MODULE 4

SYLLABUS:

X-509 certificates. Certificates, X-509 version 3 Public key infrastructure.

User Authentication: Remote user Authentication principles, Mutual Authentication, one-way authentication, remote user Authentication using Symmetric encryption, Mutual Authentication, one-way Authentication.

Kerberos: Motivation, Kerberos version 4, Kerberos version 5, Remote user Authentication using Asymmetric encryption, Mutual Authentication, one-way Authentication.

14.4 X.509 CERTIFICATES

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service. The directory is, in effect, a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates of the type discussed in Section 14.3. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

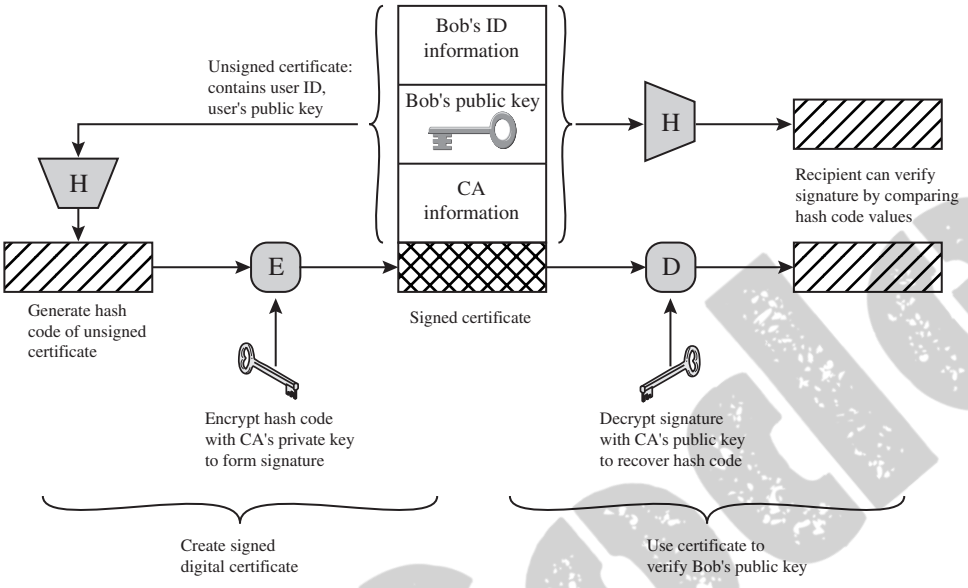

Figure 14.14 Public-Key Certificate Use

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME (Chapter 19), IP Security (Chapter 20), and SSL/TLS (Chapter 17).

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 standard does not dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation included the description of a recommended hash algorithm; this algorithm has since been shown to be insecure and was dropped from the 1993 recommendation. Figure 14.14 illustrates the generation of a public-key certificate.

Certificates

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

Figure 14.15a shows the general format of a certificate, which includes the following elements.

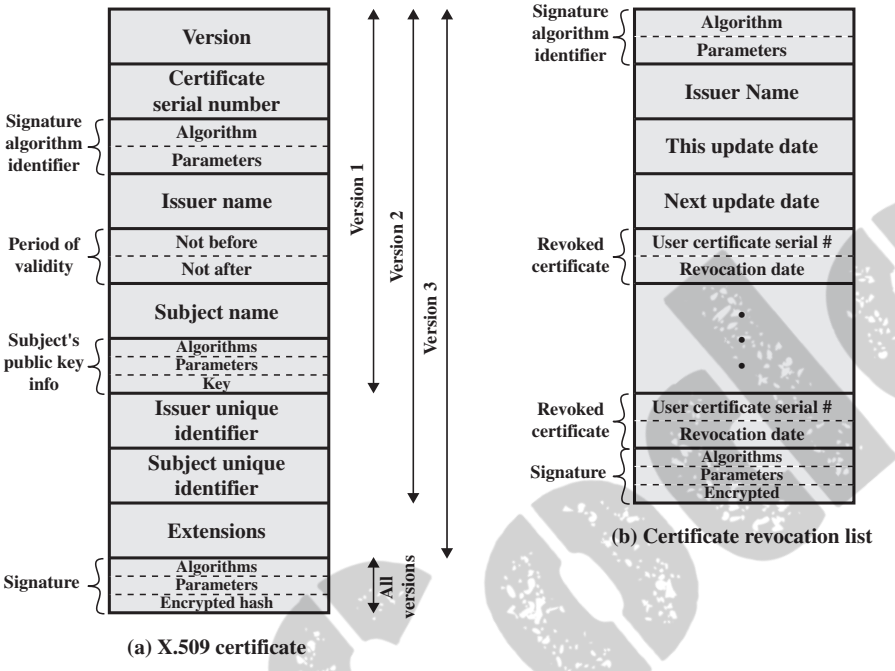

Figure 14.15 X.509 Formats

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

- **Subject unique identifier:** An optional-bit string field used to identify uniquely the subject in the event the X.500 name has been reused for different entities.
- **Extensions:** A set of one or more extension fields. Extensions were added in version 3 and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of the other fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used.

The standard uses the following notation to define a certificate:

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

where

$Y \ll X \gg$ = the certificate of user X issued by certification authority Y
$Y \{I\}$ = the signing of I by Y. It consists of I with an encrypted hash code appended
V = version of the certificate
SN = serial number of the certificate
AI = identifier of the algorithm used to sign the certificate
CA = name of certificate authority
UCA = optional unique identifier of the CA
A = name of user A
UA = optional unique identifier of the user A
Ap = public key of user A
T^A = period of validity of the certificate

The CA signs the certificate with its private key. If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid. This is the typical digital signature approach illustrated in Figure 13.2.

Obtaining a User's Certificate User certificates generated by a CA have the following characteristics:

- Any user with access to the public key of the CA can verify the user public key that was certified.
- No party other than the certification authority can modify the certificate without this being detected.

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

If all users subscribe to the same CA, then there is a common trust of that CA. All user certificates can be placed in the directory for access by all users. In addition, a user can transmit his or her certificate directly to other users. In either case, once B is in possession of A's certificate, B has confidence that messages it encrypts

with A's public key will be secure from eavesdropping and that messages signed with A's private key are unforgeable.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2. If A does not securely know the public key of X_2, then B's certificate, issued by X_2, is useless to A. A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key.

Step 1 A obtains from the directory the certificate of X_2 signed by X_1. Because A securely knows X_1's public key, A can obtain X_2's public key from its certificate and verify it by means of X_1's signature on the certificate.

Step 2 A then goes back to the directory and obtains the certificate of B signed by X_2. Because A now has a trusted copy of X_2's public key, A can verify the signature and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

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

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

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

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

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

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

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 14.16, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

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

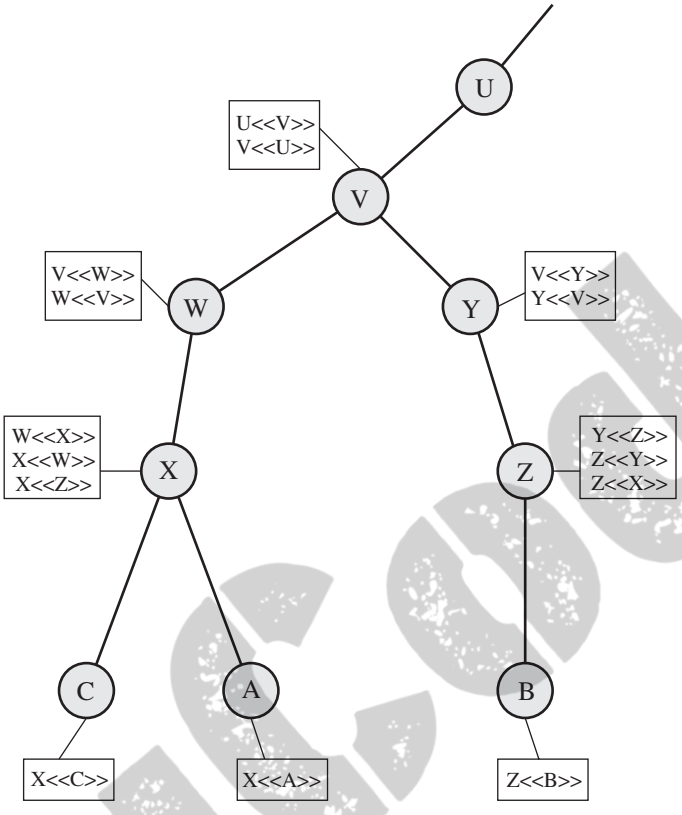

Figure 14.16 X.509 Hierarchy: A Hypothetical Example

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

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

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

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

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.

Revocation of Certificates Recall from Figure 14.15 that each certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. In addition, it may be desirable on occasion to revoke a certificate before it expires, for one of the following reasons.

1. The user's private key is assumed to be compromised.
2. The user is no longer certified by this CA. Reasons for this include that the subject's name has changed, the certificate is superseded, or the certificate was not issued in conformance with the CA's policies.
3. The CA's certificate is assumed to be compromised.

Each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. These lists should also be posted on the directory.

Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (Figure 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.

When a user receives a certificate in a message, the user must determine whether the certificate has been revoked. The user could check the directory each time a certificate is received. To avoid the delays (and possible costs) associated with directory searches, it is likely that the user would maintain a local cache of certificates and lists of revoked certificates.

X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2.

1. The subject field is inadequate to convey the identity of a key owner to a public-key user. X.509 names may be relatively short and lacking in obvious identification details that may be needed by the user.
2. The subject field is also inadequate for many applications, which typically recognize entities by an Internet 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. This feature supports key lifecycle management: in particular, the ability to update key pairs for users and CAs on a regular basis or under exceptional circumstances.

Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. Thus, version 3 includes a number of optional extensions that may be added to the version 2 format. Each extension consists of an extension identifier, a criticality indicator, and an extension value. The criticality indicator indicates whether an extension can be safely ignored. If the indicator has a value of TRUE and an implementation does not recognize the extension, it must treat the certificate as invalid.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

Key and Policy Information These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy. A certificate policy is a named set of rules that indicates the applicability of a certificate to a particular community and/or class of application with common security requirements. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes:

- **Authority key identifier:** Identifies the public key to be used to verify the signature on this certificate or CRL. Enables distinct keys of the same CA to be differentiated. One use of this field is to handle CA key pair updating.
- **Subject key identifier:** Identifies the public key being certified. Useful for subject key pair updating. Also, a subject may have multiple key pairs and, correspondingly, different certificates for different purposes (e.g., digital signature and encryption key agreement).
- **Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used. May indicate one or more of the following: digital signature, nonrepudiation, key encryption, data encryption, key agreement, CA signature verification on certificates, CA signature verification on CRLs.
- **Private-key usage period:** Indicates the period of use of the private key corresponding to the public key. Typically, the private key is used over a different period from the validity of the public key. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.
- **Certificate policies:** Certificates may be used in environments where multiple policies apply. This extension lists policies that the certificate is recognized as supporting, together with optional qualifier information.
- **Policy mappings:** Used only in certificates for CAs issued by other CAs. Policy mappings allow an issuing CA to indicate that one or more of that issuer's policies can be considered equivalent to another policy used in the subject CA's domain.

Certificate Subject and Issuer Attributes These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required.

The extension fields in this area include:

- **Subject alternative name:** Contains one or more alternative names, using any of a variety of forms. This field is important for supporting certain applications, such as electronic mail, EDI, and IPSec, which may employ their own name forms.

- **Issuer alternative name:** Contains one or more alternative names, using any of a variety of forms.
- **Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

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

The extension fields in this area include:

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

14.5 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. This section describes the PKIX 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.

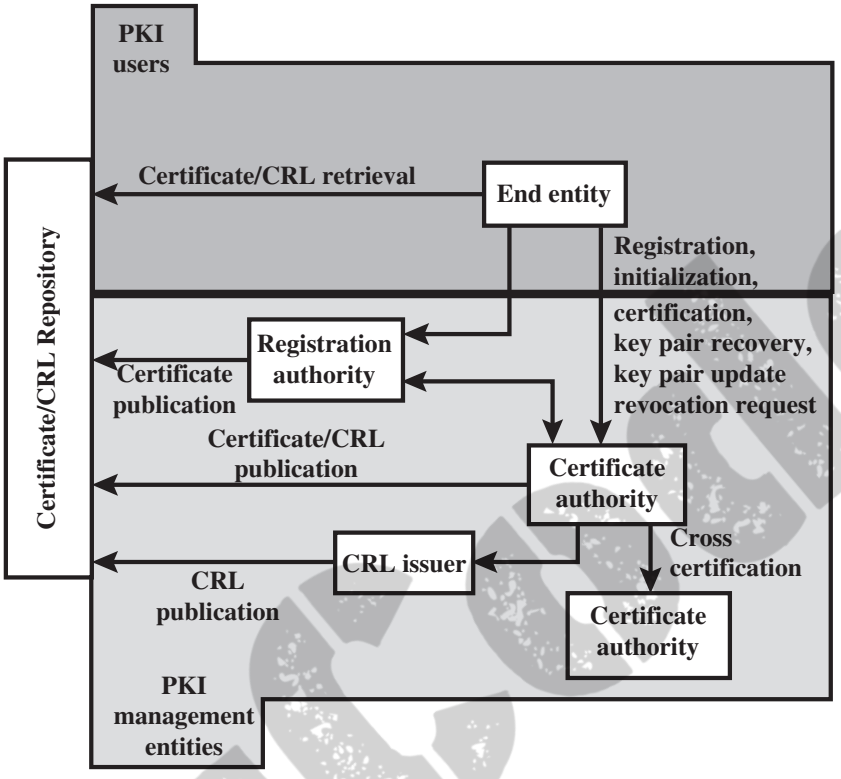

Figure 14.17 PKIX Architectural Model

- **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. For example, the client needs to be securely initialized with the public key and other assured information of the trusted CA(s), to be used in validating certificate paths.

- **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. Loss of access to the decryption key can result from forgotten passwords/PINs, corrupted disk drives, damage to hardware tokens, and so on. Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).
- **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. Although all of the PKIX functions are supported, the functions do not all map into specific protocol exchanges.

15.1 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. RFC 4949 (*Internet Security Glossary*) defines user authentication as shown on the following page.

For example, user Alice Toklas could have the user identifier ABTOKLAS. This information needs to be stored on any server or computer system that Alice wishes to use and could be known to system administrators and other users. A typical item of authentication information associated with this user

The process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:

- **Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
- **Verification step:** Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

ID is a password, which is kept secret (known only to Alice and to the system). If no one is able to obtain or guess Alice's password, then the combination of Alice's user ID and password enables administrators to set up Alice's access permissions and audit her activity. Because Alice's ID is not secret, system users can send her e-mail, but because her password is secret, no one can pretend to be Alice.

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication. As defined in Chapter 12, message authentication is a procedure that allows communicating parties to verify that the contents of a received message have not been altered and that the source is authentic. This chapter is concerned solely with user authentication.

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.

All of these methods, properly implemented and used, can provide secure user authentication. However, 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

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. This topic was examined in Chapter 14. There, the focus was key distribution. We return to this topic here to consider the wider implications of authentication.

Central to the problem of authenticated key exchange are two issues: confidentiality and 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. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

[GONG93] lists 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. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. 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.

It can be argued (e.g., [LAM92a]) that the timestamp approach should not be used for connection-oriented applications because of the inherent difficulties with this

technique. First, some sort of protocol is needed to maintain synchronization among the various processor clocks. This protocol must be both fault tolerant, to cope with network errors, and secure, to cope with hostile attacks. Second, the opportunity for a successful attack will arise if there is a temporary loss of synchronization resulting from a fault in the clock mechanism of one of the parties. Finally, because of the variable and unpredictable nature of network delays, distributed clocks cannot be expected to maintain precise synchronization. Therefore, any timestamp-based procedure must allow for a window of time sufficiently large to accommodate network delays yet sufficiently small to minimize the opportunity for attack.

On the other hand, the challenge-response approach is unsuitable for a connectionless type of application, because it requires the overhead of a handshake before any connectionless transmission, effectively negating the chief characteristic of a connectionless transaction. For such applications, reliance on some sort of secure time server and a consistent attempt by each party to keep its clocks in synchronization may be the best approach (e.g., [LAM92b]).

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that 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. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of **authentication**. Typically, the recipient wants some assurance that the message is from the alleged sender.

15.2 REMOTE USER-AUTHENTICATION USING SYMMETRIC ENCRYPTION

Mutual Authentication

As was discussed in Chapter 14, a two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution. This approach is quite common. As an example, we look at the Kerberos system in Section 15.3. The discussion in this subsection is relevant to an understanding of the Kerberos mechanisms.

Figure 14.3 illustrates a proposal initially put forth by Needham and Schroeder [NEED78] for secret key distribution using a KDC that, as was mentioned in Chapter 14, includes authentication features. The protocol can be summarized as follows.[1]

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: $E(K_s, f(N_2))$

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh message because of the use of the nonce N_2. Recall from our discussion in Chapter 14 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, 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. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. 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 [DENN81, DENN82] 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, K_a and K_b, are secure, and it consists of the following steps.

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

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

$$|\text{Clock} - T| < \Delta t_1 + \Delta t_2$$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at B.

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. [GONG92] points out a risk involved. 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.[2] 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. This latter alternative is not vulnerable to a suppress-replay attack, because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

In [KEHN92], an attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. Subsequently, an inconsistency in this latter protocol was noted and an improved strategy was presented in [NEUM93a].[3] The protocol is

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. $A \rightarrow B$: $E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, 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, N_b. 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 (ID_B) and that this is a timely message and not a replay (N_a), and it provides A with a session key (K_s) and the time limit on its use (T_b).
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 $E(K_s, N_b)$ 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. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

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

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b 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 illustrated in Figure 14.5 is impractical. This scheme 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.

With some refinement, the KDC strategy illustrated in Figure 14.3 is a candidate for encrypted electronic mail. Because we wish 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. $A \rightarrow KDC$: $ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A$: $E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B$: $E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

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.

15.3 KERBEROS

Kerberos[4] 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.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access. Rather than building in elaborate authentication protocols at each server, Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes described in this book, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

Two versions of Kerberos are in common use. Version 4 [MILL88, STEI88] implementations still exist. Version 5 [KOHL94] corrects some of the security deficiencies of version 4 and has been issued as a proposed Internet Standard (RFC 4120 and RFC 4121).[5]

We begin this section with a brief discussion of the motivation for the Kerberos approach. Then, because of the complexity of Kerberos, it is best to start with a description of the authentication protocol used in version 4. This enables us to see the essence of the Kerberos strategy without considering some of the details required to handle subtle security threats. Finally, we examine 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.

Today, neither of these scenarios is typical. 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.

In a small, closed environment in which all systems are owned and operated by a single organization, the first or perhaps the second strategy may suffice.[6] But in a more open environment in which network connections to other machines are supported, the third approach is needed to protect user information and resources housed at the server. Kerberos supports this third approach. Kerberos assumes a distributed client/server architecture and employs one or more Kerberos servers to provide an authentication service.

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

To support these requirements, the overall scheme of Kerberos is that of a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78], which was discussed in Section 15.2.

It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.[7]

Kerberos Version 4

Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service. Viewing the protocol as a whole, it is difficult to see the need for the many elements contained therein. Therefore, we adopt a strategy used by Bill Bryant of Project Athena [BRYA88] and build up to the full protocol by looking first at several hypothetical dialogues. Each successive dialogue adds additional complexity to counter security vulnerabilities revealed in the preceding dialogue.

After examining the protocol, we look at some other aspects of version 4.

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:

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

where

$$\begin{aligned}
\mathrm{C} &= \text{client} \\
\mathrm{AS} &= \text{authentication server} \\
\mathrm{V} &= \text{server} \\
ID_C &= \text{identifier of user on C} \\
ID_V &= \text{identifier of V} \\
P_C &= \text{password of user on C}
\end{aligned}$$

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 encrypted using the secret key shared by the AS and this server. 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.

Each of the ingredients of message (3) is significant. The ticket is encrypted to prevent alteration or forgery. The server's ID (ID_V) is included in the ticket so that the server can verify that it has decrypted the ticket properly. ID_C is included in the ticket to indicate that this ticket has been issued on behalf of C. Finally, AD_C serves to counter the following threat. An opponent could capture the ticket transmitted in message (2), then use the name ID_C and transmit a message of form (3) from another workstation. The server would receive a valid ticket that matches the user ID and grant access to the user on that other workstation. To prevent this attack, the AS includes in the ticket the network address from which the original request came. Now the ticket is valid only if it is transmitted from the same workstation that initially requested the ticket.

A More Secure Authentication Dialogue Although the foregoing scenario solves some of the problems of authentication in an open network environment, problems remain. Two in particular stand out. First, we would like to minimize the number of times that a user has to enter a password. Suppose each ticket can be used only once. If user C logs on to a workstation in the morning and wishes to check his or her mail at a mail server, C must supply a password to get a ticket for the mail server. If C wishes to check the mail several times during the day, each attempt requires reentering the password. We can improve matters by saying that tickets are reusable. For a single logon session, the workstation can store the mail server ticket after it is received and use it on behalf of the user for multiple accesses to the mail server.

However, under this scheme, it remains the case that a user would need a new ticket for every different service. If a user wished to access a print server, a mail server, a file server, and so on, the first instance of each access would require a new ticket and hence require the user to enter the password.

The second problem is that the earlier scenario involved a plaintext transmission of the password [message (1)]. An eavesdropper could capture the password and use any service accessible to the victim.

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 → AS: $ID_C \| ID_{tgs}$

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

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: $ID_C \| Ticket_v$

$$Ticket_{tgs} = \mathrm{E}(K_{tgs}, [ID_C \| AD_C \| ID_{tgs} \| TS_1 \| Lifetime_1])$$
$$Ticket_v = \mathrm{E}(K_v, [ID_C \| AD_C \| ID_v \| TS_2 \| Lifetime_2])$$

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket ($Ticket_{tgs}$) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

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.

Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext. The ticket itself consists of the ID and network address of the user, and the ID of the TGS. This corresponds to the first scenario. The idea is that the client can use this ticket to request multiple service-granting tickets. So the ticket-granting ticket is to be reusable. However, we do not wish an opponent to be able to capture the ticket and use it. Consider the following scenario: An opponent captures the login ticket and waits until the user has logged off his or her workstation. Then the opponent either gains access to that workstation or configures his workstation with the same network address as that of the victim. The opponent would be able to reuse the ticket

to spoof the TGS. To counter this, the ticket includes a timestamp, indicating the date and time at which the ticket was issued, and a lifetime, indicating the length of time for which the ticket is valid (e.g., eight hours). Thus, the client now has a reusable ticket and need not bother the user for a password for each new service request. Finally, note that the ticket-granting ticket is encrypted with a secret key known only to the AS and the TGS. This prevents alteration of the ticket. The ticket is reencrypted with a key based on the user's password. This assures that the ticket can be recovered only by the correct user, providing the authentication.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4.

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.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (K_v) known only to the TGS and the server, preventing alteration.

Finally, with a particular service-granting ticket, the client can gain access to the corresponding service with step 5.

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.

This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

The Version 4 Authentication Dialogue Although the foregoing scenario enhances security compared to the first attempt, two additional problems remain. 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. An opponent could eavesdrop on the network and capture a copy of the ticket-granting ticket and then wait for the legitimate user to log out. Then the opponent could forge the legitimate user's network address and send the message of step (3) to the TGS. This would give the opponent unlimited access to the resources and files available to the legitimate user.

Similarly, if an opponent captures a service-granting ticket and uses it before it expires, the opponent has access to the corresponding service.

Thus, we arrive at an additional requirement. A network service (the TGS or an application service) must be able to prove that the person using a ticket is the same person to whom that ticket was issued.

The second problem is that 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.

We examine these problems in turn and refer to Table 15.1, which shows the actual Kerberos protocol. Figure 15.1 provides a simplified overview.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. The threat is that an opponent will steal the ticket and use it before it expires. To get around this problem, let us have the AS provide both the client and the TGS with a secret piece of information in a secure manner. Then the client can prove its identity to the TGS by revealing the secret information—again in a secure manner. An efficient way of accomplishing this is to use an encryption key as the secure information; this is referred to as a session key in Kerberos.

Table 15.1a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message, encrypted with a key derived from the user's password (K_c), that contains the ticket. The encrypted message also contains a copy of the session key, $K_{c,tgs}$, where the subscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with K_c, only the user's client can read it. The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Table 15.1 Summary of Kerberos Version 4 Message Exchanges

(1)	**C → AS**	$ID_c \| ID_{tgs} \| TS_1$
(2)	**AS → C**	$\mathrm{E}(K_c, [K_{c,tgs} \| ID_{tgs} \| TS_2 \| \| Lifetime_2 \| Ticket_{tgs}])$
		$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$

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

(3)	**C → TGS**	$ID_v \| Ticket_{tgs} \| Authenticator_c$
(4)	**TGS → C**	$\mathrm{E}(K_{c,tgs}, [K_{c,v} \| ID_v \| TS_4 \| Ticket_v])$
		$Ticket_{tgs} = \mathrm{E}(\mathrm{K}_{tgs}, [\mathrm{K}_{c,tgs} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_{tgs} \| \mathrm{TS}_2 \| \mathrm{Lifetime}_2])$
		$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$
		$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,tgs}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_3])$

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

(5)	**C → V**	$Ticket_v \| Authenticator_c$
(6)	**V → C**	$\mathrm{E}(K_{c,v}, [TS_5 + 1])$ (for mutual authentication)
		$Ticket_v = \mathrm{E}(\mathrm{K}_v, [\mathrm{K}_{c,v} \| \mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{ID}_v \| \mathrm{TS}_4 \| \mathrm{Lifetime}_4])$
		$Authenticator_c = \mathrm{E}(\mathrm{K}_{c,v}, [\mathrm{ID}_C \| \mathrm{AD}_C \| \mathrm{TS}_5])$

(c) Client/Server Authentication Exchange to obtain service

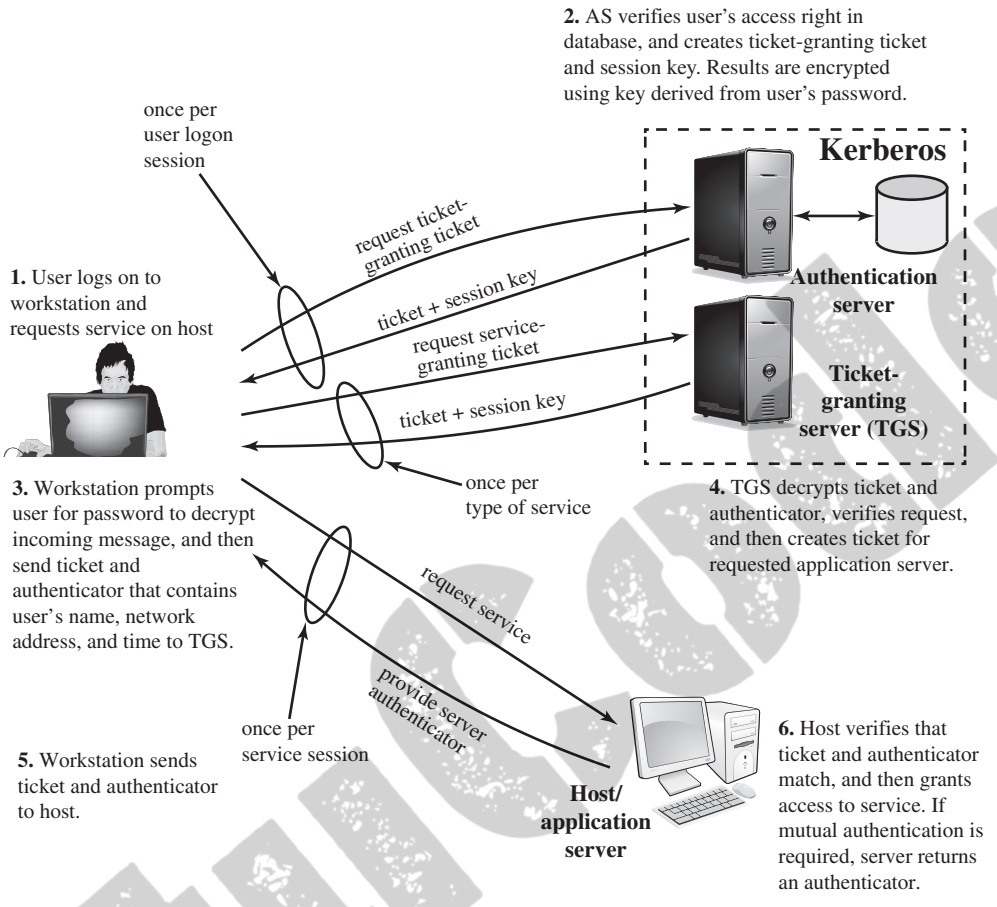

Figure 15.1 Overview of Kerberos

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time.

Armed with the ticket and the session key, C is ready to approach the TGS. As before, C sends the TGS a message that includes the ticket plus the ID of the requested service [message (3) in Table 15.1b]. In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. The TGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key $K_{c,tgs}$. In effect, the ticket says, "Anyone who uses $K_{c,tgs}$ must be C." The TGS uses the session key to decrypt the authenticator. The TGS can then check the name and address from the authenticator with that of the ticket and with the network address of the incoming message. If all match, then the TGS is assured that the sender of the ticket is indeed the ticket's real owner. In effect, the authenticator says, "At time TS_3, I hereby use $K_{c,tgs}$." Note that the ticket does not prove anyone's identity but is a way to distribute

keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered.

The reply from the TGS in message (4) follows the form of message (2). The message is encrypted with the session key shared by the TGS and C and includes a session key to be shared between C and the server V, the ID of V, and the timestamp of the ticket. The ticket itself includes the same session key.

C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator.

If mutual authentication is required, the server can reply as shown in message (6) of Table 15.1. The server returns the value of the timestamp from the authenticator, incremented by 1, and encrypted in the session key. C can decrypt this message to recover the incremented timestamp. Because the message was encrypted by the session key, C is assured that it could have been created only by V. The contents of the message assure C that this is not a replay of an old reply.

Finally, at the conclusion of this process, the client and server share a secret key. This key can be used to encrypt future messages between the two or to exchange a new random session key for that purpose.

Figure 15.2 illustrates the Kerberos exchanges among the parties. Table 15.2 summarizes the justification for each of the elements in the Kerberos protocol.

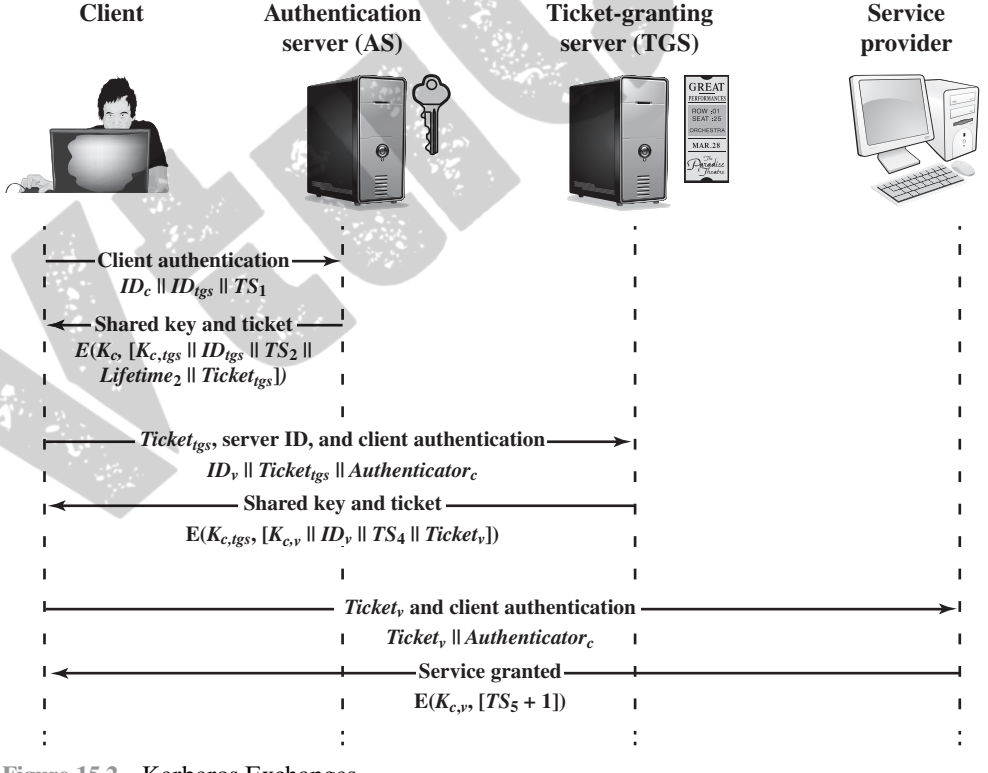

Figure 15.2 Kerberos Exchanges

Table 15.2 Rationale for the Elements of the Kerberos Version 4 Protocol

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

(*continued*)

Table 15.2 Continued

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**. The concept of **realm** can be explained as follows. A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos database resides on the Kerberos master computer system, which should be kept in a physically secure room. A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password. A related concept is that of a **Kerberos principal**, which is a service or user that is known to the Kerberos system. Each Kerberos

principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name.

Networks of clients and servers under different administrative organizations typically constitute different realms. That is, it generally is not practical or does not conform to administrative policy to have users and servers in one administrative domain registered with a Kerberos server elsewhere. However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

3. The Kerberos server in each interoperating realm shares a secret key with the server in the other realm. The two Kerberos servers are registered with each other.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

With these ground rules in place, we can describe the mechanism as follows (Figure 15.3): 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.

The details of the exchanges illustrated in Figure 15.3 are as follows (compare Table 15.1).

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

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

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

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

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

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

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

The ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request.

One problem presented by the foregoing approach is that it does not scale well to many realms. If there are N realms, then there must be $N(N - 1)/2$ secure key exchanges so that each Kerberos realm can interoperate with all other Kerberos realms.

Kerberos Version 5

Kerberos version 5 is specified in RFC 4120 and provides a number of improvements over version 4 [KOHL94]. To begin, we provide an overview of the changes from version 4 to version 5 and then look at the version 5 protocol.

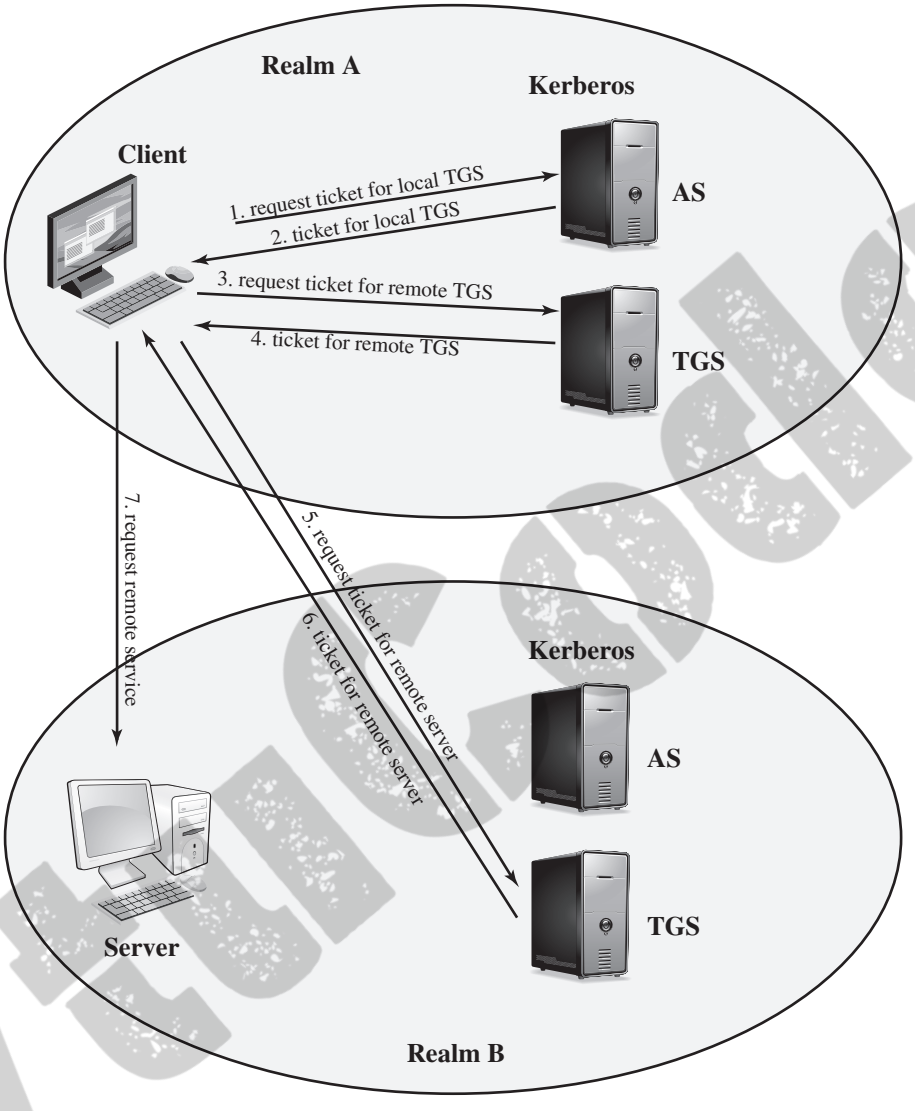

Figure 15.3 Request for Service in Another Realm

DIFFERENCES BETWEEN VERSIONS 4 AND 5 Version 5 is intended to address the limitations of version 4 in two areas: environmental shortcomings and technical deficiencies. Let us briefly summarize the improvements in each area.[8]

Kerberos version 4 was developed for use within the Project Athena environment and, accordingly, did not fully address the need to be of general purpose. This led to the following **environmental shortcomings**.

1. **Encryption system dependence:** Version 4 requires the use of DES. Export restriction on DES as well as doubts about the strength of DES were thus of

concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used. Encryption keys are tagged with a type and a length, allowing the same key to be used in different algorithms and allowing the specification of different variations on a given algorithm.

2. **Internet protocol dependence:** Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.
3. **Message byte ordering:** In version 4, the sender of a message employs a byte ordering of its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) and Basic Encoding Rules (BER), which provide an unambiguous byte ordering.
4. **Ticket lifetime:** Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 \times 5 = 1280$ minutes (a little over 21 hours). This may be inadequate for some applications (e.g., a long-running simulation that requires valid Kerberos credentials throughout execution). In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.
5. **Authentication forwarding:** Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 provides this capability.
6. **Interrealm authentication:** In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Apart from these environmental limitations, there are **technical deficiencies** in the version 4 protocol itself. Most of these deficiencies were documented in [BELL90], and version 5 attempts to address these. The deficiencies are the following.

1. **Double encryption:** Note in Table 15.1 [messages (2) and (4)] that 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.
2. **PCBC encryption:** Encryption in version 4 makes use of a nonstandard mode of DES known as **propagating cipher block chaining** (PCBC).[9] It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks [KOHL89]. PCBC was intended to provide an integrity check as part of the encryption operation. Version 5 provides explicit integrity mechanisms,

allowing the standard CBC mode to be used for encryption. In particular, a checksum or hash code is attached to the message prior to encryption using CBC.

3. **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. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection. A new access by the client would result in the use of a new subsession key.
4. **Password attacks:** Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password.[10] An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. This is the same type of password attack described in Chapter 21, with the same kinds of countermeasures being applicable. Version 5 does provide a mechanism known as preauthentication, which should make password attacks more difficult, but it does not prevent them.

THE VERSION 5 AUTHENTICATION DIALOGUE Table 15.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 15.1).

Table 15.3 Summary of Kerberos Version 5 Message Exchanges

(1)	**C → AS**	$Options \| ID_c \| Realm_c \| ID_{tgs} \| Times \| Nonce_1$
(2)	**AS → 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)	**C → TGS**	$Options \| ID_v \| Times \| Nonce_2 \| Ticket_{tgs} \| Authenticator_c$
(4)	**TGS → 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)	**C → V**	$Options \| Ticket_v \| Authenticator_c$
(6)	**V → 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

First, consider the **authentication service exchange**. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:

- **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

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the **ticket-granting service exchange** for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message (1). The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.

Finally, for the **client/server authentication exchange**, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:

- **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.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5, because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys. The subkey field, if present, overrides the subkey field, if present, in message (5).

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.

The optional sequence number field specifies the starting sequence number to be used by the client.

Ticket Flags The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4. Table 15.4 summarizes the flags that may be included in a ticket.

The INITIAL flag indicates that this ticket was issued by the AS, not by the TGS. When a client requests a service-granting ticket from the TGS, it presents a ticket-granting ticket obtained from the AS. In version 4, this was the only way to obtain a service-granting ticket. Version 5 provides the additional capability that the client can get a service-granting ticket directly from the AS. The utility of this is as follows: A server, such as a password-changing server, may wish to know that the client's password was recently tested.

The PRE-AUTHENT flag, if set, indicates that when the AS received the initial request [message (1)], it authenticated the client before issuing a ticket. The exact form of this preauthentication is left unspecified. As an example, the MIT implementation of version 5 has encrypted timestamp preauthentication, enabled by default. When a user wants to get a ticket, it has to send to the AS a preauthentication block containing a random confounder, a version number, and a timestamp all encrypted in the client's password-based key. The AS decrypts the block and will not send a ticket-granting ticket back unless the timestamp in the preauthentication block is within the allowable time skew (time interval to account for clock drift and network delays). Another possibility is the use of a smart card that generates continually changing passwords that are included in the preauthenticated messages. The passwords generated by the card can be

based on a user's password but be transformed by the card so that, in effect, arbitrary passwords are used. This prevents an attack based on easily guessed passwords. If a smart card or similar device was used, this is indicated by the HW-AUTHENT flag.

When a ticket has a long lifetime, there is the potential for it to be stolen and used by an opponent for a considerable period. If a short lifetime is used to lessen the threat, then overhead is involved in acquiring new tickets. In the case of a ticket-granting ticket, the client would either have to store the user's secret key, which is clearly risky, or repeatedly ask the user for a password. A compromise scheme is the use of renewable tickets. A ticket with the RENEWABLE flag set includes two expiration times: One for this specific ticket and one that is the latest permissible value for an expiration time. A client can have the ticket renewed by presenting it to the TGS with a requested new expiration time. If the new time is within the limit of the latest permissible value, the TGS can issue a new ticket with a new session time and a later specific expiration time. The advantage of this mechanism is that the TGS may refuse to renew a ticket reported as stolen.

A client may request that the AS provide a ticket-granting ticket with the MAY-POSTDATE flag set. The client can then use this ticket to request a ticket that is flagged as POSTDATED and INVALID from the TGS. Subsequently, the client may submit the postdated ticket for validation. This scheme can be useful for running a long batch job on a server that requires a ticket periodically. The client can obtain a number of tickets for this session at once, with spread out time values. All but the first ticket are initially invalid. When the execution reaches a point in time when a new ticket is required, the client can get the appropriate ticket validated. With this approach, the client does not have to repeatedly use its ticket-granting ticket to obtain a service-granting ticket.

In version 5, it is possible for a server to act as a proxy on behalf of a client, in effect adopting the credentials and privileges of the client to request a service from another server. If a client wishes to use this mechanism, it requests a ticket-granting ticket with the PROXIABLE flag set. When this ticket is presented to the TGS, the TGS is permitted to issue a service-granting ticket with a different network address; this latter ticket will have its PROXY flag set. An application receiving such a ticket may accept it or require additional authentication to provide an audit trail.[11]

The proxy concept is a limited case of the more powerful forwarding procedure. If a ticket is set with the FORWARDABLE flag, a TGS can issue to the requestor a ticket-granting ticket with a different network address and the FORWARDED flag set. This ticket then can be presented to a remote TGS. This capability allows a client to gain access to a server on another realm without requiring that each Kerberos maintain a secret key with Kerberos servers in every other realm. For example, realms could be structured hierarchically. Then a client could walk up the tree to a common node and then back down to reach a target realm. Each step of the walk would involve forwarding a ticket-granting ticket to the next TGS in the path.

15.4 REMOTE USER AUTHENTICATION USING ASYMMETRIC ENCRYPTION

Mutual Authentication

In Chapter 14, we presented one approach to the use of public-key encryption for the purpose of session-key distribution (Figure 14.9). This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in [DENN81]:

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

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret-key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

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.

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

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a (step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$. This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted using the KDC's private key to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key so that no other entity may

use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s, uses it to encrypt N_b, and returns it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

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: $E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 12.1b), authentication (Figure 12.1c), or both (Figure 12.1d). These approaches require that either the sender know the recipient's public key (confidentiality), the recipient know the sender's public key (authentication), or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

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. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 13.2:

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

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail 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 his, 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:

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

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, described in Chapter 14. Now we have

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

In addition to the message, A sends B the signature encrypted with A's private key and 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. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 19.