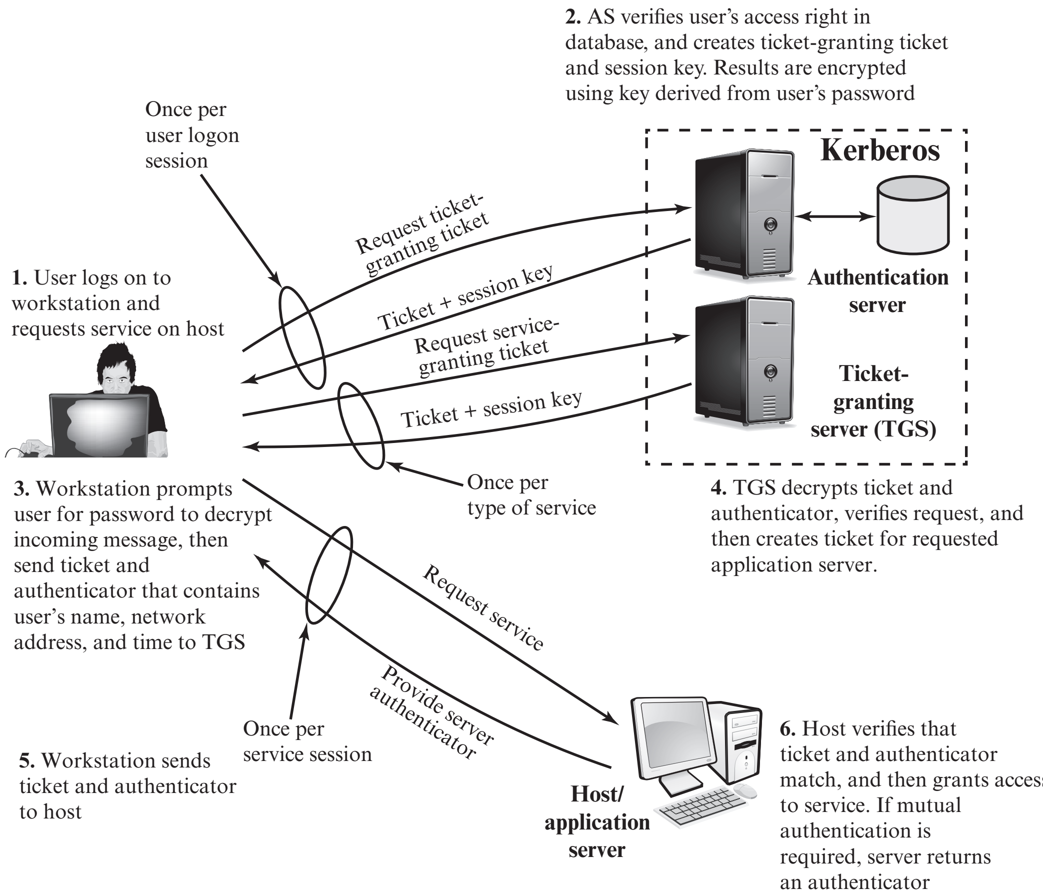

Figure 4.2 Overview of Kerberos

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 4.1b]. 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

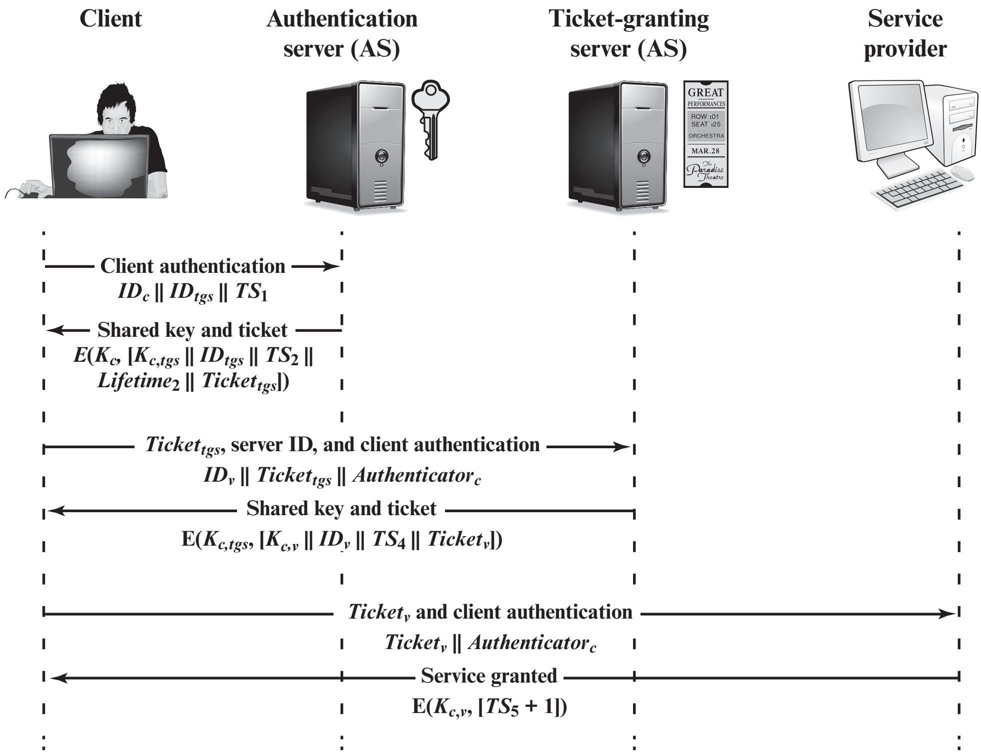

Figure 4.3 Kerberos Exchanges

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 4.1. 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 4.3 illustrates the Kerberos exchanges among the parties. Table 4.2 summarizes the justification for each of the elements in the Kerberos protocol.

Table 4.2 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

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 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.

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. We briefly summarize the improvements in each area. Kerberos version 4 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 technique 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 4.1 [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)**.[2] 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 subsequently may 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.[3] An opponent can capture this message and attempt

[2]This is described in Appendix F.

[3]Appendix F describes the mapping of passwords to encryption keys.

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 10, 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 4.3 summarizes the basic version 5 dialogue. This is best explained by comparison with version 4 (Table 4.1).

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.

Table 4.3 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	$\mathrm{E}_{K_{c,v}}[TS_2 \| Subkey \| Seq\#]$
	$\mathrm{Ticket}_v = \mathrm{E}(K_v, [\mathrm{Flags} \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times])$
	$Authenticator_c = \mathrm{E}(K_{c,v}, [ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#])$

(c) Client/Server Authentication Exchange to obtain service

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

4.4 KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

One of the major roles of public-key encryption is to address the problem of key distribution. There are actually two distinct aspects to the use of public-key encryption in this regard.

- The distribution of public keys.
- The use of public-key encryption to distribute secret keys.

We examine each of these areas in turn.

Public-Key Certificates

On the face of it, the point of public-key encryption is that the public key is public. Thus, if there is some broadly accepted public-key algorithm, such as RSA, any participant can send his or her public key to any other participant or broadcast the key to the community at large. Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. Until such time as user A discovers the forgery and alerts other participants, the forger is able to read all encrypted messages intended for A and can use the forged keys for authentication.

The solution to this problem is the **public-key certificate**. In essence, a certificate consists of a public key plus a user ID of the key owner, with the whole block signed by a trusted third party. Typically, the third party is a certificate authority (CA) that is trusted by the user community, such as a government agency or a financial institution. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. Figure 4.4 illustrates the process.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard. X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), and S/MIME—all of which are discussed in subsequent chapters. X.509 is examined in detail in the next section.

Public-Key Distribution of Secret Keys

With conventional encryption, a fundamental requirement for two parties to communicate securely is that they share a secret key. Suppose Bob wants to create a messaging application that will enable him to exchange e-mail securely with anyone

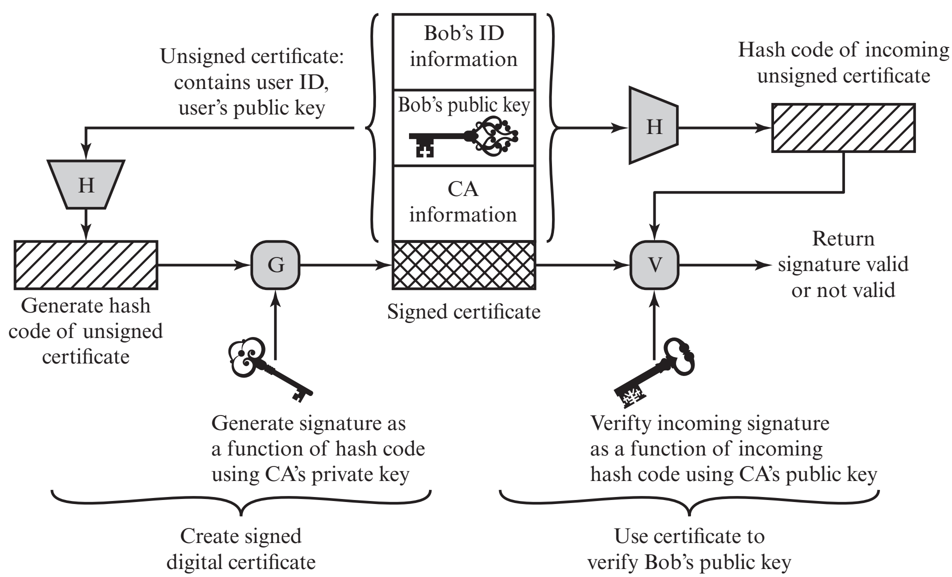

Figure 4.4 Public-Key Certificate Use

who has access to the Internet or to some other network that the two of them share. Suppose Bob wants to do this using conventional encryption. With conventional encryption, Bob and his correspondent, say, Alice, must come up with a way to share a unique secret key that no one else knows. How are they going to do that? If Alice is in the next room from Bob, Bob could generate a key and write it down on a piece of paper or store it on a diskette and hand it to Alice. But if Alice is on the other side of the continent or the world, what can Bob do? He could encrypt this key using conventional encryption and e-mail it to Alice, but this means that Bob and Alice must share a secret key to encrypt this new secret key. Furthermore, Bob and everyone else who uses this new e-mail package faces the same problem with every potential correspondent: Each pair of correspondents must share a unique secret key.

One approach is the use of Diffie–Hellman key exchange. This approach is indeed widely used. However, it suffers the drawback that, in its simplest form, Diffie–Hellman provides no authentication of the two communicating partners.

A powerful alternative is the use of public-key certificates. When Bob wishes to communicate with Alice, Bob can do the following:

1. Prepare a message.
2. Encrypt that message using conventional encryption with a one-time conventional session key.
3. Encrypt the session key using public-key encryption with Alice's public key.
4. Attach the encrypted session key to the message and send it to Alice.

Only Alice is capable of decrypting the session key and therefore of recovering the original message. If Bob obtained Alice's public key by means of Alice's public-key certificate, then Bob is assured that it is a valid key.

4.5 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. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. In addition, X.509 defines alternative authentication protocols based on the use of public-key certificates.

X.509 is 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 8), IP Security (Chapter 9), and SSL/TLS (Chapter 6).

X.509 was initially issued in 1988. The standard was subsequently revised in 1993 to address some of the security concerns documented in [IANS90] and [MITC90]. The standard is currently at version 7, issued in 2012.