MODULE 3

11.1 APPLICATIONS OF CRYPTOGRAPHIC HASH FUNCTIONS

Perhaps the most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. To better understand some of the requirements and security implications for cryptographic hash functions, it is useful to look at the range of applications in which it is employed.

Message Authentication

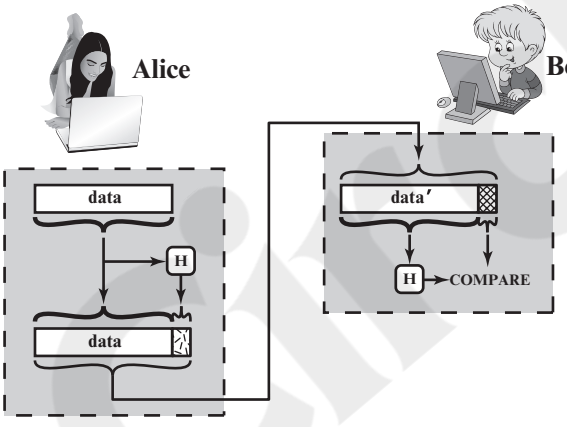
Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent (i.e., there is no modification, insertion, deletion, or replay). In many cases, there is a requirement that the authentication mechanism assures that purported identity of the sender is valid. When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.[1]

The essence of the use of a hash function for message integrity is as follows. The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.

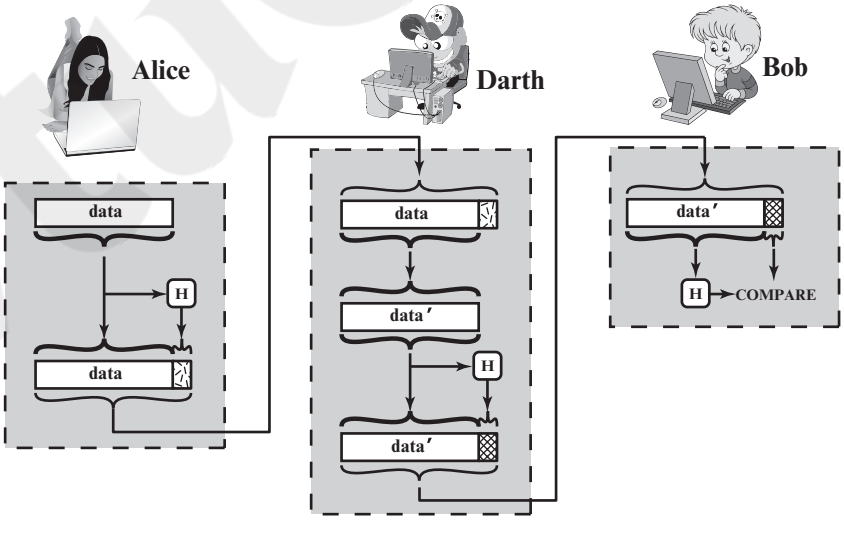
[1]The topic of this section is invariably referred to as message authentication. However, the concepts and techniques apply equally to data at rest. For example, authentication techniques can be applied to a file in storage to assure that the file is not tampered with.

If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered (Figure 11.2a).

The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver. This type of attack is shown in Figure 11.2b. In this example, Alice transmits a data block and attaches a hash value. Darth intercepts the message, alters or replaces the data block, and calculates and attaches a new hash value. Bob receives the altered data with the new hash value and does not detect the change. To prevent this attack, the hash value generated by Alice must be protected.



(a) Use of hash function to check data integrity



(b) Man-in-the-middle attack

Figure 11.2 Attack Against Hash Function

Figure 11.3 illustrates a variety of ways in which a hash code can be used to provide message authentication, as follows.

a. The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.

b. Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.

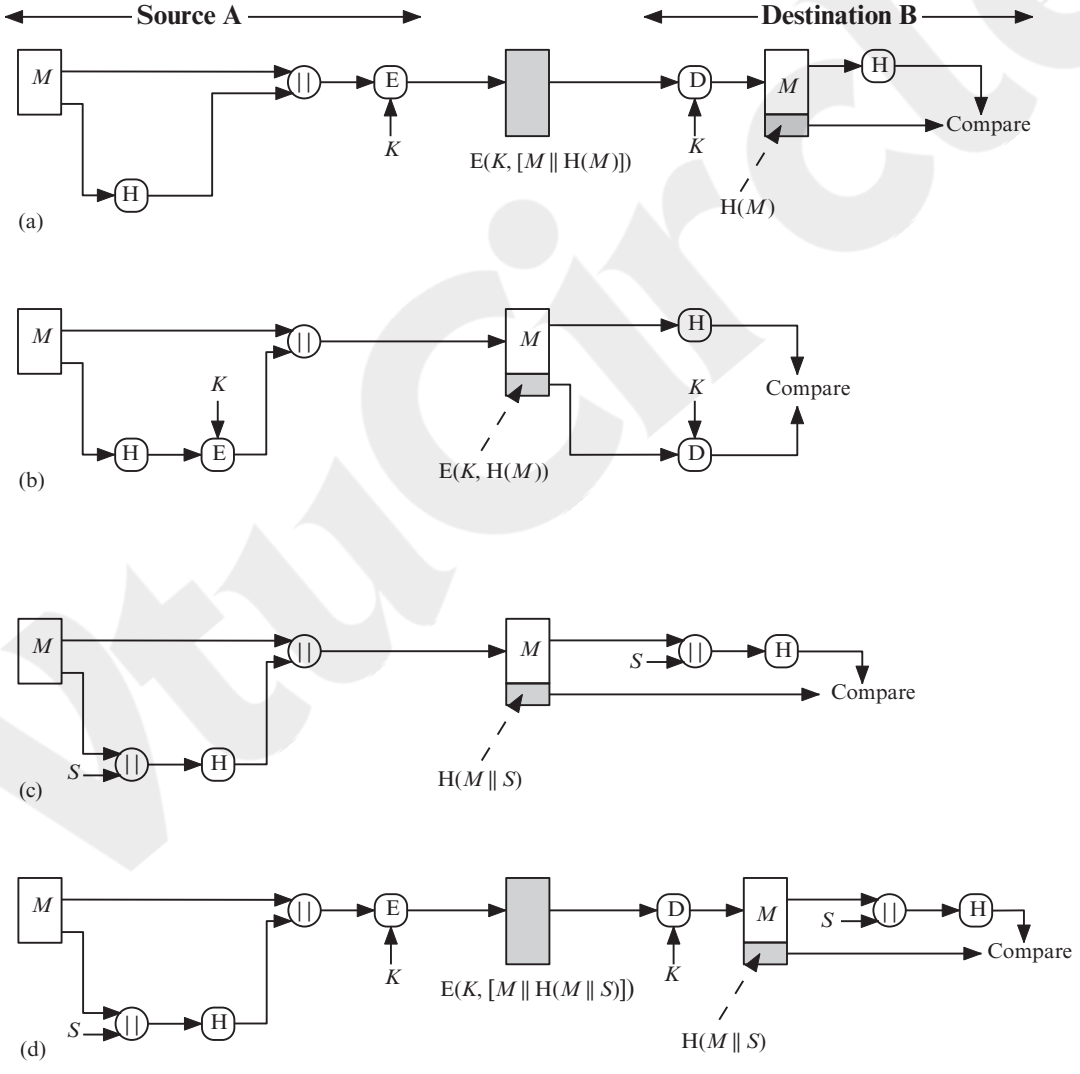


Figure 11.3 Simplified Examples of the Use of a Hash Function for Message Authentication

c. It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value S. A computes the hash value over the concatenation of M and S and appends the resulting hash value to M. Because B possesses S, it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.

d. Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.

When confidentiality is not required, method (b) has an advantage over methods (a) and (d), which encrypts the entire message, in that less computation is required. Nevertheless, there has been growing interest in techniques that avoid encryption (Figure 11.3c). Several reasons for this interest are pointed out in [TSUD92].

- Encryption software is relatively slow. Even though the amount of data to be encrypted per message is small, there may be a steady stream of messages into and out of a system.
- Encryption hardware costs are not negligible. Low-cost chip implementations of DES are available, but the cost adds up if all nodes in a network must have this capability.
- Encryption hardware is optimized toward large data sizes. For small blocks of data, a high proportion of the time is spent in initialization/invocation overhead.
- Encryption algorithms may be covered by patents, and there is a cost associated with licensing their use.

More commonly, message authentication is achieved using a **message authentication code (MAC)**, also known as a **keyed hash function**. Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties. A MAC function takes as input a secret key and a data block and produces a hash value, referred to as the MAC, which is associated with the protected message. If the integrity of the message needs to be checked, the MAC function can be applied to the message and the result compared with the associated MAC value. An attacker who alters the message will be unable to alter the associated MAC value without knowledge of the secret key. Note that the verifying party also knows who the sending party is because no one else knows the secret key.

Note that the combination of hashing and encryption results in an overall function that is, in fact, a MAC (Figure 11.3b). That is, $E(K, H(M))$ is a function of a variable-length message M and a secret key K, and it produces a fixed-size output that is secure against an opponent who does not know the secret key. In practice, specific MAC algorithms are designed that are generally more efficient than an encryption algorithm.

We discuss MACs in Chapter 12.

Digital Signatures

Another important application, which is similar to the message authentication application, is the **digital signature**. The operation of the digital signature is similar to that of the MAC. In the case of the digital signature, the hash value of a message

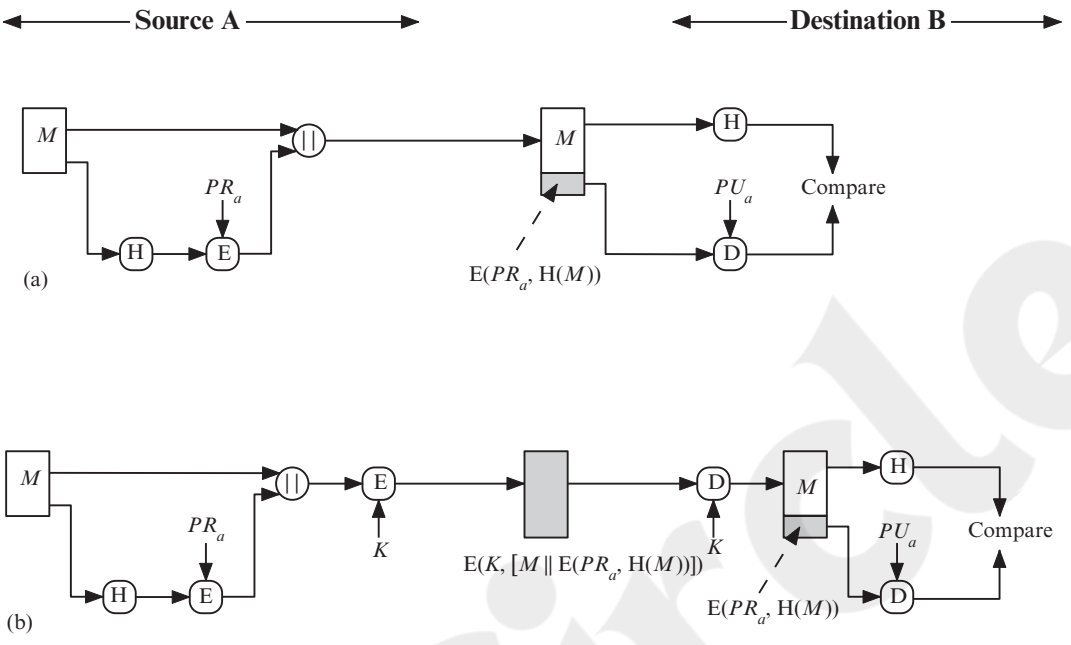


Figure 11.4 Simplified Examples of Digital Signatures

is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message would need to know the user's private key. As we shall see in Chapter 14, the implications of digital signatures go beyond just message authentication.

Figure 11.4 illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.

a. The hash code is encrypted, using public-key encryption with the sender's private key. As with Figure 11.3b, this provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.

b. If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key. This is a common technique.

Other Applications

Hash functions are commonly used to create a **one-way password file**. Chapter 21 explains a scheme in which a hash of a password is stored by an operating system rather than the password itself. Thus, the actual password is not retrievable by a hacker who gains access to the password file. In simple terms, when a user enters a password, the hash of that password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.

Hash functions can be used for **intrusion detection** and **virus detection**. Store H(F) for each file on a system and secure the hash values (e.g., on a CD-R that is

kept secure). One can later determine if a file has been modified by recomputing H(F). An intruder would need to change F without changing H(F).

A cryptographic hash function can be used to construct a **pseudorandom function (PRF)** or a **pseudorandom number generator (PRNG)**. A common application for a hash-based PRF is for the generation of symmetric keys. We discuss this application in Chapter 12.

11.2 TWO SIMPLE HASH FUNCTIONS

To get some feel for the security considerations involved in cryptographic hash functions, we present two simple, insecure hash functions in this section. All hash functions operate using the following general principles. The input (message, file, etc.) is viewed as a sequence of n -bit blocks. The input is processed one block at a time in an iterative fashion to produce an n-bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \cdots \oplus b_{im}$$

where

C_i = ith bit of the hash code, $1 \leq i \leq$ n
m = number of n-bit blocks in the input
b_{ij} = ith bit in jth block
$\oplus$ = XOR operation

This operation produces a simple parity bit for each bit position and is known as a longitudinal redundancy check. It is reasonably effective for random data as a data integrity check. Each n-bit hash value is equally likely. Thus, the probability that a data error will result in an unchanged hash value is 2^{-n}. With more predictably formatted data, the function is less effective. For example, in most normal text files, the high-order bit of each octet is always zero. So if a 128-bit hash value is used, instead of an effectiveness of 2^{-128}, the hash function on this type of data has an effectiveness of 2^{-112}.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.

1. Initially set the n-bit hash value to zero.
2. Process each successive n-bit block of data as follows:
   a. Rotate the current hash value to the left by one bit.
   b. XOR the block into the hash value.

This has the effect of "randomizing" the input more completely and overcoming any regularities that appear in the input. Figure 11.5 illustrates these two types of hash functions for 16-bit hash values.

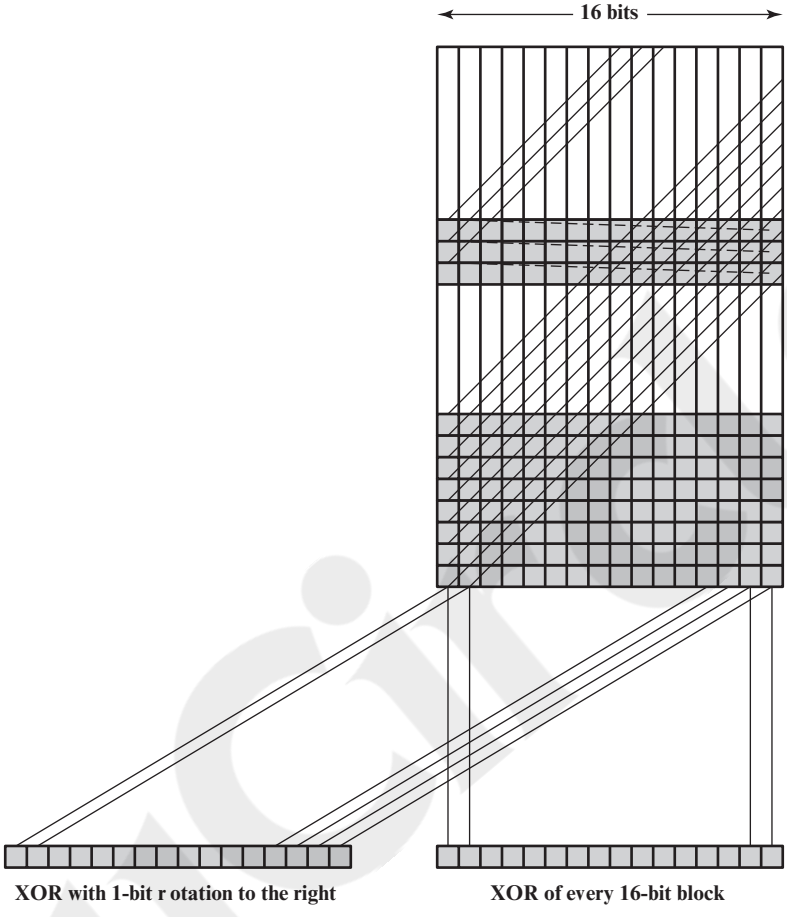


Figure 11.5 Two Simple Hash Functions

Although the second procedure provides a good measure of data integrity, it is virtually useless for data security when an encrypted hash code is used with a plaintext message, as in Figures 11.3b and 11.4a. Given a message, it is an easy matter to produce a new message that yields that hash code: Simply prepare the desired alternate message and then append an n-bit block that forces the new message plus block to yield the desired hash code.

Although a simple XOR or rotated XOR (RXOR) is insufficient if only the hash code is encrypted, you may still feel that such a simple function could be useful when the message together with the hash code is encrypted (Figure 11.3a). But you must be careful. A technique originally proposed by the National Bureau of Standards used the simple XOR applied to 64-bit blocks of the message and then an encryption of the entire message that used the cipher block chaining (CBC) mode. We can define the scheme as follows: Given a message M consisting of a sequence of 64-bit blocks $X_1, X_2, \ldots, X_N$, define the hash code

$h = \mathrm{H}(M)$ as the block-by-block XOR of all blocks and append the hash code as the final block:

$$h = X_{N+1} = X_1 \oplus X_2 \oplus \ \ldots \ \oplus X_N$$

Next, encrypt the entire message plus hash code using CBC mode to produce the encrypted message $Y_1, Y_2, \ldots, Y_{N+1}$. [JUEN85] points out several ways in which the ciphertext of this message can be manipulated in such a way that it is not detectable by the hash code. For example, by the definition of CBC (Figure 6.4), we have

$$\begin{aligned} X_1 &= IV \oplus \mathrm{D}(K, Y_1) \\ X_i &= Y_{i-1} \oplus \mathrm{D}(K, Y_i) \\ X_{N+1} &= Y_N \oplus \mathrm{D}(K, Y_{N+1}) \end{aligned}$$

But X_{N+1} is the hash code:

$$\begin{aligned} X_{N+1} &= X_1 \oplus X_2 \oplus \ \ldots \ \oplus X_N \\ &= [IV \oplus \mathrm{D}(K, Y_1)] \oplus [Y_1 \oplus \mathrm{D}(K, Y_2)] \oplus \ \ldots \ \oplus [Y_{N-1} \oplus \mathrm{D}(K, Y_N)] \end{aligned}$$

Because the terms in the preceding equation can be XORed in any order, it follows that the hash code would not change if the ciphertext blocks were permuted.

14.1 SYMMETRIC KEY DISTRIBUTION USING SYMMETRIC ENCRYPTION

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the *key distribution technique*, a term that refers to the means of delivering a key to two parties who wish to exchange data without allowing others to see the key. For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for **end-to-end encryption** over a network, manual delivery is awkward. In a distributed system, any given host or terminal may need to engage in exchanges with many other

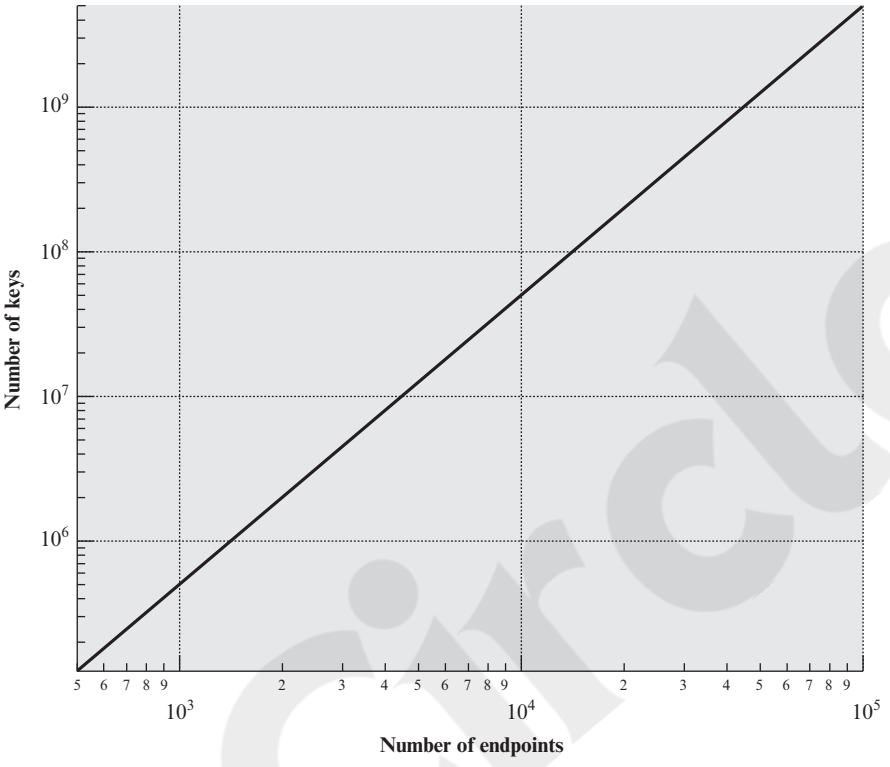


Figure 14.1 Number of Keys Required to Support Arbitrary Connections between Endpoints

hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system.

The scale of the problem depends on the number of communicating pairs that must be supported. If end-to-end encryption is done at a network or IP level, then a key is needed for each pair of hosts on the network that wish to communicate. Thus, if there are N hosts, the number of required keys is $[N(N - 1)]/2$. If encryption is done at the application level, then a key is needed for every pair of users or processes that require communication. Thus, a network may have hundreds of hosts but thousands of users and processes. Figure 14.1 illustrates the magnitude of the key distribution task for end-to-end encryption.[1] A network using node-level encryption with 1000 nodes would conceivably need to distribute as many as half a million keys. If that same network supported 10,000 applications, then as many as 50 million keys may be required for application-level encryption.

Returning to our list, option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys still must be made.

[1]Note that this figure uses a log-log scale, so that a linear graph indicates exponential growth. A basic review of log scales is in the math refresher document at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html.

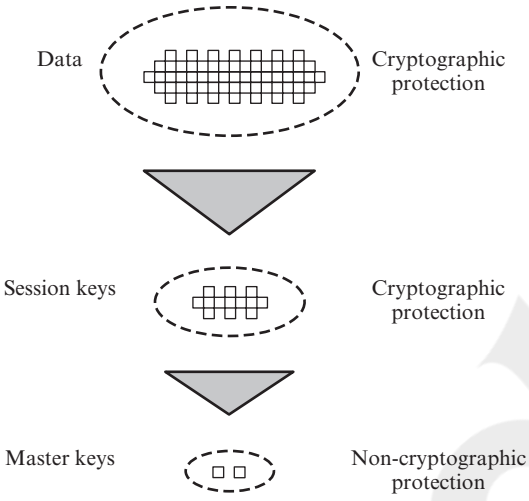


Figure 14.2 The Use of a Key Hierarchy

For end-to-end encryption, some variation on option 4 has been widely adopted. In this scheme, a key distribution center is responsible for distributing keys to pairs of users (hosts, processes, applications) as needed. Each user must share a unique key with the key distribution center for purposes of key distribution.

The use of a key distribution center is based on the use of a hierarchy of keys. At a minimum, two levels of keys are used (Figure 14.2). Communication between end systems is encrypted using a temporary key, often referred to as a **session key**. Typically, the session key is used for the duration of a logical connection, such as a frame relay connection or transport connection, and then discarded. Each session key is obtained from the key distribution center over the same networking facilities used for end-user communication. Accordingly, session keys are transmitted in encrypted form, using a **master key** that is shared by the key distribution center and an end system or user.

For each end system or user, there is a unique master key that it shares with the key distribution center. Of course, these master keys must be securely distributed in some fashion. However, the scale of the problem is vastly reduced. If there are N entities that wish to communicate in pairs, then, as was mentioned, as many as $[N(N - 1)]/2$ session keys are needed at any one time. However, only N master keys are required, one for each entity. Thus, master keys can be distributed in some non-cryptographic way, such as physical delivery.

A Key Distribution Scenario

The key distribution concept can be deployed in a number of ways. A typical scenario is illustrated in Figure 14.3, which is based on a figure in [POPE79]. The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection.

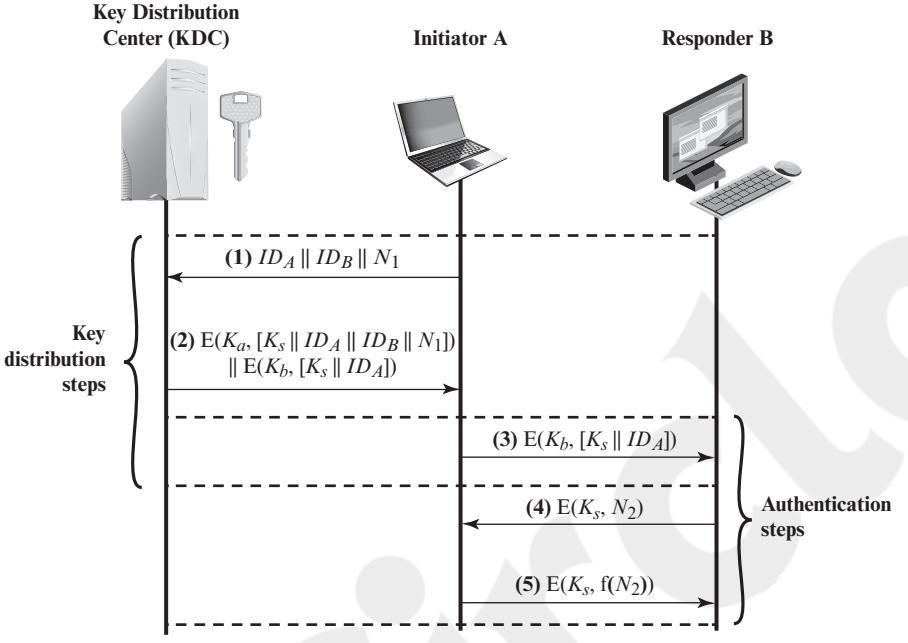


Figure 14.3 Key Distribution Scenario

A has a master key, K_a, known only to itself and the KDC; similarly, B shares the master key K_b with the KDC. The following steps occur.

1. A issues a request to the KDC for a session key to protect a logical connection to B. The message includes the identity of A and B and a unique identifier, N_1, for this transaction, which we refer to as a **nonce**. The nonce may be a timestamp, a counter, or a random number; the minimum requirement is that it differs with each request. Also, to prevent masquerade, it should be difficult for an opponent to guess the nonce. Thus, a random number is a good choice for a nonce.
2. The KDC responds with a message encrypted using K_a. Thus, A is the only one who can successfully read the message, and A knows that it originated at the KDC. The message includes two items intended for A:
   - The one-time session key, K_s, to be used for the session
   - The original request message, including the nonce, to enable A to match this response with the appropriate request

   Thus, A can verify that its original request was not altered before reception by the KDC and, because of the nonce, that this is not a replay of some previous request.

   In addition, the message includes two items intended for B:
   - The one-time session key, K_s, to be used for the session
   - An identifier of A (e.g., its network address), ID_A

   These last two items are encrypted with K_b (the master key that the KDC shares with B). They are to be sent to B to establish the connection and prove A's identity.

3. A stores the session key for use in the upcoming session and forwards to B the information that originated at the KDC for B, namely, $E(K_b,[K_s \| ID_A])$. Because this information is encrypted with K_b, it is protected from eavesdropping. B now knows the session key (K_s), knows that the other party is A (from ID_A), and knows that the information originated at the KDC (because it is encrypted using K_b).

At this point, a session key has been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

4. Using the newly minted session key for encryption, B sends a nonce, N_2, to A.
5. Also, using K_s, A responds with f(N_2), where f is a function that performs some transformation on N_2 (e.g., adding one).

These steps assure B that the original message it received (step 3) was not a replay.

Note that the actual key distribution involves only steps 1 through 3, but that steps 4 and 5, as well as step 3, perform an authentication function.

Hierarchical Key Control

It is not necessary to limit the key distribution function to a single KDC. Indeed, for very large networks, it may not be practical to do so. As an alternative, a hierarchy of KDCs can be established. For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC. In this case, any one of the three KDCs involved can actually select the key. The hierarchical concept can be extended to three or even more layers, depending on the size of the user population and the geographic scope of the internetwork.

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

Session Key Lifetime

The more frequently session keys are exchanged, the more secure they are, because the opponent has less ciphertext to work with for any given session key. On the other hand, the distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key.

For connection-oriented protocols, one obvious choice is to use the same session key for the length of time that the connection is open, using a new session key for each new session. If a logical connection has a very long lifetime, then it would be prudent to change the session key periodically, perhaps every time the PDU (protocol data unit) sequence number cycles.

For a connectionless protocol, such as a transaction-oriented protocol, there is no explicit connection initiation or termination. Thus, it is not obvious how often one needs to change the session key. The most secure approach is to use a new

session key for each exchange. However, this negates one of the principal benefits of connectionless protocols, which is minimum overhead and delay for each transaction. A better strategy is to use a given session key for a certain fixed period only or for a certain number of transactions.

A Transparent Key Control Scheme

The approach suggested in Figure 14.3 has many variations, one of which is described in this subsection. The scheme (Figure 14.4) is useful for providing end-to-end encryption at a network or transport level in a way that is transparent to the end users. The approach assumes that communication makes use of a connection-oriented end-to-end protocol, such as TCP. The noteworthy element of this approach is a session security module (SSM), which may consist of functionality

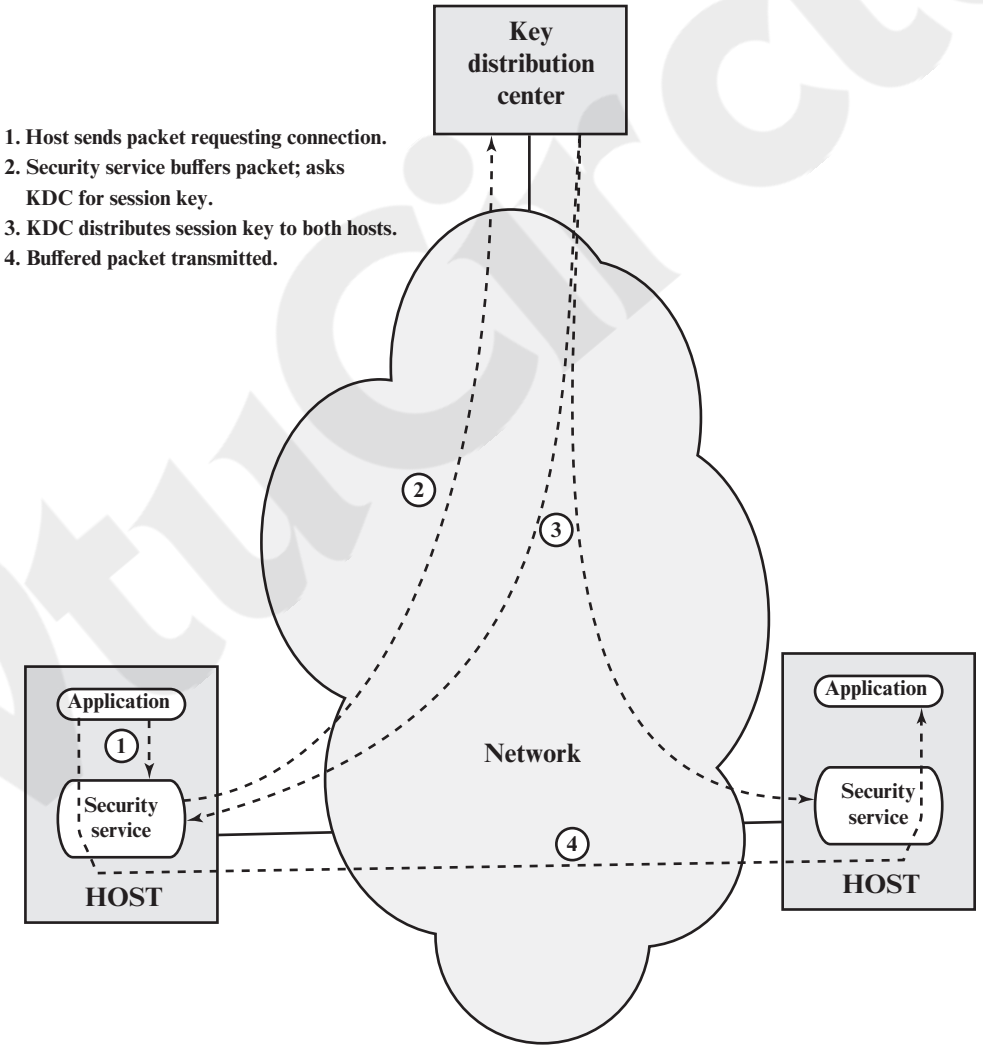


Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

at one protocol layer, that performs end-to-end encryption and obtains session keys on behalf of its host or terminal.

The steps involved in establishing a connection are shown in Figure 14.4. When one host wishes to set up a connection to another host, it transmits a connection-request packet (step 1). The SSM saves that packet and applies to the KDC for permission to establish the connection (step 2). The communication between the SSM and the KDC is encrypted using a master key shared only by this SSM and the KDC. If the KDC approves the connection request, it generates the session key and delivers it to the two appropriate SSMs, using a unique permanent key for each SSM (step 3). The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems (step 4). All user data exchanged between the two end systems are encrypted by their respective SSMs using the one-time session key.

The automated key distribution approach provides the flexibility and dynamic characteristics needed to allow a number of terminal users to access a number of hosts and for the hosts to exchange data with each other.

Decentralized Key Control

The use of a key distribution center imposes the requirement that the KDC be trusted and be protected from subversion. This requirement can be avoided if key distribution is fully decentralized. Although full decentralization is not practical for larger networks using symmetric encryption only, it may be useful within a local context.

A decentralized approach requires that each end system be able to communicate in a secure manner with all potential partner end systems for purposes of session key distribution. Thus, there may need to be as many as $[n(n - 1)]/2$ master keys for a configuration with n end systems.

A session key may be established with the following sequence of steps (Figure 14.5).

1. A issues a request to B for a session key and includes a nonce, N_1.
2. B responds with a message that is encrypted using the shared master key. The response includes the session key selected by B, an identifier of B, the value $f(N_1)$, and another nonce, N_2.
3. Using the new session key, A returns $f(N_2)$ to B.

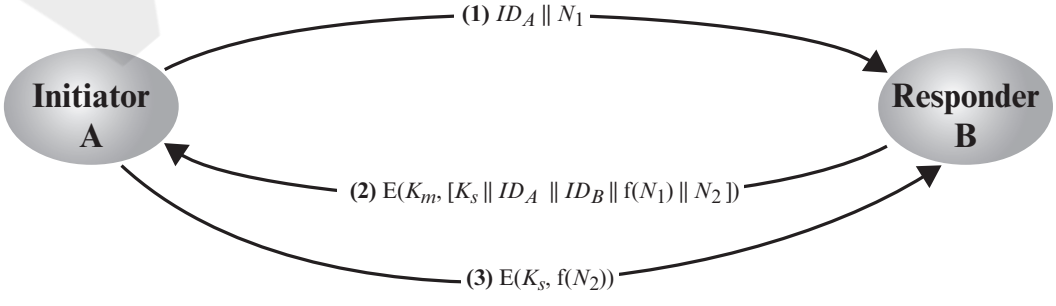


Figure 14.5 Decentralized Key Distribution

Thus, although each node must maintain at most $(n - 1)$ master keys, as many session keys as required may be generated and used. Because the messages transferred using the master key are short, cryptanalysis is difficult. As before, session keys are used for only a limited time to protect them.

Controlling Key Usage

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

- Data-encrypting key, for general communication across a network
- PIN-encrypting key, for personal identification numbers (PINs) used in electronic funds transfer and point-of-sale applications
- File-encrypting key, for encrypting files stored in publicly accessible locations

To illustrate the value of separating keys by type, consider the risk that a master key is imported as a data-encrypting key into a device. Normally, the master key is physically secured within the cryptographic hardware of the key distribution center and of the end systems. Session keys encrypted with this master key are available to application programs, as are the data encrypted with such session keys. However, if a master key is treated as a session key, it may be possible for an unauthorized application to obtain plaintext of session keys encrypted with that master key.

Thus, it may be desirable to institute controls in systems that limit the ways in which keys are used, based on characteristics associated with those keys. One simple plan is to associate a tag with each key ([JONE82]; see also [DAVI89]). The proposed technique is for use with DES and makes use of the extra 8 bits in each 64-bit DES key. That is, the eight non-key bits ordinarily reserved for parity checking form the key tag. The bits have the following interpretation:

- One bit indicates whether the key is a session key or a master key
- One bit indicates whether the key can be used for encryption
- One bit indicates whether the key can be used for decryption
- The remaining bits are spares for future use.

Because the tag is embedded in the key, it is encrypted along with the key when that key is distributed, thus providing protection. The drawbacks of this scheme are

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

A more flexible scheme, referred to as the control vector, is described in [MATY91a and b]. In this scheme, each session key has an associated control vector consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.

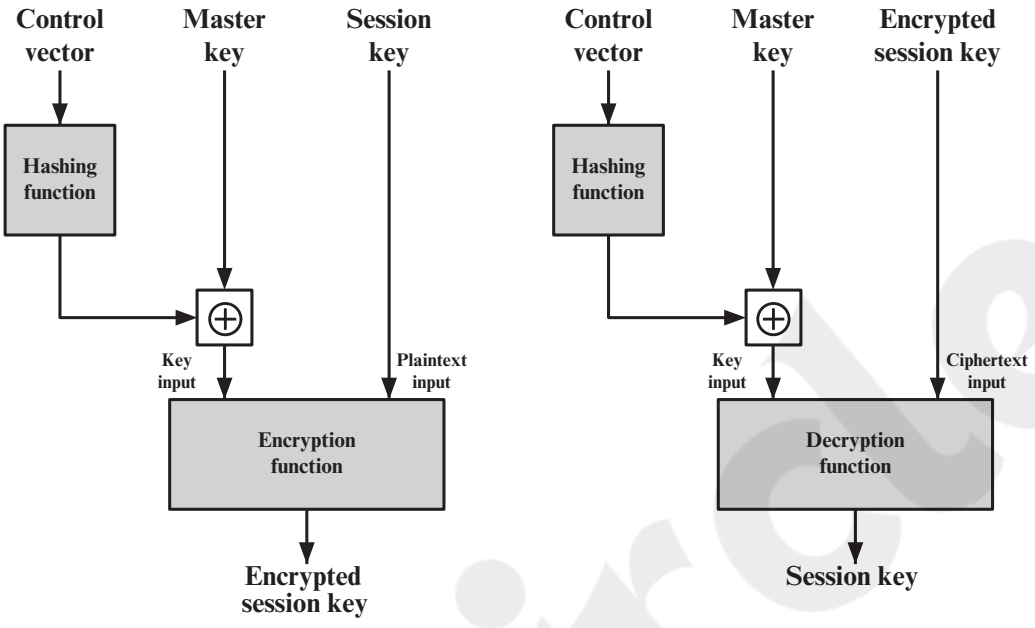


Figure 14.6 Control Vector Encryption and Decryption

The control vector is cryptographically coupled with the key at the time of key generation at the KDC. The coupling and decoupling processes are illustrated in Figure 14.6. As a first step, the control vector is passed through a hash function that produces a value whose length is equal to the encryption key length. Hash functions are discussed in detail in Chapter 11. In essence, a hash function maps values from a larger range into a smaller range with a reasonably uniform spread. Thus, for example, if numbers in the range 1 to 100 are hashed into numbers in the range 1 to 10, approximately 10% of the source values should map into each of the target values.

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

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

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

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

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

Use of the control vector has two advantages over use of an 8-bit tag. First, there is no restriction on length of the control vector, which enables arbitrarily complex controls to be imposed on key use. Second, the control vector is available in clear form at all stages of operation. Thus, control of key use can be exercised in multiple locations.

14.2 SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

Because of the inefficiency of public-key cryptosystems, they are almost never used for the direct encryption of sizable blocks of data, but are limited to relatively small blocks. One of the most important uses of a public-key cryptosystem is to encrypt secret keys for distribution. We see many specific examples of this in Part Five. Here, we discuss general principles and typical approaches.

Simple Secret Key Distribution

An extremely simple scheme was put forward by Merkle [MERK79], as illustrated in Figure 14.7. If A wishes to communicate with B, the following procedure is employed:

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message to B consisting of PU_a and an identifier of A, ID_A.
2. B generates a secret key, K_s, and transmits it to A, which is encrypted with A's public key.
3. A computes $D(PR_a, E(PU_a, K_s))$ to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of K_s.
4. A discards PU_a and PR_a and B discards PU_a.

A and B can now securely communicate using conventional encryption and the session key K_s. At the completion of the exchange, both A and B discard K_s. Despite its simplicity, this is an attractive protocol. No keys exist before the start of the communication and none exist after the completion of communication. Thus, the risk of compromise of the keys is minimal. At the same time, the communication is secure from eavesdropping.

The protocol depicted in Figure 14.7 is insecure against an adversary who can intercept messages and then either relay the intercepted message or substitute another message (see Figure 1.3c). Such an attack is known as a **man-in-the-middle attack** [RIVE84]. We saw this type of attack in Chapter 10 (Figure 10.2). In the present

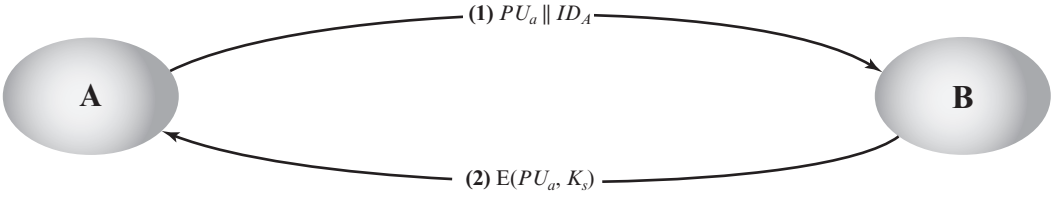


Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

case, if an adversary, D, has control of the intervening communication channel, then D can compromise the communication in the following fashion without being detected (Figure 14.8).

1. A generates a public/private key pair $\{PU_a, PR_a\}$ and transmits a message intended for B consisting of PU_a and an identifier of A, ID_A.
2. D intercepts the message, creates its own public/private key pair $\{PU_d, PR_d\}$ and transmits $PU_d \| ID_A$ to B.
3. B generates a secret key, K_s, and transmits $E(PU_d, K_s)$.
4. D intercepts the message and learns K_s by computing $D(PR_d, E(PU_d, K_s))$.
5. D transmits $E(PU_a, K_s)$ to A.

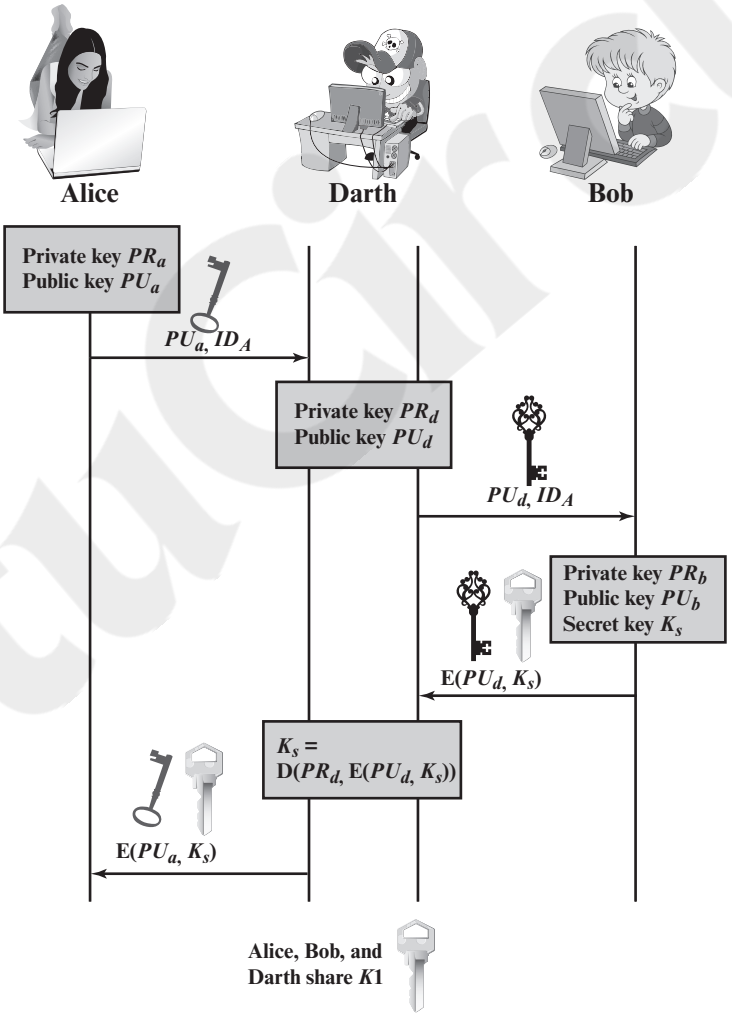


Figure 14.8 Another Man-in-the-Middle Attack

The result is that both A and B know K_s and are unaware that K_s has also been revealed to D. A and B can now exchange messages using K_s. D no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s, D can decrypt all messages, and both A and B are unaware of the problem. Thus, this simple protocol is only useful in an environment where the only threat is eavesdropping.

Secret Key Distribution with Confidentiality and Authentication

Figure 14.9, based on an approach suggested in [NEED78], provides protection against both active and passive attacks. We begin at a point when it is assumed that A and B have exchanged public keys by one of the schemes described subsequently in this chapter. Then the following steps occur.

1. A uses B's public key to encrypt a message to B containing an identifier of A(ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
2. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (1), the presence of N_1 in message (2) assures A that the correspondent is B.
3. A returns N_2, encrypted using B's public key, to assure B that its correspondent is A.
4. A selects a secret key K_s and sends $M = \text{E}(PU_b, \text{E}(PR_a, K_s))$ to B. Encryption of this message with B's public key ensures that only B can read it; encryption with A's private key ensures that only A could have sent it.
5. B computes $\text{D}(PU_a, \text{D}(PR_b, M))$ to recover the secret key.

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

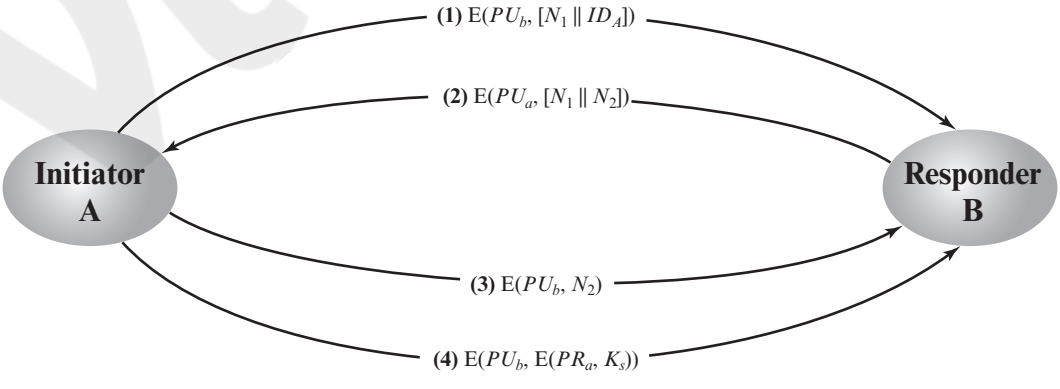


Figure 14.9 Public-Key Distribution of Secret Keys

A Hybrid Scheme

Yet another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes [LE93]. This scheme retains the use of a key distribution center (KDC) that shares a secret master key with each user and distributes secret session keys encrypted with the master key. A public-key scheme is used to distribute the master keys. The following rationale is provided for using this three-level approach:

- **Performance:** There are many applications, especially transaction-oriented applications, in which the session keys change frequently. Distribution of session keys by public-key encryption could degrade overall system performance because of the relatively high computational load of public-key encryption and decryption. With a three-level hierarchy, public-key encryption is used only occasionally to update the master key between a user and the KDC.
- **Backward compatibility:** The hybrid scheme is easily overlaid on an existing KDC scheme with minimal disruption or software changes.

The addition of a public-key layer provides a secure, efficient means of distributing master keys. This is an advantage in a configuration in which a single KDC serves a widely distributed set of users.

14.3 DISTRIBUTION OF PUBLIC KEYS

Several techniques have been proposed for the distribution of public keys. Virtually all these proposals can be grouped into the following general schemes:

- Public announcement
- Publicly available directory
- Public-key authority
- Public-key certificates

Public Announcement of Public Keys

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 (Figure 14.10). For example, because of the growing popularity of PGP (pretty good privacy, discussed in Chapter 19), which makes use of RSA, many PGP users have adopted the practice of appending their public key to messages that they send to public forums, such as USENET newsgroups and Internet mailing lists.

Although this approach is convenient, it has a major weakness. Anyone can forge such a public announcement. That is, some user could pretend to be user A and send a public key to another participant or broadcast such a public key. 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 (see Figure 9.3).

Figure 14.10 Uncontrolled Public-Key Distribution

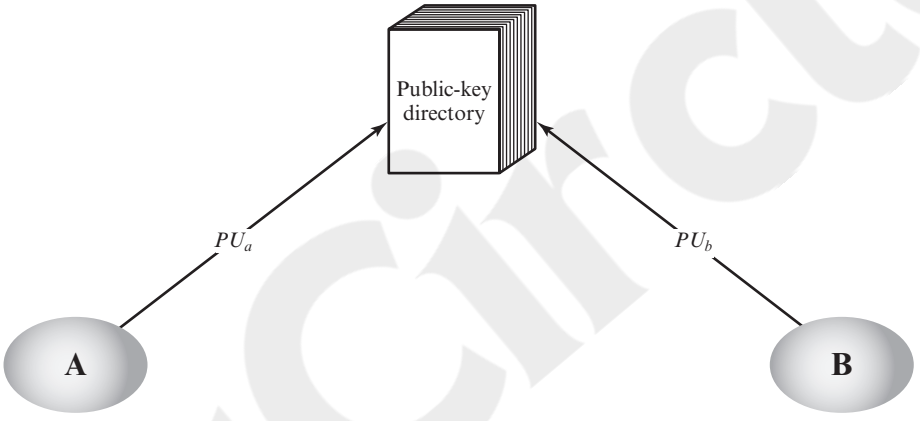


Figure 14.11 Public-Key Publication

Publicly Available Directory

A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys. Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization (Figure 14.11). Such a scheme would include the following elements:

1. The authority maintains a directory with a {name, public key} entry for each participant.
2. Each participant registers a public key with the directory authority. Registration would have to be in person or by some form of secure authenticated communication.
3. A participant may replace the existing key with a new one at any time, either because of the desire to replace a public key that has already been used for a large amount of data, or because the corresponding private key has been compromised in some way.
4. Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant. Another way to achieve the same end is for the adversary to tamper with the records kept by the authority.

Public-Key Authority

Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure 14.12, which is based on a figure in [POPE79]. As before, the scenario assumes that a central authority maintains a dynamic directory of public keys of all participants. In addition, each participant reliably knows a public key for the authority, with only the authority knowing the corresponding private key. The following steps (matched by number to Figure 14.12) occur.

1. A sends a timestamped message to the public-key authority containing a request for the current public key of B.
2. The authority responds with a message that is encrypted using the authority's private key, PR_{auth}. Thus, A is able to decrypt the message using the authority's public key. Therefore, A is assured that the message originated with the authority. The message includes the following:
   - B's public key, PU_b, which A can use to encrypt messages destined for B
   - The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority
   - The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key
3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A (ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

4, 5. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.

At this point, public keys have been securely delivered to A and B, and they may begin their protected exchange. However, two additional steps are desirable:

6. B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.
7. A returns N_2, which is encrypted using B's public key, to assure B that its correspondent is A.

Thus, a total of seven messages are required. However, the initial five messages need be used only infrequently because both A and B can save the other's public key for future use—a technique known as caching. Periodically, a user should request fresh copies of the public keys of its correspondents to ensure currency.

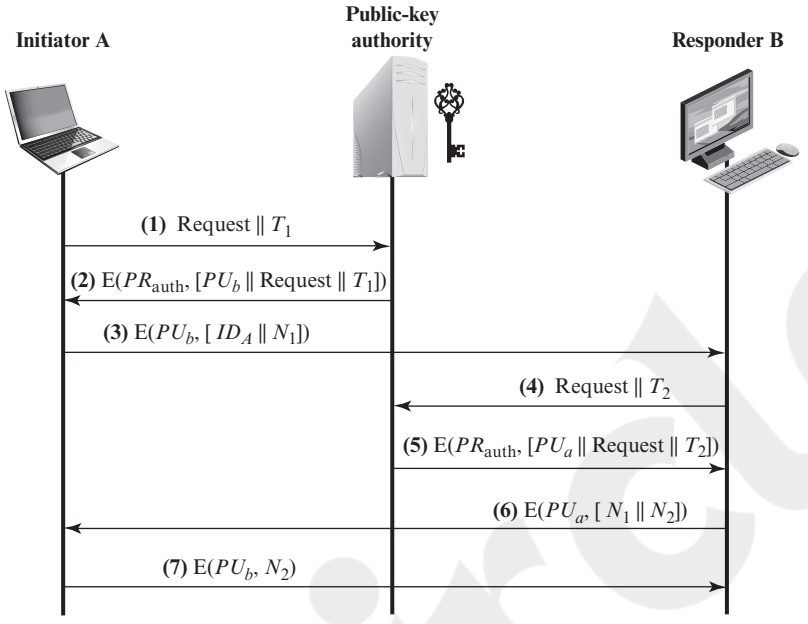


Figure 14.12 Public-Key Distribution Scenario

Public-Key Certificates

The scenario of Figure 14.12 is attractive, yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact. As before, the directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOHN78], is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority, in a way that is as reliable as if the keys were obtained directly from a public-key authority. In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution, that is trusted by the user community. A user can present his or her public key to the authority in a secure manner and obtain a certificate. The user can then publish the certificate. Anyone needing this user's public key can obtain the certificate and verify that it is valid by way of the attached trusted signature. A participant can also convey its key information to another by transmitting its certificate. Other participants can verify that the certificate was created by the authority. We can place the following requirements on this scheme:

1. Any participant can read a certificate to determine the name and public key of the certificate's owner.
2. Any participant can verify that the certificate originated from the certificate authority and is not counterfeit.
3. Only the certificate authority can create and update certificates.

These requirements are satisfied by the original proposal in [KOHN78]. Denning [DENN83] added the following additional requirement:

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

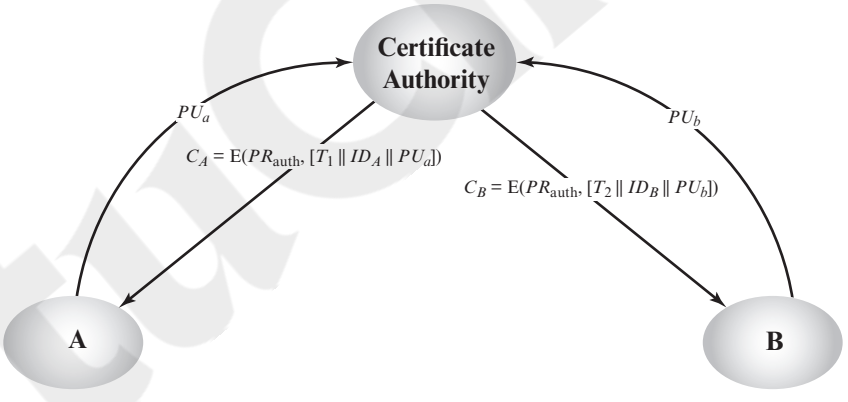
A certificate scheme is illustrated in Figure 14.13. Each participant applies to the certificate authority, supplying a public key and requesting a certificate. Application must be in person or by some form of secure authenticated communication. For participant A, the authority provides a certificate of the form

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

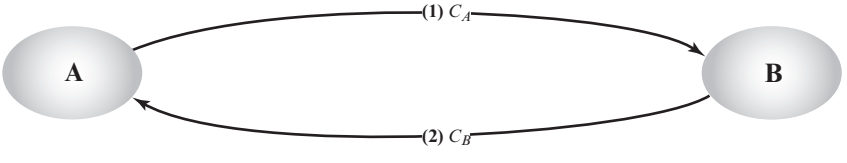
where PR_{auth} is the private key used by the authority and T is a timestamp. A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

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

The recipient uses the authority's public key, PU_{auth}, to decrypt the certificate. Because the certificate is readable only using the authority's public key, this verifies that the certificate came from the certificate authority. The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate. The timestamp



(a) Obtaining certificates from CA



(b) Exchanging certificates

Figure 14.13 Exchange of Public-Key Certificates

counters the following scenario. A's private key is learned by an adversary. A generates a new private/public key pair and applies to the certificate authority for a new certificate. Meanwhile, the adversary replays the old certificate to B. If B then encrypts messages using the compromised old public key, the adversary can read those messages.

In this context, the compromise of a private key is comparable to the loss of a credit card. The owner cancels the credit card number but is at risk until all possible communicants are aware that the old credit card is obsolete. Thus, the timestamp serves as something like an expiration date. If a certificate is sufficiently old, it is assumed to be expired.

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, transport layer security (TLS), and S/MIME, all of which are discussed in Part Five. X.509 is examined in detail in the next section.

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.

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 in 1993 to address some of the security concerns documented in [IANS90] and [MITC90]. The standard is currently at version 7, issued in 2012.

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific digital signature algorithm nor a specific hash function. Figure 14.14 illustrates the overall X.509 scheme for generation of a public-key certificate. The certificate for Bob's public key includes unique identifying information for Bob, Bob's public key, and identifying information about the CA, plus other information as explained subsequently. This information is then signed by computing a hash value of the information and generating a digital signature using the hash value and the CA's private key. X.509 indicates that the signature is formed by encrypting the hash value. This suggests the use of one of the RSA schemes discussed in Section 13.6. However, the current version of X.509 does

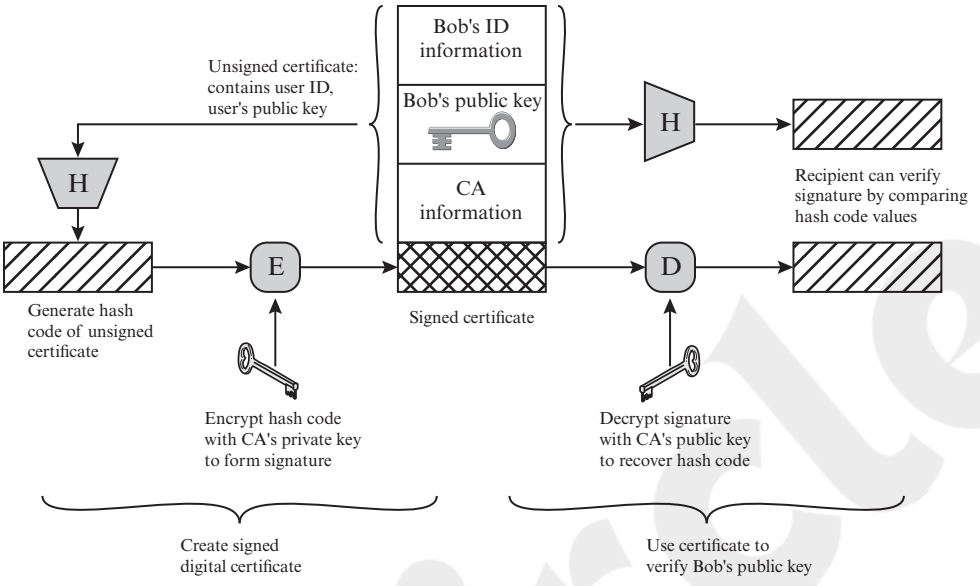


Figure 14.14 X.509 Public-Key Certificate Use

not dictate a specific digital signature algorithm. If the NIST DSA (Section 13.4) or the ECDSA (Section 13.5) scheme is used, then the hash value is not encrypted but serves as input to a digital signature generation algorithm.

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.

- **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. Although the X.509 specification is currently at version 7, no changes have been made to the fields that make up the certificate since 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.

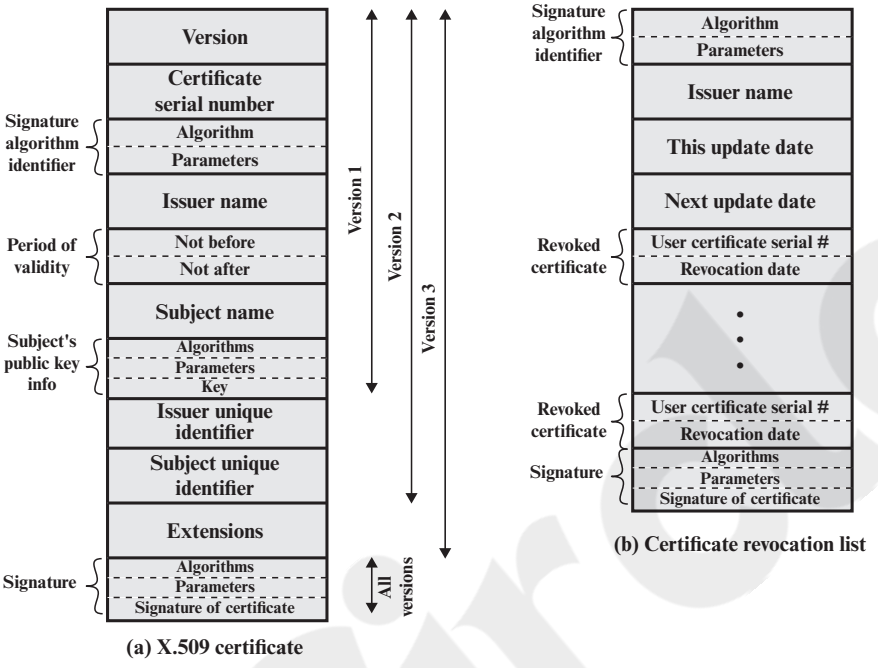


Figure 14.15 X.509 Formats

- **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. One component of this field is the digital signature applied to the other fields of the certificate. This field includes the signature algorithm identifier.

The 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

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

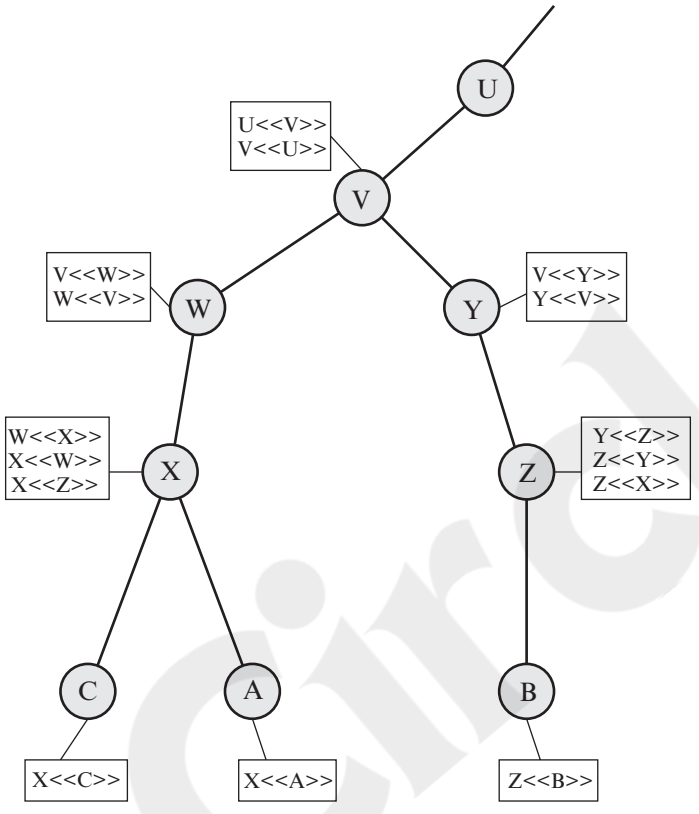


Figure 14.16 X.509 Hierarchy: A Hypothetical Example

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 email 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.
- **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.

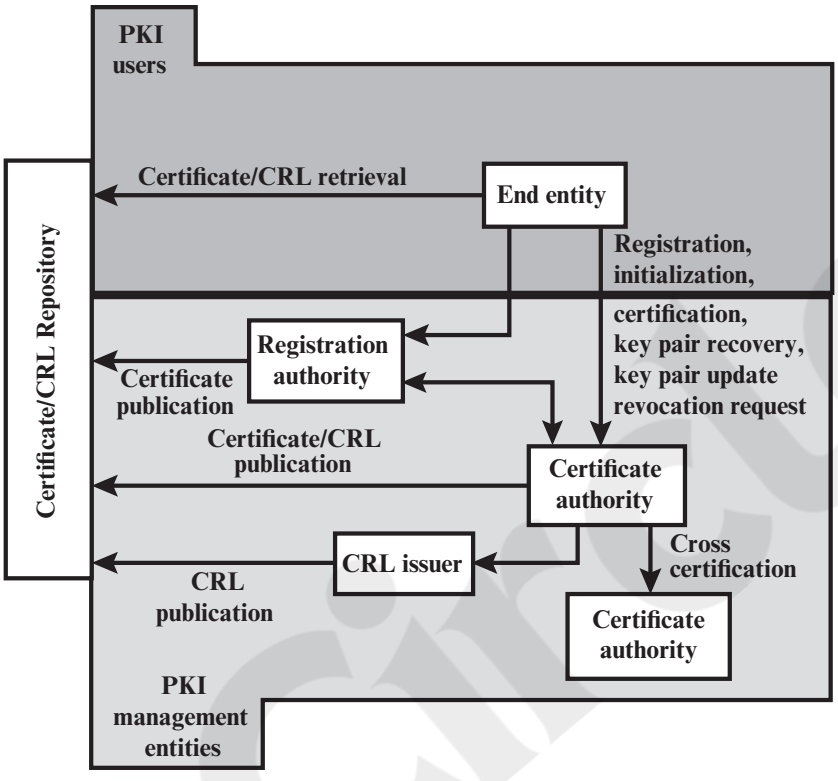


Figure 14.17 PKIX Architectural Model

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.