Authentication and E-Mail Security

Prof. A. A. Daptardar

HIT, Nidasoshi

X.509 Certificates

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

- X.509 was initially issued in 1988.
- The standard was subsequently revised to address some of the security concerns documented in [IANS90] and [MITC90];
- A revised recommendation was issued in 1993.
- A third version was issued in 1995 and revised in 2000.
- X.509 is based on the use of public-key cryptography and digital signatures.
- The digital signature scheme is assumed to require the use of a hash function.

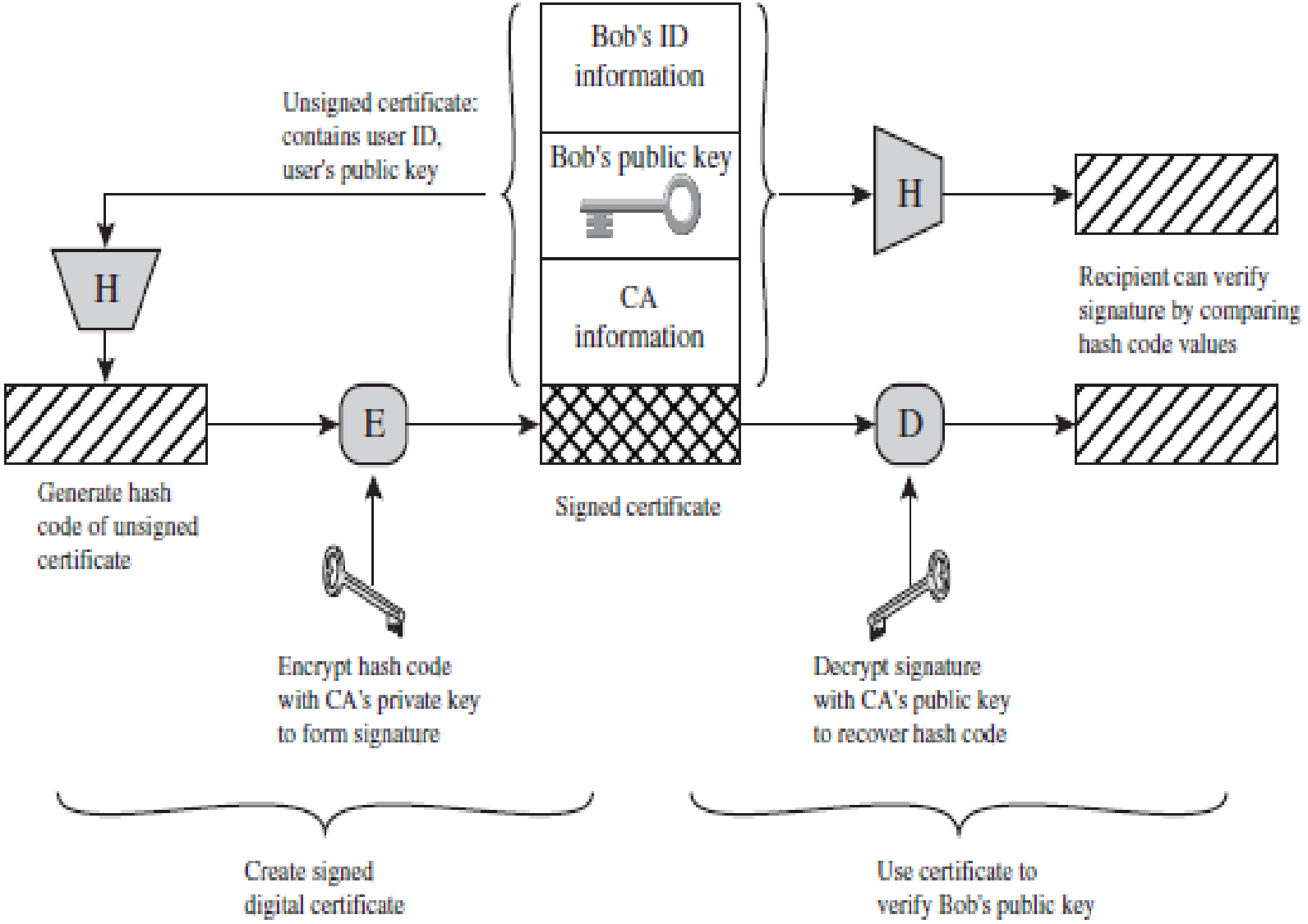
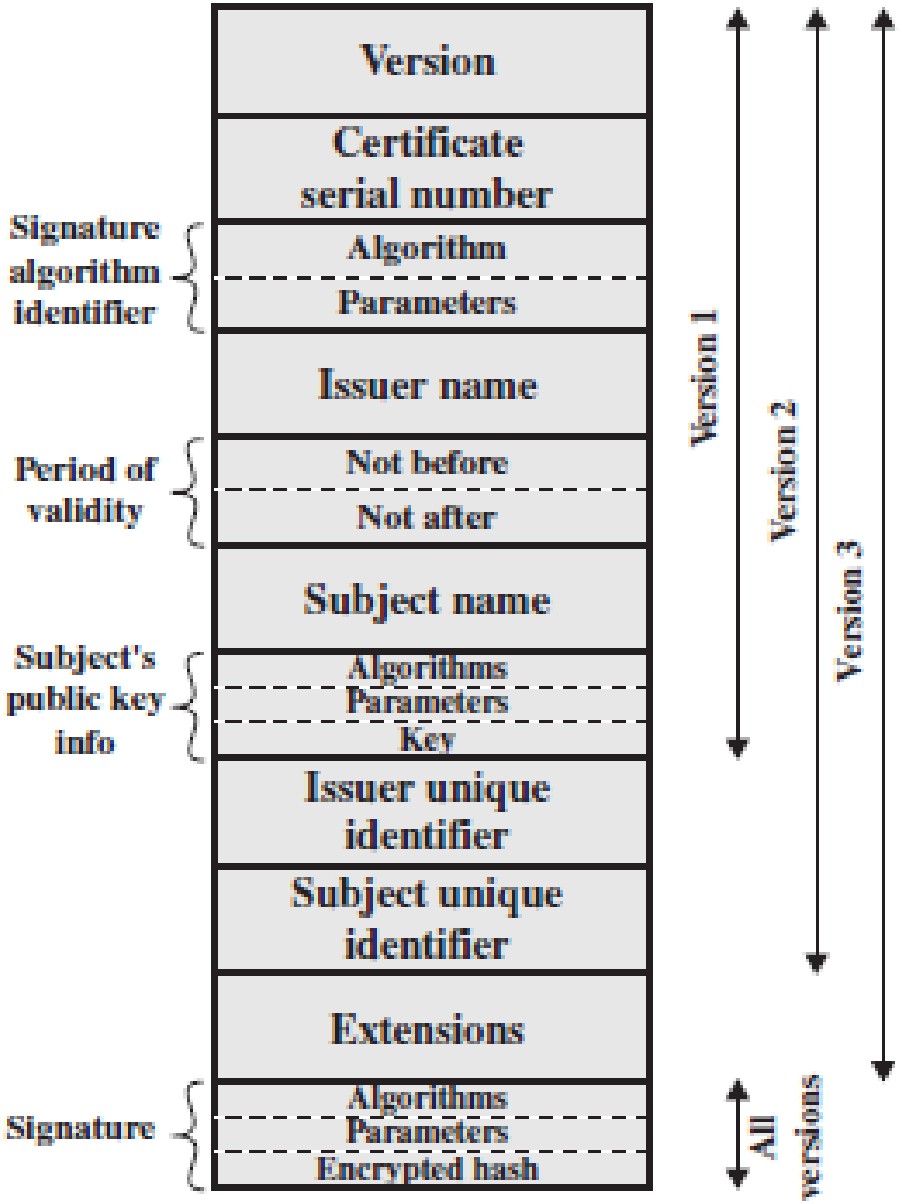


Figure 14.14 Public-Key Certificate Use

Certificates

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



(a) X.509 certificate

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

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

- The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time.
- These fields are rarely used.
- The standard uses the following notation to define a certificate:
- CA <<A>> = CA {V, SN, AI, CA, UCA, A, UA, Ap, T^A}
- where
- Y <<X>> = the certificate of user X issued by certification authority Y
- Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended
- V = version of the certificate
- SN = serial number of the certificate
- AI = identifier of the algorithm used to sign the certificate
- CA = name of certificate authority
- UCA = optional unique identifier of the CA
- A = name of user A
- UA = optional unique identifier of the user A
- Ap = public key of user A
- T^A = period of validity of the certificate

Obtaining a User's Certificate

- User certificates generated by a CA have the following characteristics:
  - Any user with access to the public key of the CA can verify the user public key that was certified.
  - No party other than the certification authority can modify the certificate without this being detected.
- Because certificates are unforgeable, they can be placed in a directory without the need for the directory to make special efforts to protect them.

- Now suppose that A has obtained a certificate from certification authority X1 and 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. 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 X2 signed by X1. Because A securely knows X1's public key, A can obtain X2's public key from its certificate and verify it by means of X1's signature on the certificate.
  - **Step 2 -** A then goes back to the directory and obtains the certificate of B signed by X2. Because A now has a trusted copy of X2'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

  X1 << X2 >> X2 << B >>

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

  X2 << X1 >> X1 << A >>

- 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

  X1 << X2 >> X2 << X3 >> ……X_N << *B* >>

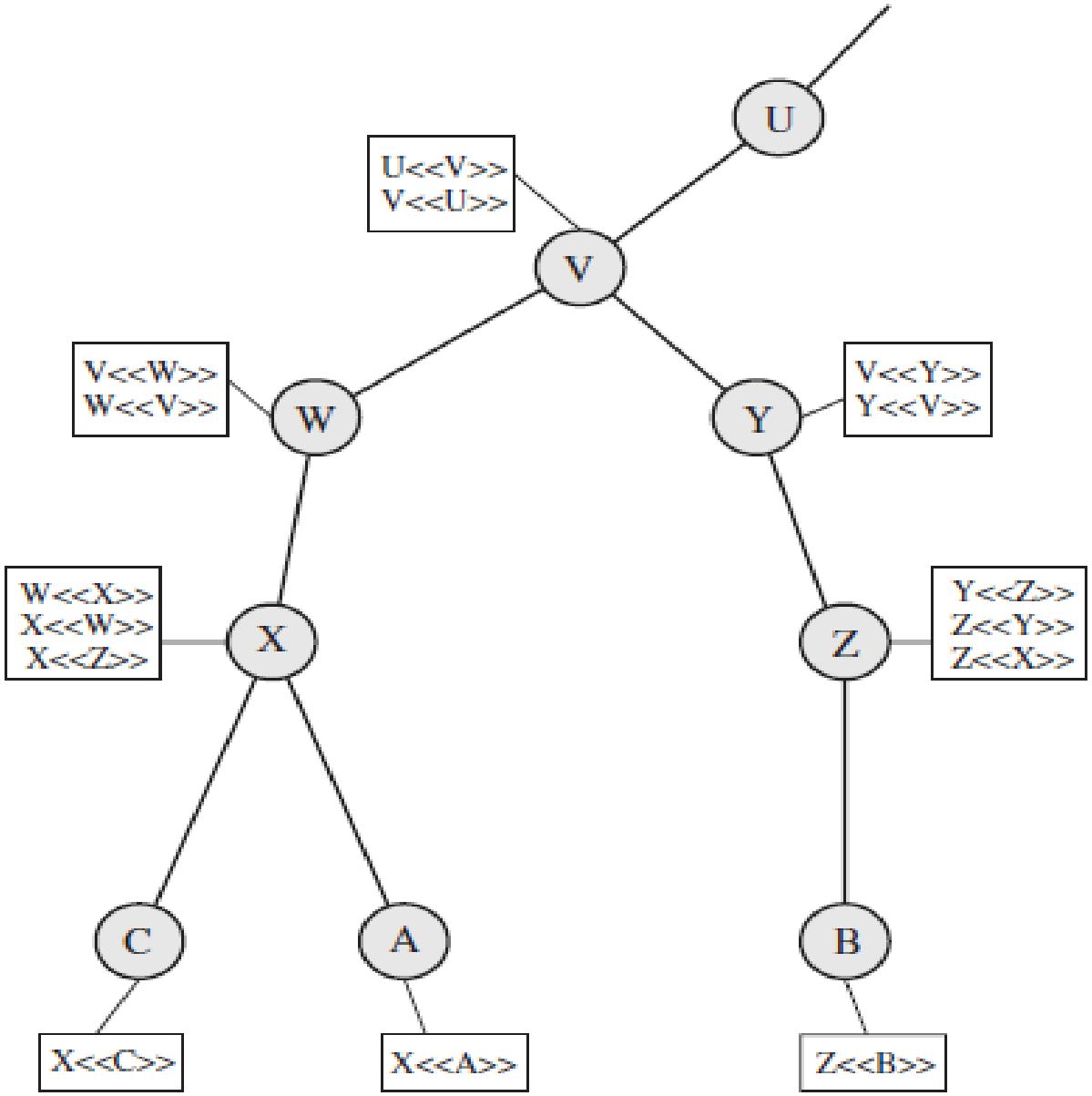


Figure 14.16 X.509 Hierarchy: A Hypothetical Example

- The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry.
- The directory entry for each CA includes two types of certificates:
- **Forward certificates:** Certificates of X generated by other CAs
- **Reverse certificates:** Certificates generated by X that are the certificates of other CAs

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

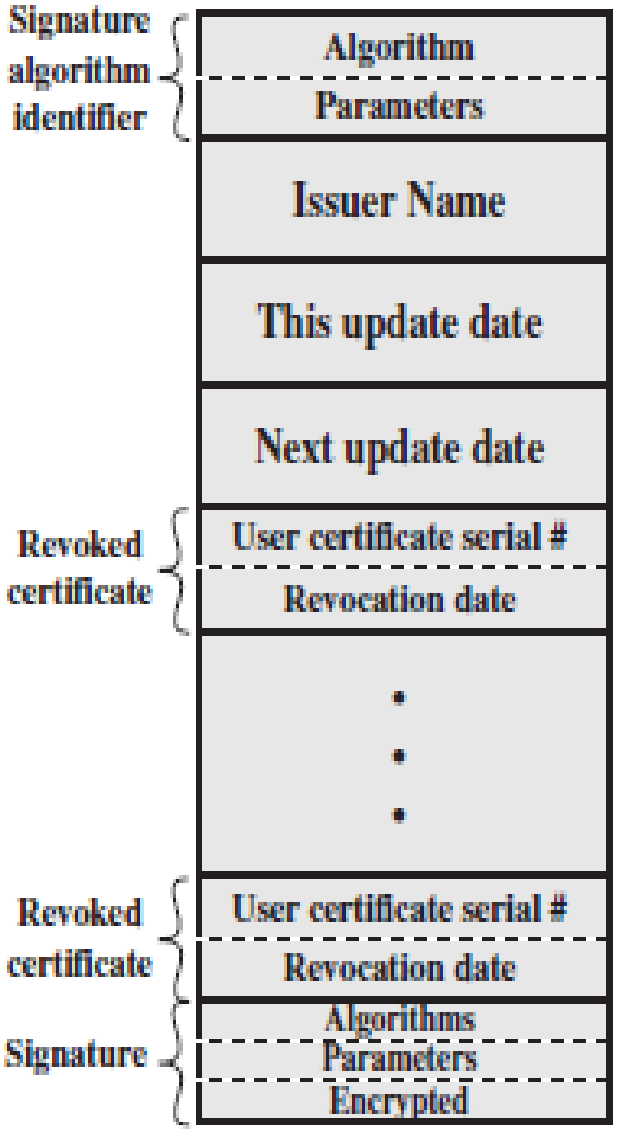
$$X \ll W \gg \; W \ll V \gg \; V \ll Y \gg \; Y \ll Z \gg \; Z \ll B \gg$$

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

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

Revocation of Certificates

- Each certificate includes a period of validity, much like a credit card.
- Typically, a new certificate is issued just before the expiration of the old one.
- In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons.
  - 1. The user's private key is assumed to be compromised.
  - 2. The user is no longer certified by this CA.
  - 3. The CA's certificate is assumed to be compromised.



(b) Certificate revocation list

- Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs.
- These lists should also be posted on the directory.
- Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 14.15b) the issuer's name, the date the list was created, the date the next CRL is scheduled to be issued, and an entry for each revoked certificate.
- Each entry consists of the serial number of a certificate and revocation date for that certificate.
- Because serial numbers are unique within a CA, the serial number is sufficient to identify the certificate.

X.509 Version 3

- The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2.
  - **1**. The subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.
  - **2.** The subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet-related identification.
  - **3.** There is a need to indicate security policy information. This enables a security application or function, such as IPSec, to relate an X.509 certificate to a given policy.
  - **4.** There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.
  - **5**. It is important to be able to identify different keys used by the same owner at different times.

- Version 3 includes a number of optional extensions that may be added to the version 2 format.
- Each extension consists of an extension identifier, a criticality indicator, and an extension value.
- The criticality indicator indicates whether an extension can be safely ignored.
- If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.
- The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information

- These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.
- A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements.

- This area includes:
  - **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL.
  - **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes.
  - **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.
  - **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. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
  - **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate Subject and Issuer Attributes

- These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject to increase a certificate user's confidence that the certificate subject is a particular person or entity.

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

Certification Path Constraints

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

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

Public-Key Infrastructure

- RFC 4949 (Internet Security Glossary) defines public-key infrastructure (PKI) as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.
- The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys.
- The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet.

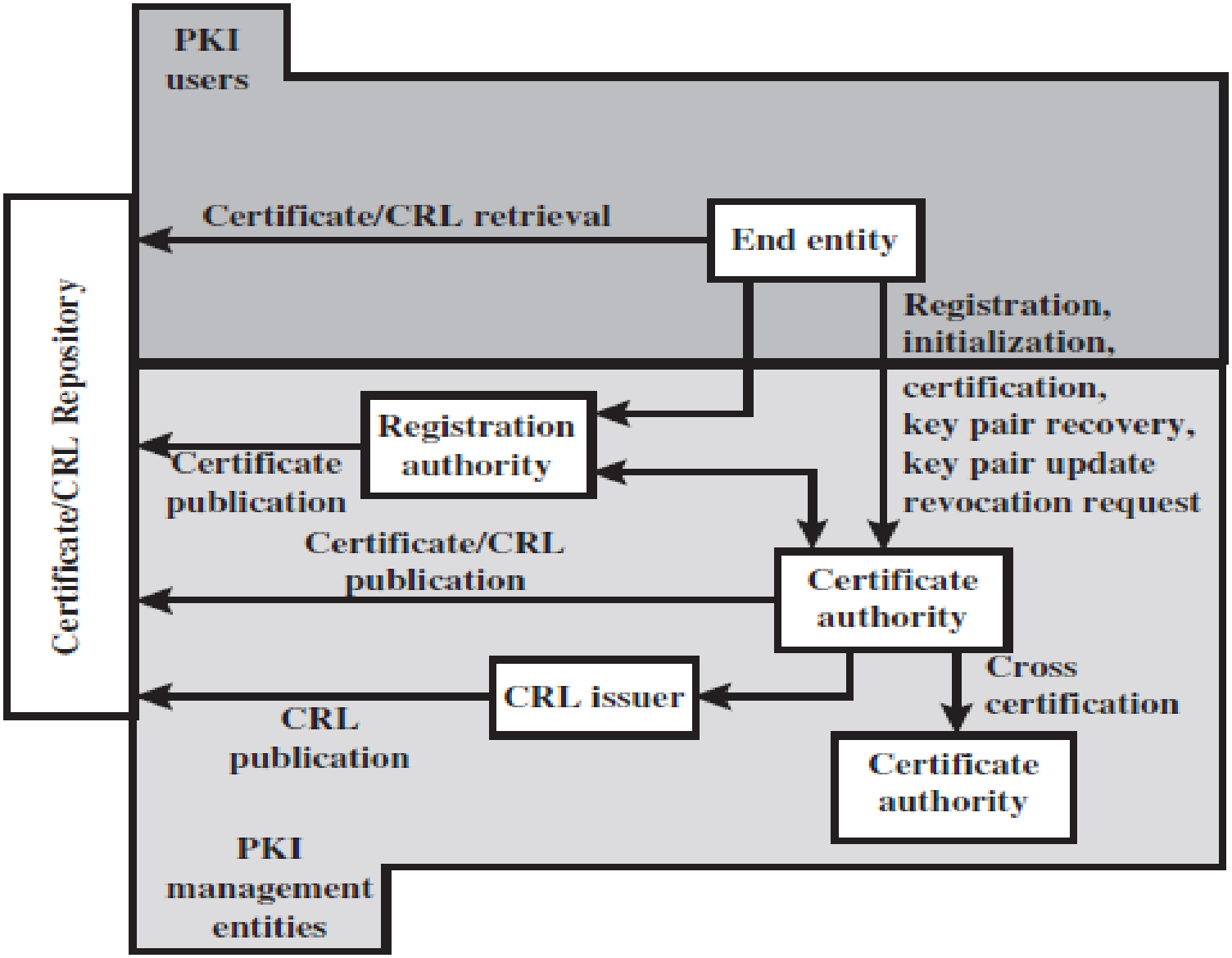


Figure 14.17 PKIX Architectural Model

- Figure 14.17 shows the interrelationship among the key elements of the PKIX model. These elements are
  - **End entity:** A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public-key certificate. End entities typically consume and/or support PKI-related services.
  - **Certification authority (CA):** The issuer of certificates and (usually) certificate revocation lists (CRLs). It may also support a variety of administrative functions, although these are often delegated to one or more Registration Authorities.
  - **Registration authority (RA):** An optional component that can assume a number of administrative functions from the CA. The RA is often associated with the end entity registration process but can assist in a number of other areas as well.
  - **CRL issuer:** An optional component that a CA can delegate to publish CRLs.
  - **Repository:** A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

PKIX Management Functions

- PKIX identifies a number of management functions that potentially need to be supported by management protocols. These are indicated in Figure 14.17 and include the following:
  - **Registration:** This is the process whereby a user first makes itself known to a CA (directly or through an RA), prior to that CA issuing a certificate or certificates for that user. Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.

- **Initialization:** Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure.
- **Certification:** This is the process in which a CA issues a certificate for a user's public key, returns that certificate to the user's client system, and/or posts that certificate in a repository.
- **Key pair recovery:** Key pairs can be used to support digital signature creation and verification, encryption and decryption, or both. When a key pair is used for encryption/decryption, it is important to provide a mechanism to recover the necessary decryption keys when normal access to the keying material is no longer possible, otherwise it will not be possible to recover the encrypted data.

- **Key pair update:** All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires and as a result of certificate revocation.
- **Revocation request:** An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.
- **Cross certification:** Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols

- The PKIX working group has defines two alternative management protocols between PKIX entities that support the management functions listed in the preceding subsection.
  - RFC 2510 defines the certificate management protocols (CMP). Within CMP, each of the management functions is explicitly identified by specific protocol exchanges. CMP is designed to be a flexible protocol able to accommodate a variety of technical, operational, and business models.
  - RFC 2797 defines certificate management messages over CMS (CMC), where CMS refers to RFC 2630, cryptographic message syntax. CMC is built on earlier work and is intended to leverage existing implementations.

User Authentication

Remote User-Authentication Principles

- In most computer security contexts, user authentication is the fundamental building block and the primary line of defense.
- User authentication is the basis for most types of access control and for user accountability.

- There are four general means of authenticating a user's identity, which can be used alone or in combination:
  - **Something the individual knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
  - **Something the individual possesses:** Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token.*
  - **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.
  - **Something the individual does (dynamic biometrics):** Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.

- Each method has problems.
  - An adversary may be able to guess or steal a password.
  - Similarly, an adversary may be able to forge or steal a token.
  - A user may forget a password or lose a token.
  - Furthermore, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems.
  - With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance, cost, and convenience.
  - For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Mutual Authentication

- Mutual Authentication Protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.
- Central to the problem of authenticated key exchange are two issues:
  - confidentiality
  - timeliness

- To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form.
- This requires the prior existence of secret or public keys that can be used for this purpose.
- The second issue, timeliness, is important because of the threat of message replays.
- Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party.

- The following examples of replay attacks:
- 1. The simplest replay attack is one in which the opponent simply copies a message and replays it later.
- 2. An opponent can replay a timestamped message within the valid time window. If both the original and the replay arrive within then time window, this incident can be logged.
- 3. As with example (2), an opponent can replay a timestamped message within the valid time window, but in addition, the opponent suppresses the original message. Thus, the repetition cannot be detected.
- 4. Another attack involves a backward replay without modification. This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

- One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange.
- A new message is accepted only if its sequence number is in the proper order.
- The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with.

- Instead, one of the following two general approaches is used:
  - Timestamps: Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
  - • Challenge/response: Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

One-Way Authentication

- One application for which encryption is growing in popularity is electronic mail (e-mail).
- It is not necessary for the sender and receiver to be online at the same time.
- Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.
- The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400.
- A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

Remote User-Authentication Using Symmetric Encryption

- Mutual Authentication
- One-Way Authentication

Mutual Authentication

- The proposal is put by Needham and Schroeder for secret key distribution using KDC.
- The protocol can be summarized as follows

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

- The purpose of the protocol is to distribute securely a session key Ks to A and B.
- 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.
- Her proposal assumes that the master keys, Ka and Kb, are secure, and it consists of the following steps.

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

- T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange.
- A and B can verify timeliness by checking that
- $| \text{Clock} - T | < \Delta t1 + \Delta t2$
- where $\Delta t1$ is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and $\Delta t2$ is the expected network delay time.

- The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol.
- However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network.
- The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.

- 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 counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock.
- The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces.
- The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

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

- 1. A initiates the authentication exchange by generating a nonce, Na, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.
- 2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, Nb. This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness.

- B’s message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.
- 3. The KDC passes on to A B’s nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a “ticket” that can be used by A for subsequent authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC.

- This block verifies that B has received A's initial message (IDB) and that this is a timely message and not a replay (Na), and it provides A with a session key (Ks) and the time limit on its use (Tb).
- 4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt E(Ks, Nb) to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

- This protocol provides an effective, secure means for A and B to establish a session with a secure session key.
- Furthermore, 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.

One-Way Authentication

- 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 e-mail process, such timestamps may have limited usefulness.

Kerberos

- Kerberos is an authentication service developed as part of Project Athena at MIT.
- The problem that Kerberos addresses is this:
- Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network.
- We would like for servers to be able to restrict access to authorized users and to be able to authenticate requests for service.
- In this environment, a workstation cannot be trusted to identify its users correctly to network services.

- In particular, the following three threats exist:
  - 1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
  - 2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
  - 3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

- Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users.
- Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.
- Two versions of Kerberos are in common use.
  - Version 4
  - Version 5

Motivation

- If a set of users is provided with dedicated personal computers that have no network connections, then a user's resources and files can be protected by physically securing each personal computer.
- When these users instead are served by a centralized time-sharing system, the time-sharing operating system must provide the security.
- The operating system can enforce access-control policies based on user identity and use the logon procedure to identify users.

- More common is a distributed architecture consisting of dedicated user workstations (clients) and distributed or centralized servers.
- In this environment, three approaches to security can be envisioned.
  - 1. Rely on each individual client workstation to assure the identity of its user or users and rely on each server to enforce a security policy based on user identification (ID).
  - 2. Require that client systems authenticate themselves to servers, but trust the client system concerning the identity of its user.
  - 3. Require the user to prove his or her identity for each service invoked. Also require that servers prove their identity to clients.

Kerberos Requirements

- **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.
- **Reliable:** For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services.
- **Transparent:** Ideally, 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. This suggests a modular, distributed architecture.

Kerberos Version 4

- Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service.
  - **A Simple Authentication Dialogue**
  - **A More Secure Authentication Dialogue**
  - **The Version 4 Authentication Dialogue**

A Simple Authentication Dialogue

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

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

$$\begin{aligned}
&\textbf{(1) C} \rightarrow \textbf{AS:} \quad ID_C \| P_C \| ID_V \\
&\textbf{(2) AS} \rightarrow \textbf{C:} \quad Ticket \\
&\textbf{(3) C} \rightarrow \textbf{V:} \quad ID_C \| Ticket \\
&Ticket = E(K_v, [ID_C \| AD_C \| ID_V])
\end{aligned}$$

where

C = client

AS = authentication server

V = server

ID_C = identifier of user on C

ID_V = identifier of V

P_C = password of user on C

AD_C = network address of C

K_v = secret encryption key shared by AS and V

- In this scenario, the user logs on to a workstation and requests access to server V.
- The client module C in the user's workstation requests the user's password and then sends a message to the AS that includes the user's ID, the server's ID, and the user's password.
- The AS checks its database to see if the user has supplied the proper password for this user ID and whether this user is permitted access to server V.
- If both tests are passed, the AS accepts the user as authentic and must now convince the server that this user is authentic.
- To do so, the AS creates a **ticket** that contains the user's ID and network address and the server's ID.

- This ticket is then sent back to C.
- Because the ticket is encrypted, it cannot be altered by C or by an opponent.
- With this ticket, C can now apply to V for service. C sends a message to V containing C's ID and the ticket.
- V decrypts the ticket and verifies that the user ID in the ticket is the same as the unencrypted user ID in the message.
- If these two match, the server considers the user authenticated and grants the requested service.

Problems

- Two problems arises :
  - We would like to minimize the number of times that a user has to enter a password.
  - The scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

A More Secure Authentication Dialogue

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

Once per user logon session:

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

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

Once per type of service:

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

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

Once per service session:

(5) $C \rightarrow V$: $ID_C \| Ticket_v$

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

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

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

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

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

Problems

- Two problems arises :
  - The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greater opportunity for replay.
  - there may be a requirement for servers to Authenticate themselves to users. Without such authentication, an opponent could sabotage the configuration so that messages to a server were directed to another location. The false server would then be in a position to act as a real server and capture any information from the user and deny the true service to the user.

The Version 4 Authentication Dialogue

Table 15.1 Summary of Kerberos Version 4 Message Exchanges

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

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

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

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

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

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

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

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

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

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

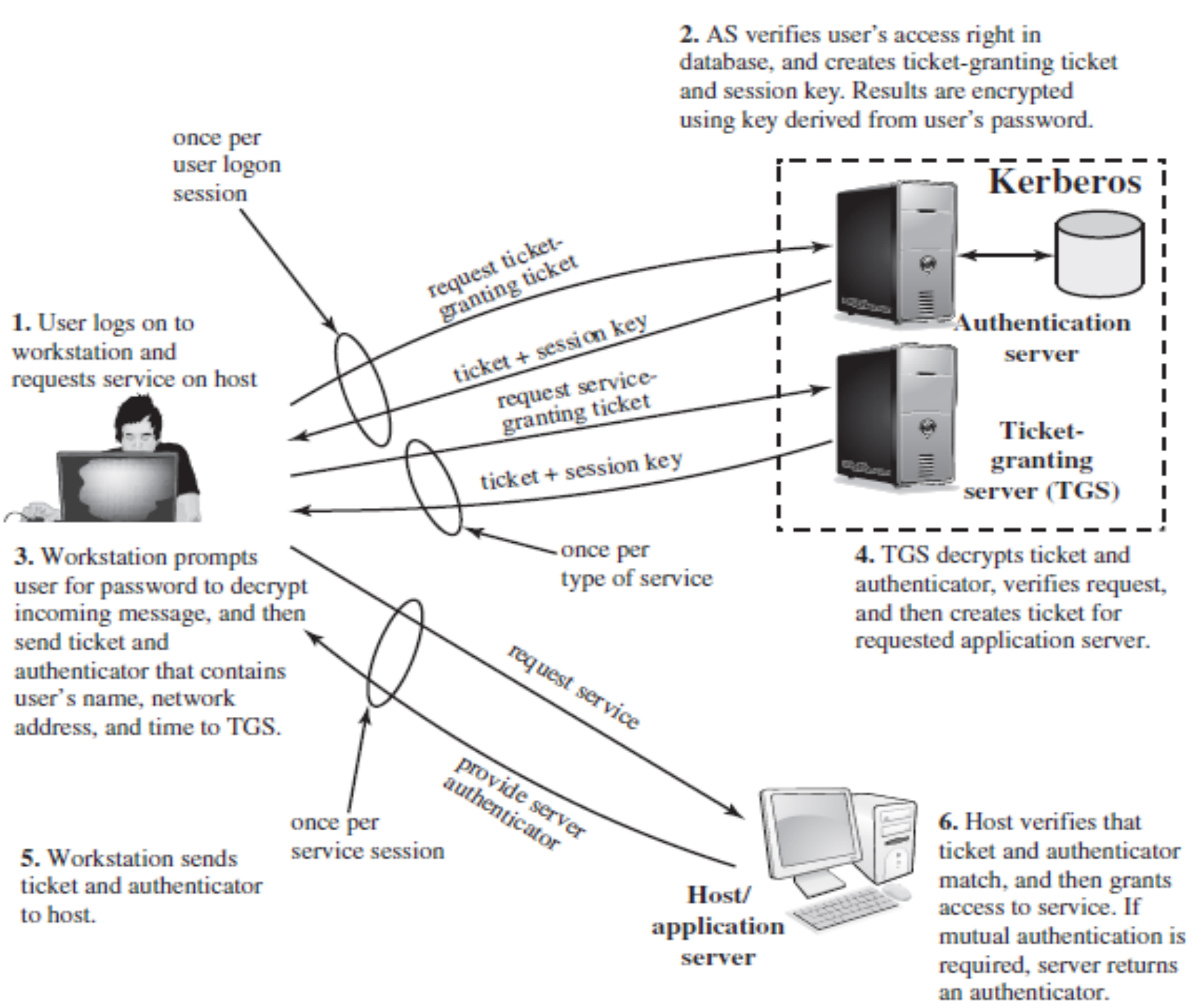
(5) $\mathbf{C \rightarrow V}$ $Ticket_v \| Authenticator_c$

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

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

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

(c) Client/Server Authentication Exchange to obtain service


2. AS verifies user's access right in database, and creates ticket-granting ticket and session key. Results are encrypted using key derived from user's password.
once per user logon session
Kerberos
request ticket-granting ticket
ticket + session key
Authentication server
1. User logs on to workstation and requests service on host
request service-granting ticket
ticket + session key
Ticket-granting server (TGS)
once per type of service
4. TGS decrypts ticket and authenticator, verifies request, and then creates ticket for requested application server.
3. Workstation prompts user for password to decrypt incoming message, and then send ticket and authenticator that contains user's name, network address, and time to TGS.
request service
provide server authenticator
once per service session
5. Workstation sends ticket and authenticator to host.
Host/ application server
6. Host verifies that ticket and authenticator match, and then grants access to service. If mutual authentication is required, server returns an authenticator.

Kerberos Exchange

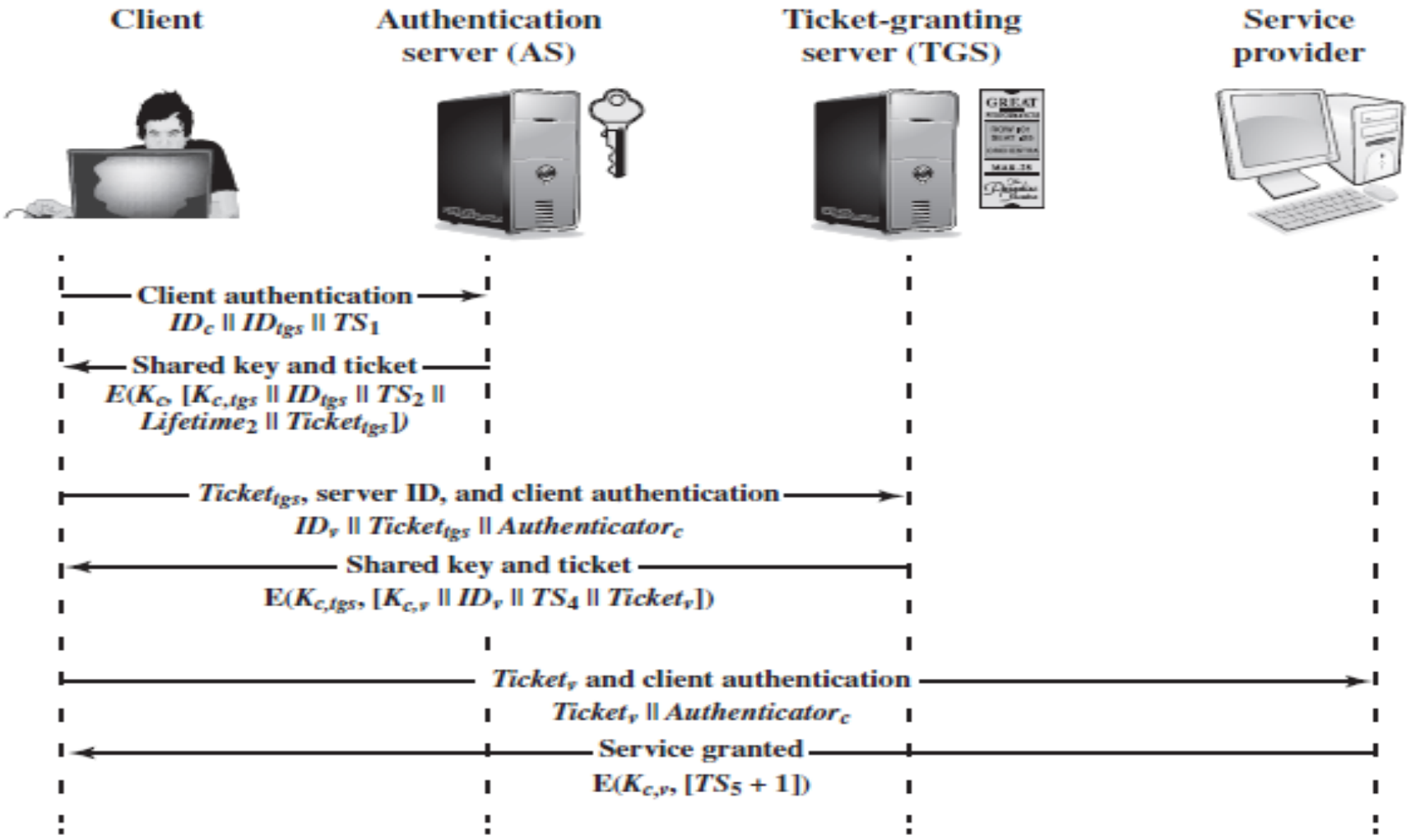


Figure 15.2 Kerberos Exchanges

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

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,\,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,\,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.
$K_{c,\,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

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

(c) Client/Server Authentication Exchange

Kerberos Realms and Multiple Kerberi

- A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:
  - 1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
  - 2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.
- Such an environment is referred to as a **Kerberos realm.**

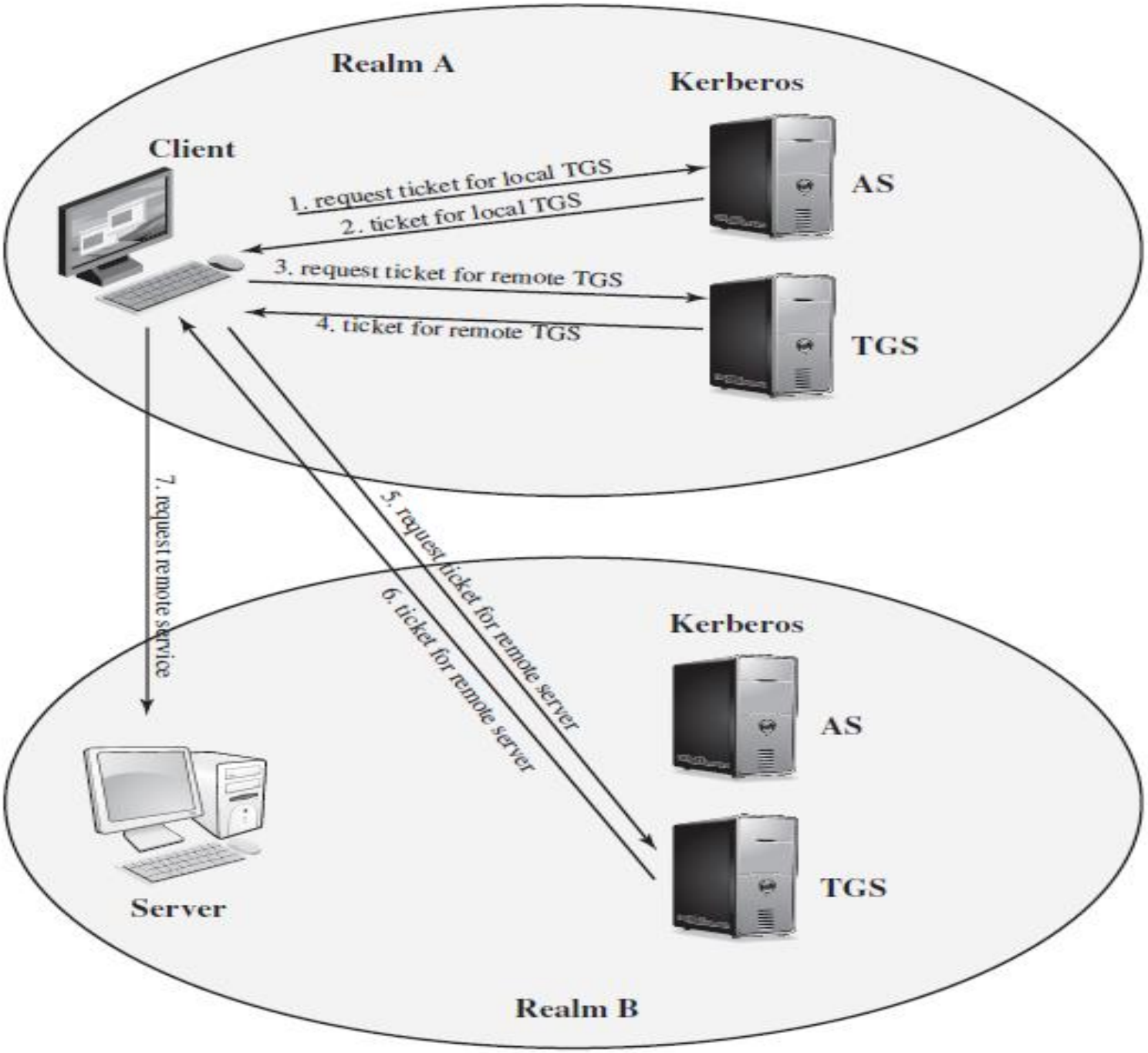


Figure 15.3 Request for Service in Another Realm

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

- A user wishing service on a server in another realm needs a ticket for that server. The user's client follows the usual procedures to gain access to the local TGS and then requests a ticket-granting ticket for a remote TGS (TGS in another realm). The client can then apply to the remote TGS for a service-granting ticket for the desired server in the realm of the remote TGS.

Differences between Versions 4 and 5

- Version 5 is intended to address the limitations of version 4 in two areas:
  - environmental shortcomings
  - technical deficiencies.

Environmental Shortcomings

1. Encryption system dependence
2. Internet protocol dependence
3. Message byte ordering
4. Ticket lifetime
5. Authentication forwarding
6. Interrealm authentication

Technical Deficiencies

1. Double encryption
2. PCBC encryption
3. Session keys
4. Password attacks

The Version 5 Authentication Dialogue

Table 15.3 Summary of Kerberos Version 5 Message Exchanges

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

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

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

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

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

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

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

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

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

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

(5) $\mathbf{C} \rightarrow \mathbf{V}$ $Options \| Ticket_v \| Authenticator_c$

(6) $\mathbf{V} \rightarrow \mathbf{C}$ $\mathrm{E}_{K_{c,v}}[TS_2 \| Subkey \| Seq\ \#]$

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

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

(c) Client/Server Authentication Exchange to obtain service

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.