrypted in an invalid way; either it was not an even multiple of the block length or its padding values, when checked, were incorrect.
- **record_overflow:** A TLS record was received with a payload (ciphertext) whose length exceeds $2^{14} + 2048$ bytes, or the ciphertext decrypted to a length of greater than $2^{14} + 1024$ bytes.
- **unknown_ca:** A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or could not be matched with a known, trusted CA.
- **access_denied:** A valid certificate was received, but when access control was applied, the sender decided not to proceed with the negotiation.
- **decode_error:** A message could not be decoded, because either a field was out of its specified range or the length of the message was incorrect.
- **export_restriction:** A negotiation not in compliance with export restrictions on key length was detected.
- **protocol_version:** The protocol version the client attempted to negotiate is recognized but not supported.
- **insufficient_security:** Returned instead of handshake_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
- **internal_error:** An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue.

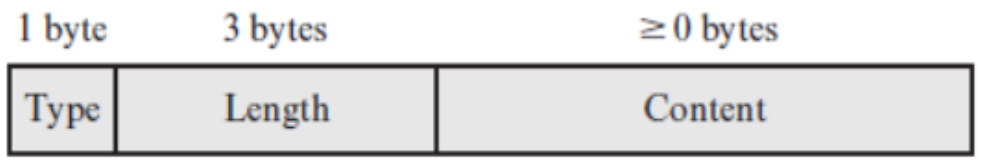
The remaining alerts are the following.

- **close_notify:** Notifies the recipient that the sender will not send any more messages on this connection. Each party is required to send a close_notify alert before closing the write side of a connection.
- **bad_certificate:** A received certificate was corrupt (e.g., contained a signature that did not verify).
- **unsupported_certificate:** The type of the received certificate is not supported.
- **certificate_revoked:** A certificate has been revoked by its signer.
- **certificate_expired:** A certificate has expired.
- **certificate_unknown:** Some other unspecified issue arose in processing the certificate, rendering it unacceptable.
- **decrypt_error:** A handshake cryptographic operation failed, including being unable to verify a signature, decrypt a key exchange, or validate a finished message.
- **user_canceled:** This handshake is being canceled for some reason unrelated to a protocol failure.
- **no_renegotiation:** Sent by a client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these messages would normally result in renegotiation, but this alert indicates that the sender is not able to renegotiate. This message is always a warning.

Handshake Protocol

- The most complex part of TLS is the **Handshake Protocol**
- This protocol allows the **server and client to authenticate each other.**
- The Handshake Protocol is used before any application data is transmitted.
- The Handshake Protocol consists of a series of messages exchanged by client and server
- Each message has three fields:

  - **Type (1 byte):** Indicates one of 10 messages. Table 17.2 lists the defined message types.
  - **Length (3 bytes):** The length of the message in bytes.
  - **Content (≥ 0 bytes):** The parameters associated with this message; these are listed in Table 17.2.



(c) Handshake Protocol

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

Table 17.2 TLS Handshake Protocol Message Types

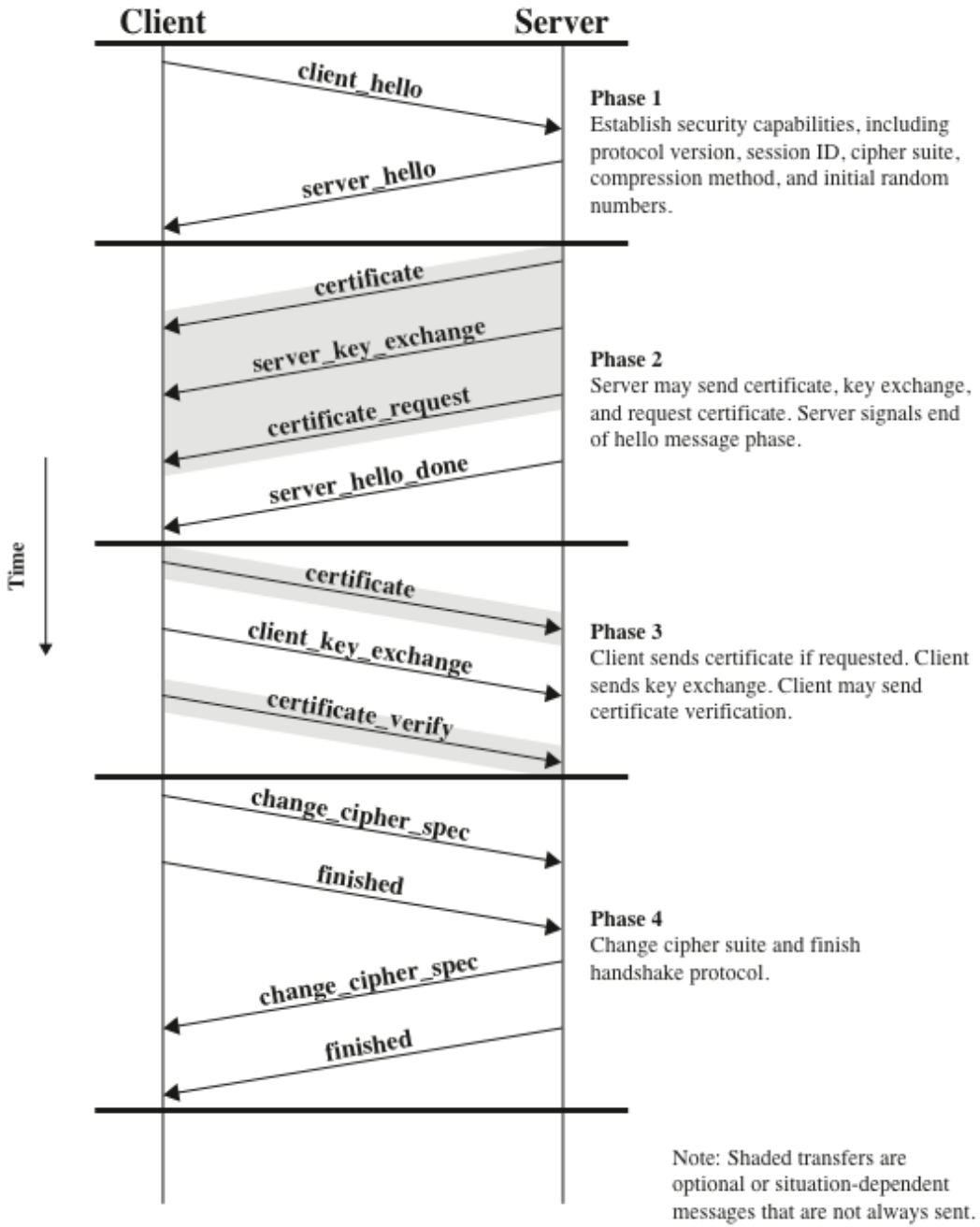


Figure 17.6 Handshake Protocol Action

Cryptographic Computations

Two Steps :

- The creation of a shared master secret by means of the key exchange
  - a one-time 48-byte value
  - generated using secure key exchange (RSA / Diffie-Hellman) and then hashing info
  - The creation is in two stages
    - First, a pre_master_secret is exchanged
    - Second, the master_secret is calculated by both parties
- The generation of cryptographic parameters from the master secret
  - client write MAC secret, a server write MAC secret, a client write key, a server write key, a client write IV, and a server write IV
  - generated by hashing master secret by hashing the master secret into a sequence of secure bytes of sufficient length

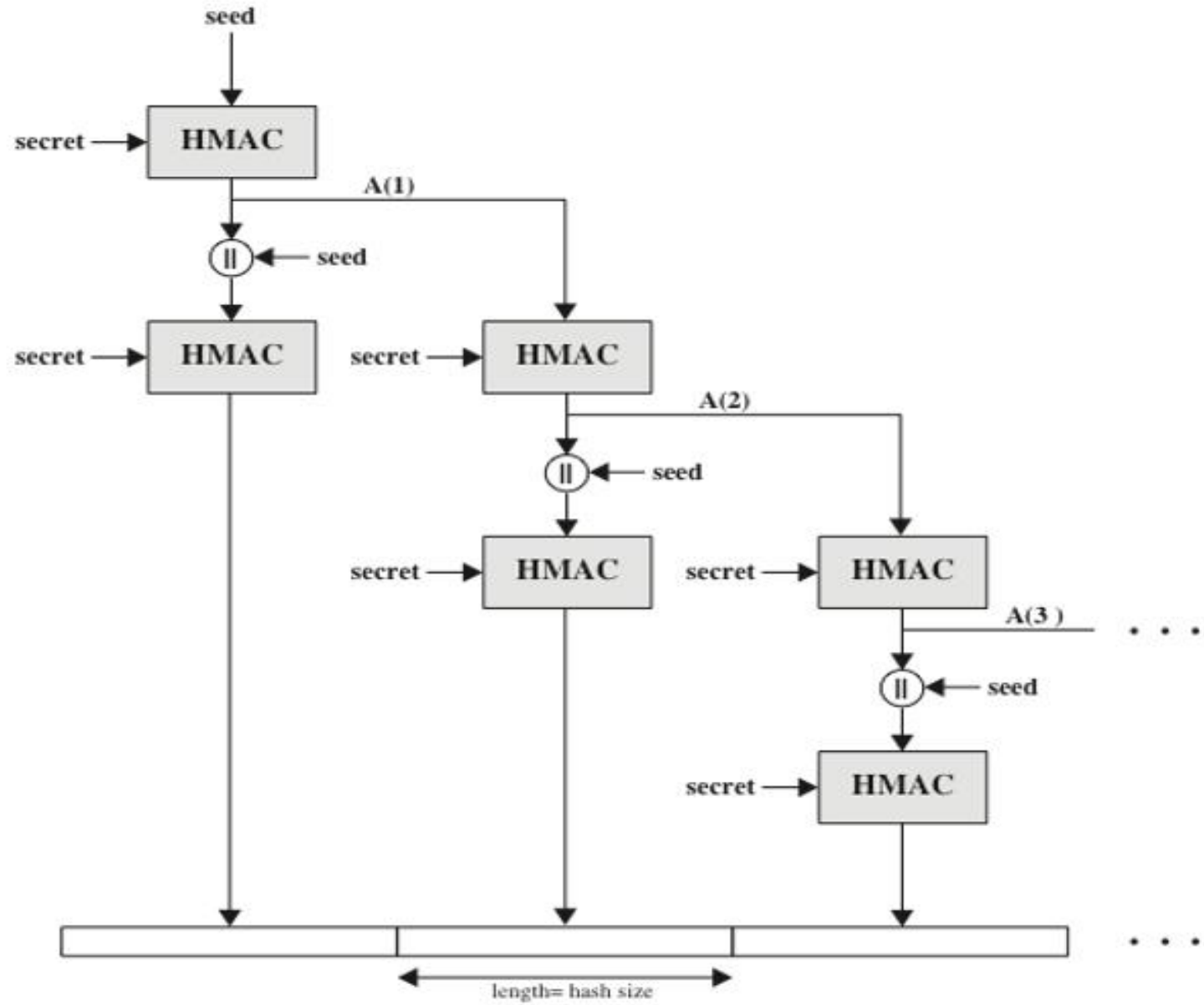


Figure 17.7 TLS Function P_hash (secret, seed)

Heartbeat Protocol

- Is a periodic signal generated by hardware or software to indicate normal operation or to synchronize other parts of a system
- Typically used to monitor the availability of a protocol entity
- In the specific case of TLS, a Heartbeat protocol was defined in 2012 in RFC 6250
  - Transport Layer Security (TLS)
  - Datagram Transport Layer Security (DTLS) Heartbeat Extentsion
- Runs on top of the TLS Record Protocol
- Consists of two message types
  - **heartbeat_request**
  - **heartbeat_response**
- The use of the Heartbeat protocol is established during Phase 1 of the Handshake protocol
- **The heartbeat serves two purposes**
  - It assures the sender that the recipient is still alive
  - The heartbeat generates activity across the connection during idle periods, which avoids closure by a firewall that does not tolerate idle connections
- Heartbeat protocol supports its use in a connectionless version of TLS known as Datagram Transport Layer Security (DTLS)

SSL/TLS Attacks

- The attacks can be grouped into four general categories:
  - **Attacks on the handshake protocol :** an approach to compromising the handshake protocol
  - **Attacks on the record and application data protocols**
  - **Attacks on the PKI :** Checking the validity of X.509 certificates is an activity subject to a variety of attacks
  - **Other attacks :** lists a number of attacks that do not fit into any of the preceding categories. Example : DoS attack

TLSv1.3

- Primary aim is to improve the security of TLS
- Significant changes from version 1.2 are:
  - TLSv1.3 removes support for a number of options and functions
    - Deleted items include:
      - Compression
      - Ciphers that do not offer authenticated encryption
      - Static RSA and DH key exchange
      - 32-bit timestamp as part of the Random parameter in the client_hello message
      - Renegotiation
      - Change Cipher Spec Protocol
      - RC4
      - Use of MD5 and SHA-224 hashes with signatures
  - TLSv1.3 uses Diffie-Hellman or Elleptic Curve Diffie-Hellman for key exchange and does not permit RSA
  - TLSv1.3 allows for a “1 round trip time” handshake by changing the order of message sent with establishing a secure connection

HTTPS (HTTP over SSL)

Connection Initiation

Connection Closure

- Refers to the combination of HTTP and SSL to **implement secure communication between a Web browser and a Web server**
- The HTTPS capability is built into all modern Web browsers
- A user of a Web browser will see URL addresses that begin with **https://** rather than http://
- If HTTPS is specified, port 443 is used, which invokes SSL
- Documented in RFC 2818, *HTTP Over TLS*
  - There is no fundamental change in using HTTP over either SSL or TLS and both implementations are referred to as HTTPS
- When HTTPS is used, the following elements of the communication are encrypted:
  - URL of the requested document
  - Contents of the document
  - Contents of browser forms
  - Cookies sent from browser to server and from server to browser
  - Contents of HTTP header

Connection Initiation

For HTTPS, the agent acting as the HTTP client also acts as the TLS client

The client initiates a connection to the server on the appropriate port and then sends the TLS ClientHello to begin the TLS handshake

When the TLS handshake has finished, the client may then initiate the first HTTP request

All HTTP data is to be sent as TLS application data

There are three levels of awareness of a connection in HTTPS:

At the HTTP level, an HTTP client requests a connection to an HTTP server by sending a connection request to the next lowest layer

- Typically the next lowest layer is TCP, but is may also be TLS/SSL

At the level of TLS, a session is established between a TLS client and a TLS server

- This session can support one or more connections at any time

A TLS request to establish a connection begins with the establishment of a TCP connection between the TCP entity on the client side and the TCP entity on the server side

Connection Closure

- An HTTP client or server can indicate the closing of a connection by including the line Connection: **close in an HTTP record**
- The closure of an HTTPS connection requires that TLS close the connection with the peer TLS entity on the remote side, which will involve closing the underlying TCP connection
- TLS implementations must initiate an exchange of closure alerts before closing a connection
  - A TLS implementation may, after sending a closure alert, close the connection without waiting for the peer to send its closure alert, generating an “incomplete close”
- An unannounced TCP closure could be evidence of some sort of attack so the HTTPS client should issue some sort of security warning when this occurs

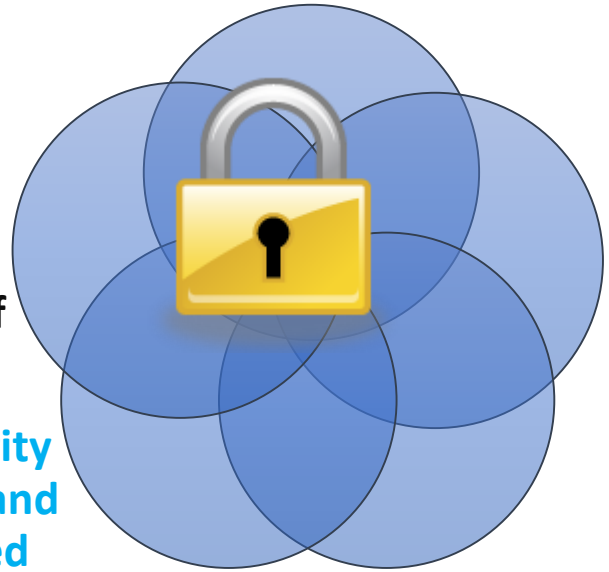
Secure Shell (SSH)

Transport Layer Protocol
User Authentication Protocol
Connection Protocol

A protocol for secure network communications designed to be relatively simple and inexpensive to implement

SSH client and server applications are widely available for most operating systems

- Has become the method of choice for remote login and X tunneling
- Is rapidly becoming one of the most pervasive applications for encryption technology outside of embedded systems



The initial version, SSH1 was focused on providing a secure remote logon facility to replace TELNET and other remote logon schemes that provided no security

SSH2 fixes a number of security flaws in the original scheme and is documented as a proposed standard in IETF RFCs 4250 through 4256

SSH also provides a more general client/server capability and can be used for such network functions as file transfer and e-mail

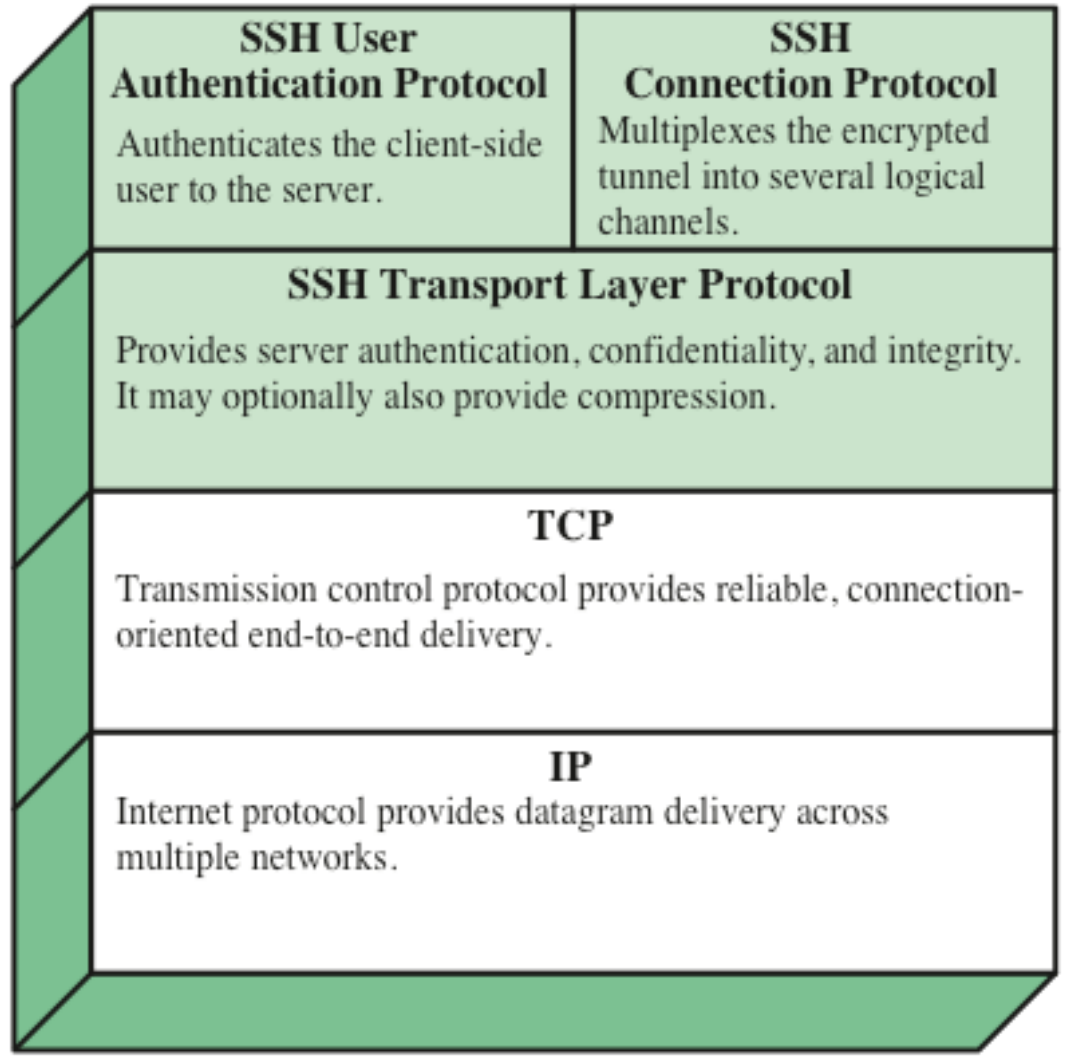


Figure 17.8 SSH Protocol Stack

Transport Layer Protocol

- Server authentication occurs at the transport layer, based on the server possessing a public/private key pair
- A server may have multiple host keys using multiple different asymmetric encryption algorithms
- Multiple hosts may share the same host key
- The server host key is used during key exchange to authenticate the identity of the host
- RFC 4251 dictates two alternative trust models:
  - The client has a local database that associates each host name with the corresponding public host key
  - The host name-to-key association is certified by a trusted certification authority (CA); the client only knows the CA root key and can verify the validity of all host keys certified by accepted CAs

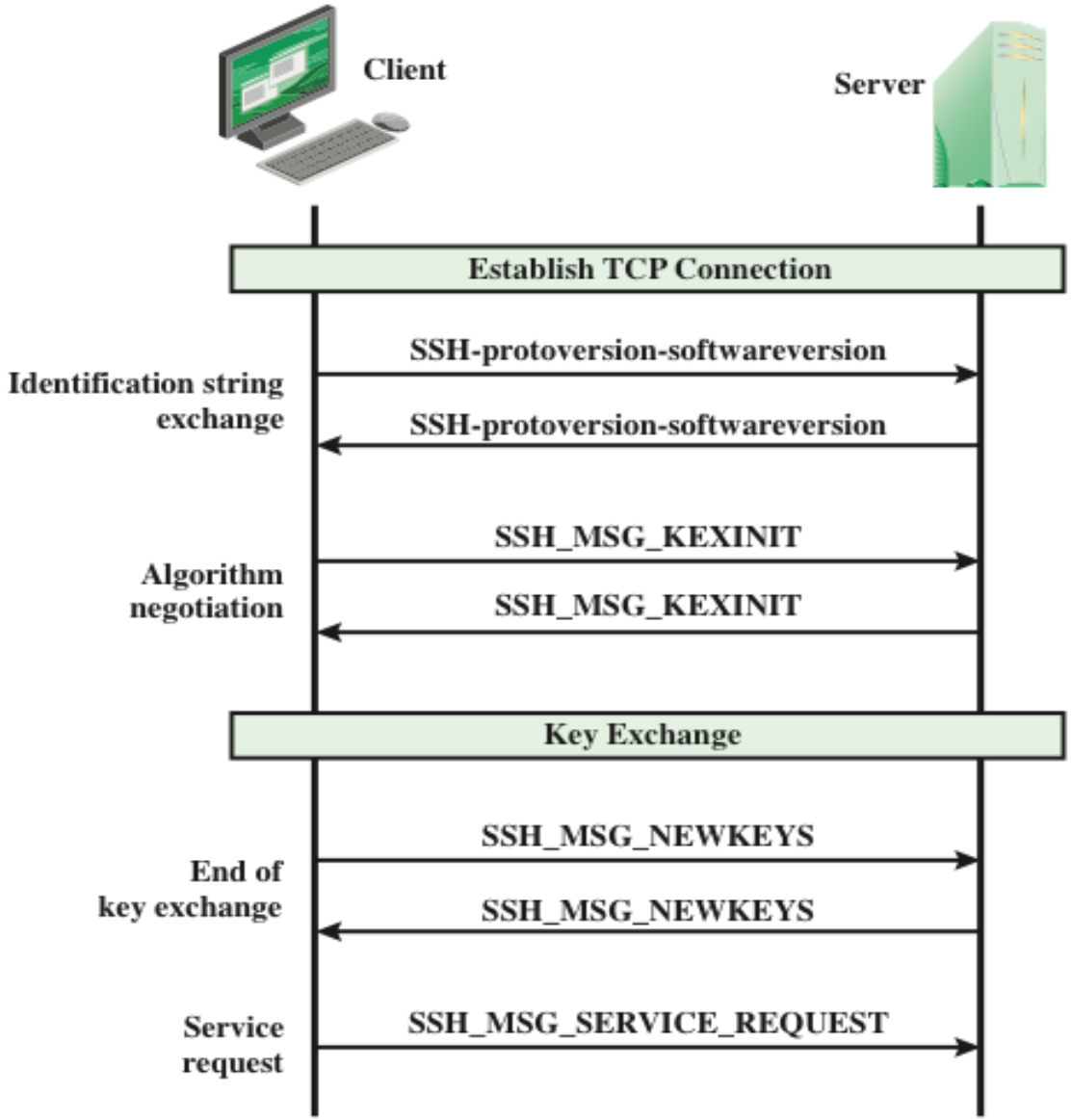


Figure 17.9 SSH Transport Layer Protocol Packet Exchanges

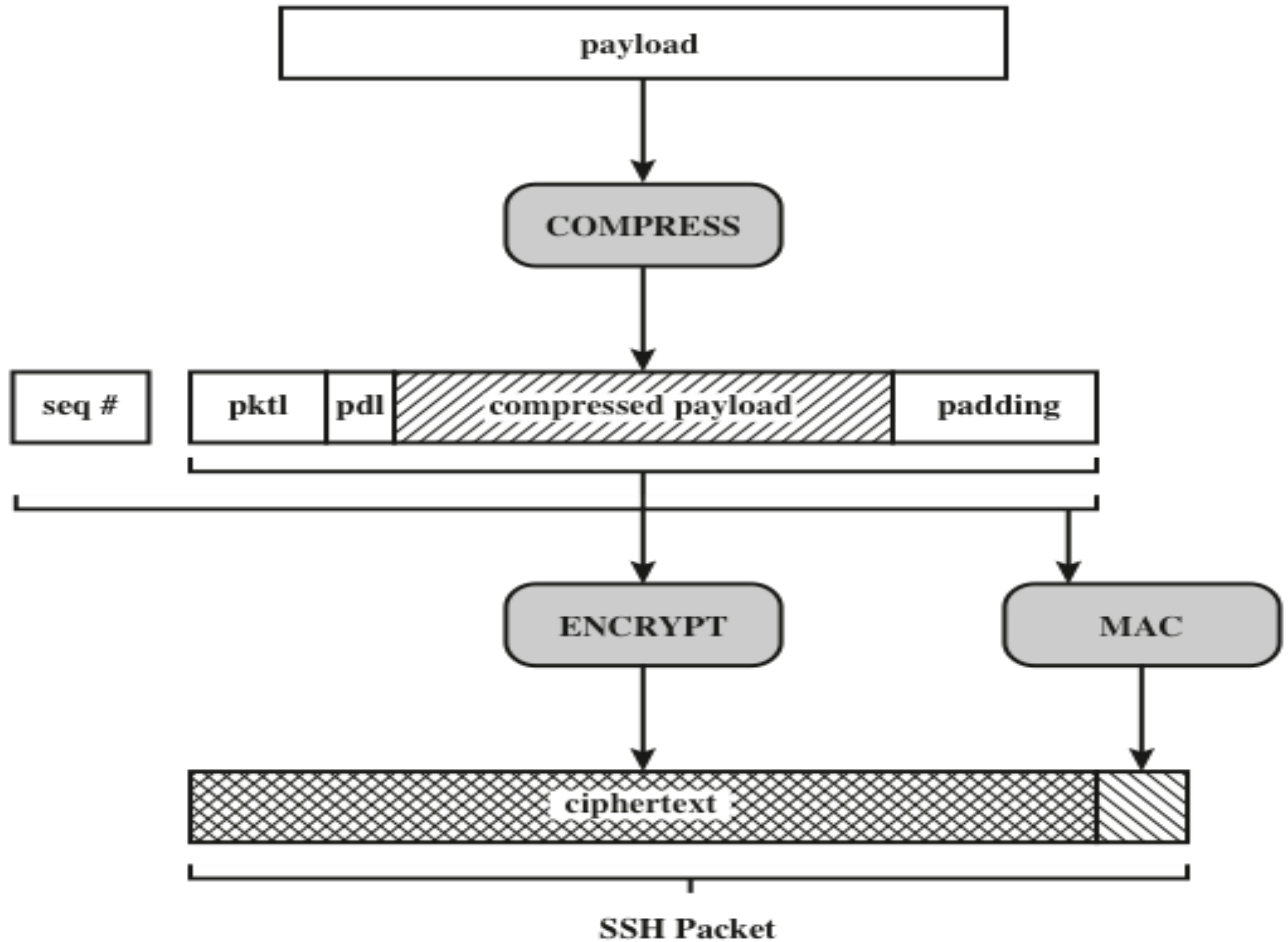


Figure 17.10 SSH Transport Layer Protocol Packet Formation

SSH Transport Layer Cryptographic Algorithms

Cipher	
3des-cbc*	Three-key 3DES in CBC mode
blowfish-cbc	Blowfish in CBC mode
twofish256-cbc	Twofish in CBC mode with a 256-bit key
twofish192-cbc	Twofish with a 192-bit key
twofish128-cbc	Twofish with a 128-bit key
aes256-cbc	AES in CBC mode with a 256-bit key
aes192-cbc	AES with a 192-bit key
aes128-cbc**	AES with a 128-bit key
Serpent256-cbc	Serpent in CBC mode with a 256-bit key
Serpent192-cbc	Serpent with a 192-bit key
Serpent128-cbc	Serpent with a 128-bit key
arcfour	RC4 with a 128-bit key
cast128-cbc	CAST-128 in CBC mode

MAC algorithm	
hmac-sha1*	HMAC-SHA1; digest length = key length = 20
hmac-sha1-96**	First 96 bits of HMAC-SHA1; digest length = 12; key length = 20
hmac-md5	HMAC-MD5; digest length = key length = 16
hmac-md5-96	First 96 bits of HMAC-MD5; digest length = 12; key length = 16

Compression algorithm	
none*	No compression
zlib	Defined in RFC 1950 and RFC 1951

Authentication Methods

- Publickey
  - The client sends a message to the server that contains the client's public key, with the message signed by the client's private key
  - When the server receives this message, it checks whether the supplied key is acceptable for authentication and, if so, it checks whether the signature is correct
- Password
  - The client sends a message containing a plaintext password, which is protected by encryption by the Transport Layer Protocol
- Hostbased
  - Authentication is performed on the client's host rather than the client itself
  - This method works by having the client send a signature created with the private key of the client host
  - Rather than directly verifying the user's identity, the SSH server verifies the identity of the client host

Connection Protocol

- The SSH Connection Protocol runs on top of the SSH Transport Layer Protocol and assumes that a secure authentication connection is in use
  - The secure authentication connection, referred to as a *tunnel,* is used by the Connection Protocol to multiplex a number of logical channels

- Channel mechanism
  - All types of communication using SSH are supported using separate channels
  - Either side may open a channel
  - For each channel, each side associates a unique channel number
  - Channels are flow controlled using a window mechanism
  - No data may be sent to a channel until a message is received to indicate that window space is available
  - The life of a channel progresses through three stages: opening a channel, data transfer, and closing a channel

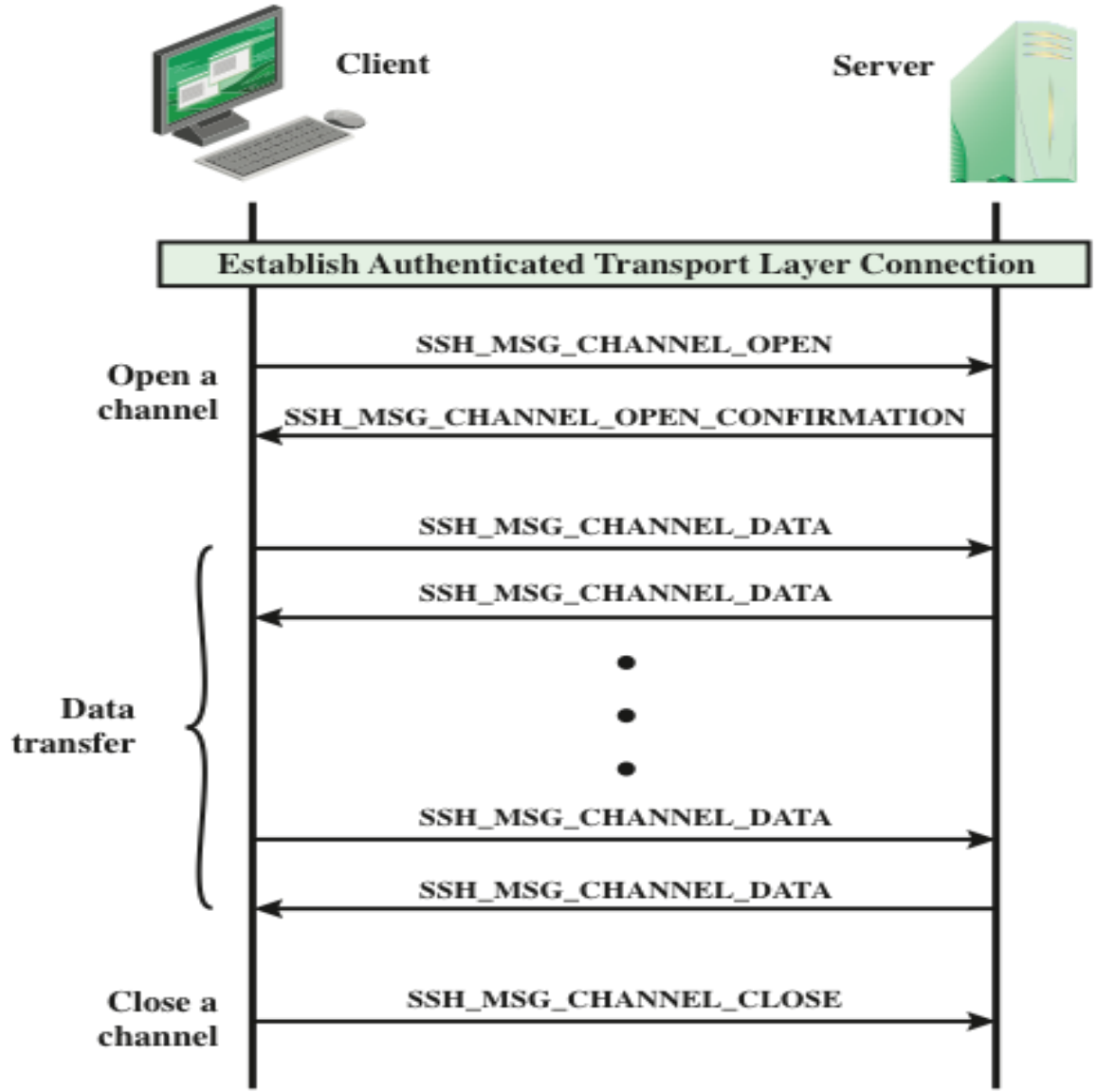


Figure 17.11 Example SSH Connection Protocol Message Exchange

Channel Types

- Four channel types are recognized in the SSH Connection Protocol specification

Session

- The remote execution of a program
- The program may be a shell, an application such as file transfer or e-mail, a system command, or some built-in subsystem
- Once a session channel is opened, subsequent requests are used to start the remote program

X11

- Refers to the X Window System, a computer software system and network protocol that provides a graphical user interface (GUI) for networked computers
- X allows applications to run on a network server but to be displayed on a desktop machine

Forwarded-tcpip

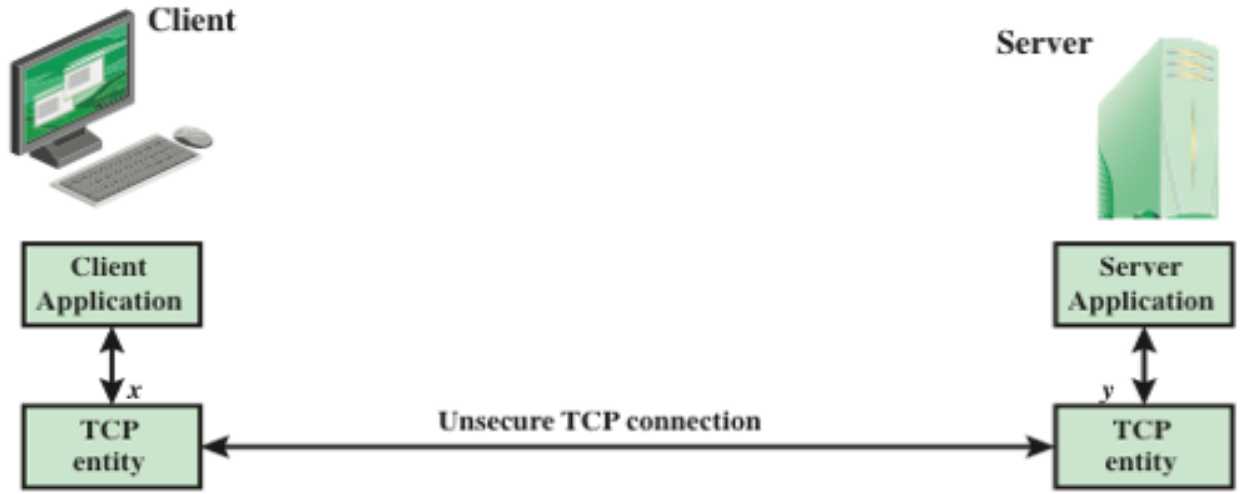
- Remote port forwarding

Direct-tcpip

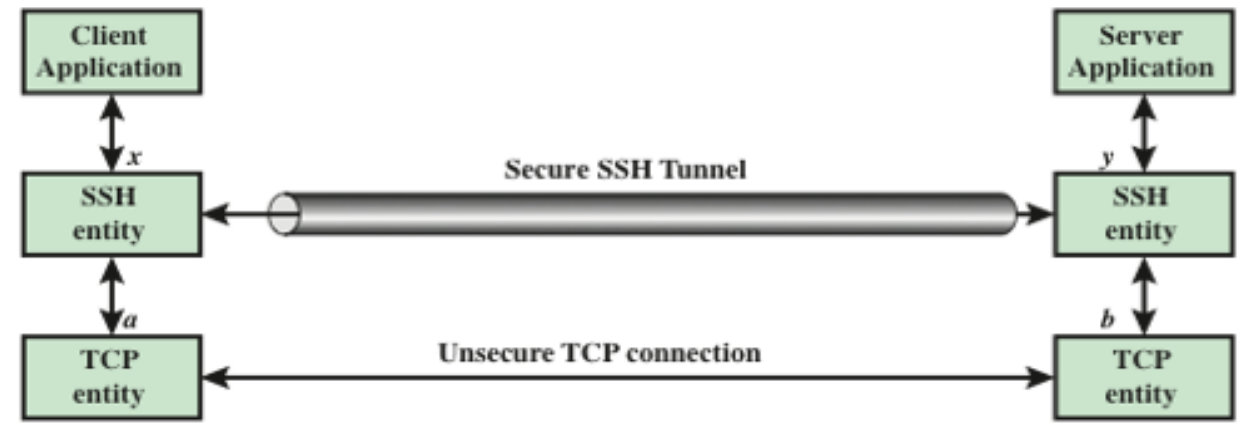
- Local port forwarding

Port Forwarding

- One of the most useful features of SSH
- Provides the ability to convert any insecure TCP connection into a secure SSH connection (also referred to as SSH tunneling)
- Incoming TCP traffic is delivered to the appropriate application on the basis of the port number (a port is an identifier of a user of TCP)
- An application may employ multiple port numbers



(a) Connection via TCP



(b) Connection via SSH Tunnel

Summary

- Web security considerations
  - Web security threats
  - Web traffic security approaches
- Secure sockets layer
  - SSL architecture
  - SSL record protocol
  - Change cipher spec protocol
  - Alert protocol
  - Handshake protocol
  - Cryptographic computations
  - Heartbeat protocol
  - SSL/TLS attacks
  - TLSv1.3

- Secure shell (SSH)
  - Transport layer protocol
  - User authentication protocol
  - Communication protocol
- HTTPS
  - Connection initiation
  - Connection closure