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 deliver- ing 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 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 poten-tially millions of keys still must be made.

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

For end-to-end   encryption,   some variation on option 4 has been widely adopted. In this scheme, a key distribution ce nter 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 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 noncryptographic way, such as physical delivery.

Figure 14.2 The Use of a Key Hierarchy

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).

Figure 14.3 Key Distribution Scenario

The diagram shows the following messages:

(1) $ID_A \| ID_B \| N_1$

(2) $E(K_a, [K_s \| ID_A \| ID_B \| N_1]) \| E(K_b, [K_s \| ID_A])$

(3) $E(K_b, [K_s \| ID_A])$

(4) $E(K_s, N_2)$

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

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, K_a, known only to itself and the KDC; similarly, B shares the master key K_b with the KDC. The following steps occur.

1.   A issues a request to the KDC for a session key to protect a logical con nection
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 time-stamp, 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 o nly one who can successfully read the message, and A knows that it originated at th e 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 matc h 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 eavesdrop-
ping. 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 at one protocol

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

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.

Figure 14.4   Automatic Key Distribution for Connection-Oriented Protocol

The steps
involved in establishing a connection are shown in Figure 14.4. When one host wi
shes to set up a connection to another host, it transmits a connec-   tion-
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 communi-   cate in a secure manner with all potential partner end systems for purposes of ses- sion 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 k ey. The
response includes the session key selected by B, an identifier of B, the value $f(N_1)$, and a nother 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 elec- tronic 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 mas- ter key is physically secured within the cryptographic hardware of the key distrib- ution 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

(a) Control vector encryption   (b) Control vector decryption

Figure 14.6   Control Vector Encryption and Decryption

consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.

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,

Hash value $= H = h(CV)$

Key input $= K_m \otimes H$

Ciphertext $= E([K_m \otimes H], K_s)$

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

$D([K_m \otimes H], E([K_m \otimes H], K_s))$

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.

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

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 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 noncryptographic way, such as physical delivery.

Figure 14.2   The Use of a Key Hierarchy

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).

Figure 14.3 Key Distribution Scenario

Let us assume that user A wishes to establish a logical connection with B and requires a one-time session key to protect the data transmitted over the connection. A has a master key, Ka, known only to itself and the KDC; similarly, B shares the master key Kb with the KDC. The following steps occur.

1.   A issues a request to the KDC for a session key to protect a logical con nection 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 time-stamp, 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 Ka. Thus, A is the o nly one who can successfully read the message, and A knows that it originated at th e KDC. The message includes two items intended for A:

•   The one-time session key, Ks, to be used for the session

•   The original request message, including the nonce, to enable A to matc h 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 eavesdrop-
ping. 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 at one protocol

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

1. Host sends packet requesting connection.
2. Security service buffers packet; asks KDC for session key.
3. KDC distributes session key to both hosts.
4. Buffered packet transmitted.

Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

The steps
involved in establishing a connection are shown in Figure 14.4. When one host wi shes to set up a connection to another host, it transmits a connec-   tion-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 communi- cate in a secure manner with all potential partner end systems for purposes of ses- sion 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 trans- ferred 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 elec- tronic 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 mas- ter key is physically secured within the cryptographic hardware of the key distrib- ution 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 pro- posed 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

(a) Control vector encryption (b) Control vector decryption

Figure 14.6 Control Vector Encryption and Decryption

consisting of a number of fields that specify the uses and restrictions for that session key. The length of the control vector may vary.

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,

Hash value $= H = h(CV)$

Key input $= K_m \otimes H$

Ciphertext $= E([K_m \otimes H], K_s)$

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

$D([K_m \otimes H], E([K_m \otimes H], K_s))$

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.

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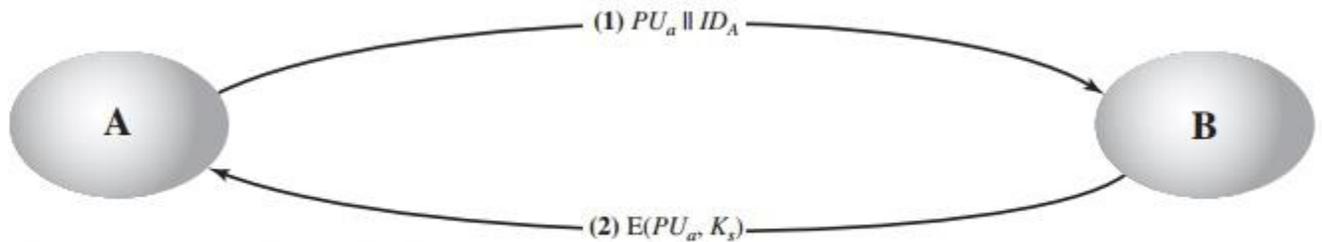
Figure 14.7 Simple Use of Public-Key Encryption to Establish a Session Key

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]. In this case, if an adversary, E, has control of the intervening communication channel, then E can compromise the communication in the following fashion without being detected.

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.   E intercepts the message, creates its own public/private key pair $\{PU_e, PR_e\}$ and transmits $PU_e \| ID_A$ to B.

B generates a secret key, K_s, and transmits $E(PU_e, K_s)$.

E intercepts the message and learns K_s by computing $D(PR_e, E(PU_e, K_s))$.

E transmits $E(PU_a, K_s)$ to A.

The result is that both A and B know K_s and are unaware that K_s has also been revealed to E. A and B can now exchange messages using K_s. E no longer actively interferes with the communications channel but simply eavesdrops. Knowing K_s, E 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 14.8, 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.
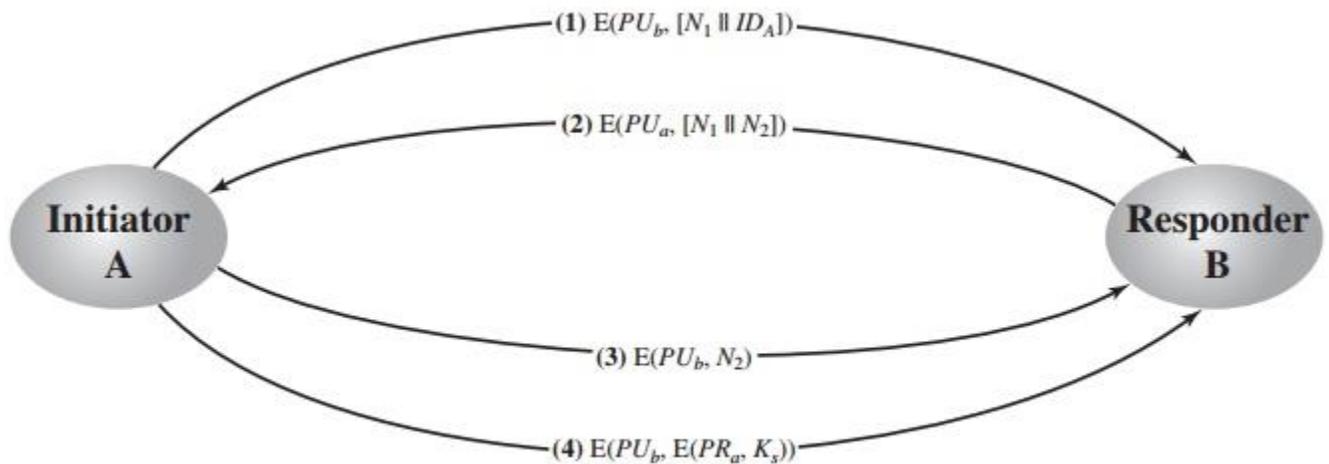
Figure 14.8   Public-Key Distribution of Secret Keys

1. A uses B's public key to encrypt a message to B containing an identifier of A(ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.

B sends a message to A encrypted with PU_a and containing A's nonce (N_1) as ell as a new nonce generated by B (N_2). Because only B could have (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.

2. A returns N_2, encrypted using B's public key, to assure B that its corresp ondent is A.

A selects a secret key K_s and sends $M = E(PU_b, E(PR_a, K_s))$ 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.

3. B computes $D(PU_a, D(PR_b, M))$ to recover the secret key.

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

A Hybrid Scheme

Yet another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes [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 s cheme 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 s es- sion 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 distrib- uting master keys. This is an advantage in a configuration in which a single KDC serves a widely distributed set of users.

DISTRIBUTION OF PUBLIC KEYS

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

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

Figure 14.9   Uncontrolled Public-Key Distribution

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 par-
ticipant can send his or her public key to any other participant or broadcast the key to the community at large (Figure 14.9). For example, because of the growing popularity of PGP (pretty good privacy, discussed in Chapter 18), 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).

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 14.10). 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.
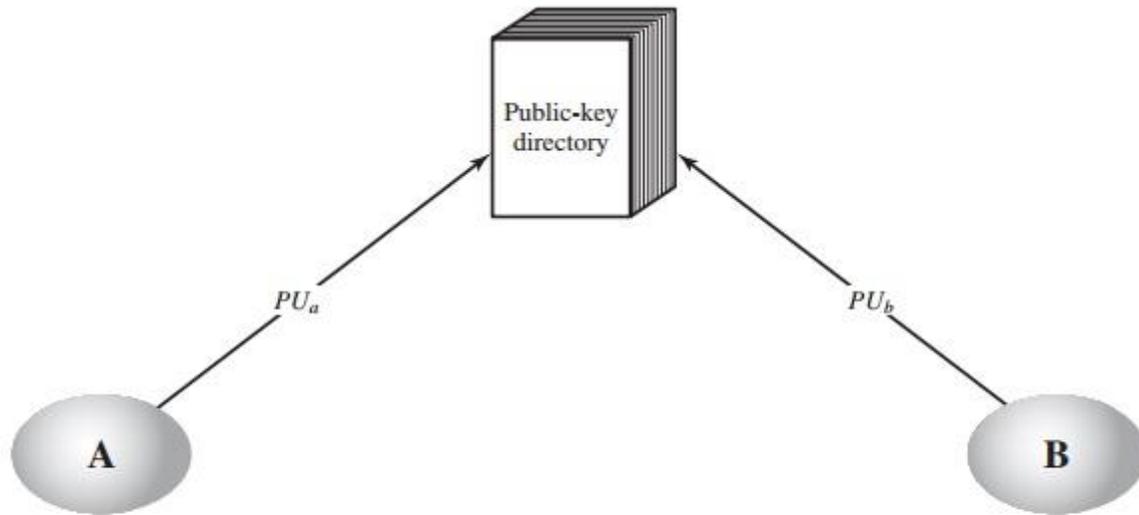
Figure 14.10   Public-Key Publication

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 h as 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 eaves- drop 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 14.11, 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 14.11) 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 authori ty's pri-

vate 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 cor- responding earlier request and to verify that the original request was not altered before reception by the authority
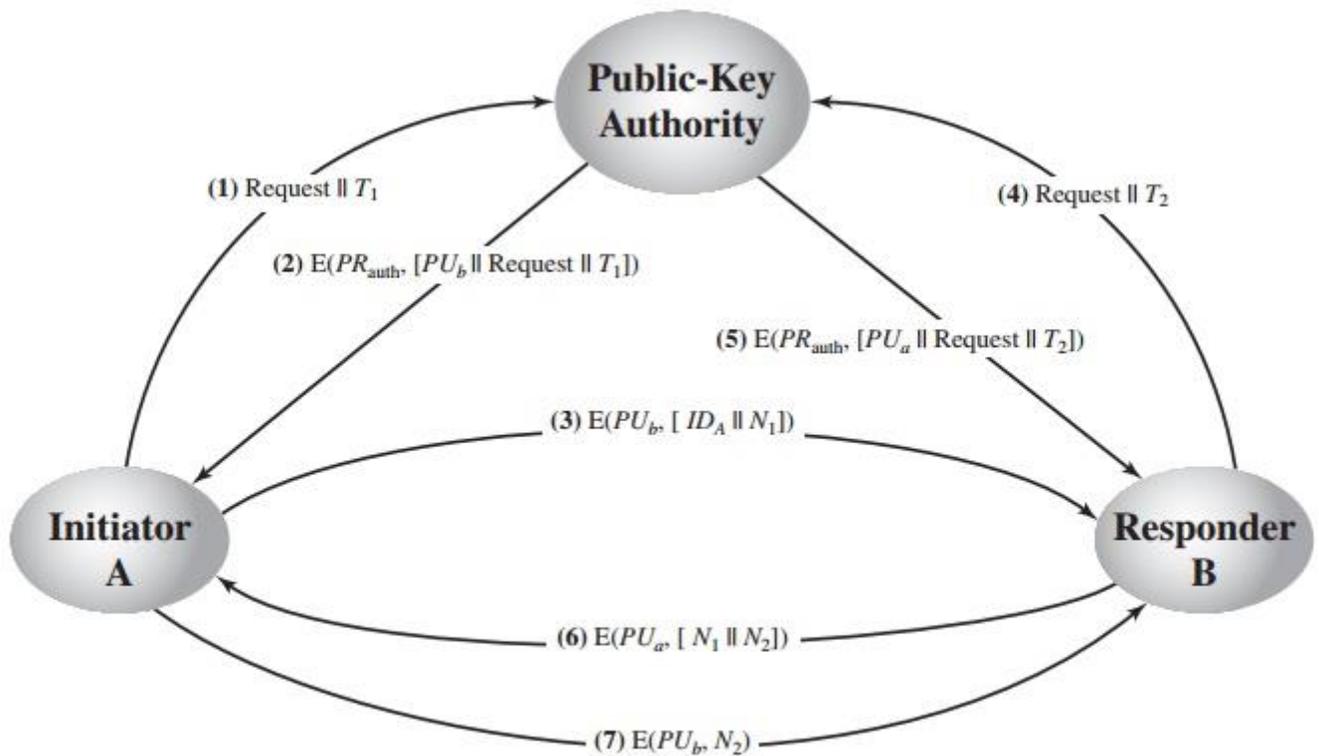
Figure 14.11 Public-Key Distribution Scenario

The diagram labels:

Public-Key Authority

(1) Request || T_1

(2) $E(PR_{auth}, [PU_b || Request || T_1])$

(4) Request || T_2

(5) $E(PR_{auth}, [PU_a || Request || T_2])$

(3) $E(PU_b, [ID_A || N_1])$

Initiator A

Responder B

(6) $E(PU_a, [N_1 || N_2])$

(7) $E(PU_b, N_2)$

- 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 PUa and containing A's nonce (N_1) as well as a new nonce generated by B (N_2). Because only B could have decrypted message (3), the presence of N_1 in message (6) assures A that the correspondent is B.

6.　　　　　A returns $N2$, which is encrypted using B's public key, to assure B that its cor- respondent is A.

Thus, a total of seven messages are required. However, the initial four mes- sages 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.

Public-Key Certificates

The scenario of Figure 14.11 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 vul- nerable to tampering.

An alternative approach, first suggested by Kohnfelder [KOHN78], is to use **certificates** that can be used by participants to exchange keys without contact ing 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 cer- tificate. The user can then publish the certificate. Anyone needing this user's pub- lic 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 currency of the certificate.

A certificate scheme is illustrated in Figure 14.12. Each participant applies to the ce rtificate authority, supplying a public key and requesting a certificate.
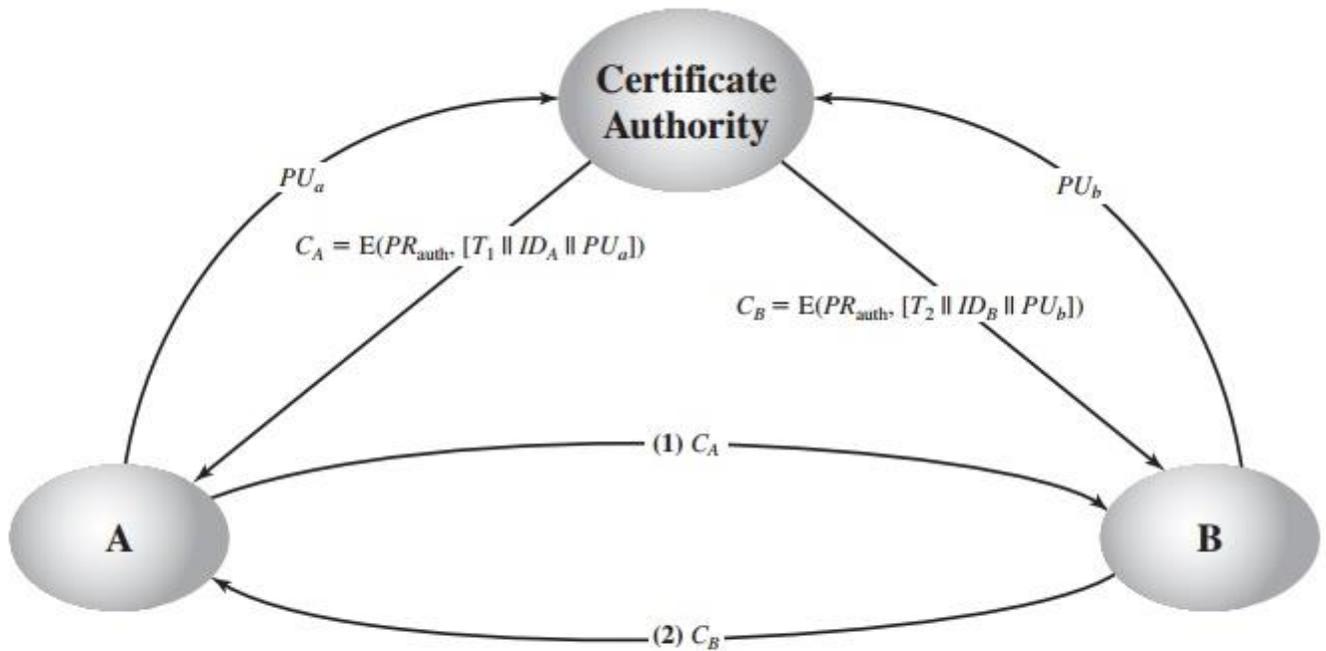
Figure 14.12   Exchange of Public-Key Certificates

Application must be in person or by some form of secure authenticated communi- cation. For participant A, the authority provides a certificate of the form

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

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 cer- tificate as follows:

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

The recipient uses the authority's public key, PU_{auth}, to decrypt the certifi- cate. 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
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.