MODULE 4:

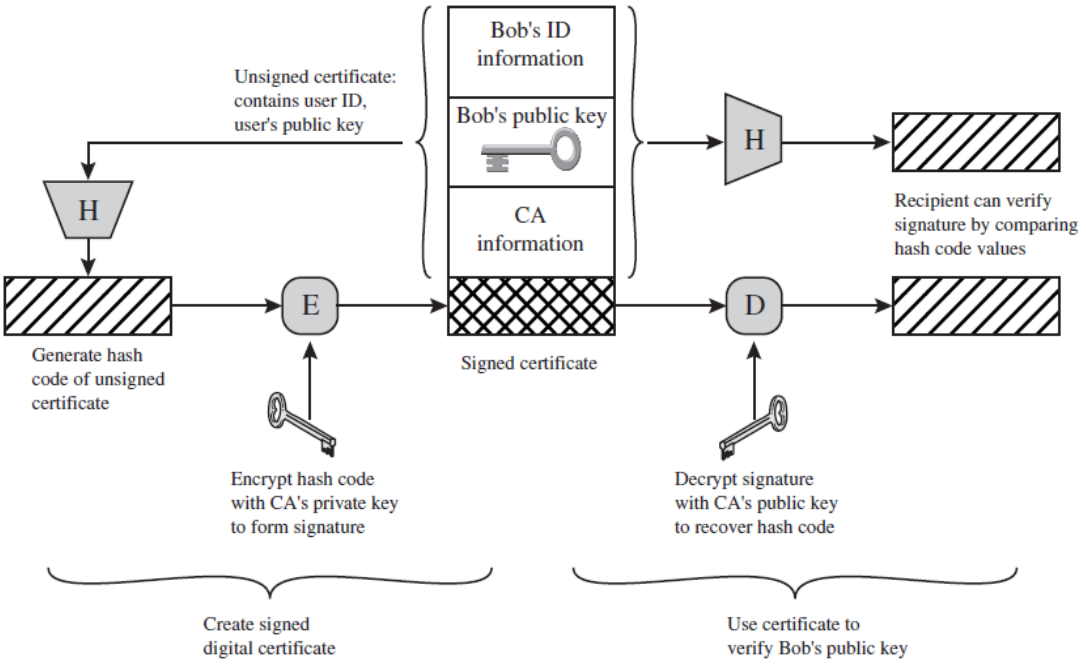
X.509 CERTIFICATES

- X.509 is part of the X.500 series of recommendations that define a directory service, being a server or distributed set of servers that maintains a database of information about 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

- Eachcertificate 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 based on the use of public-key cryptography and digital signatures.

- The X.509 certificate format is widely used, in for example S/MIME, IP Security and SSL/TLS and SET. X.509 was initially issued in 1988. The standard was ubsequently revisedto address some of the security concerns; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.



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 providesan easily accessible location for users to obtain certificates.

The standard uses the notation for a certificate of:

CA<<A>> where the CA signs the certificate for user A with its private key. In more detail CA<<A>> = CA {V, SN, AI, CA, UCA, A, UA, Ap, TA}.

If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

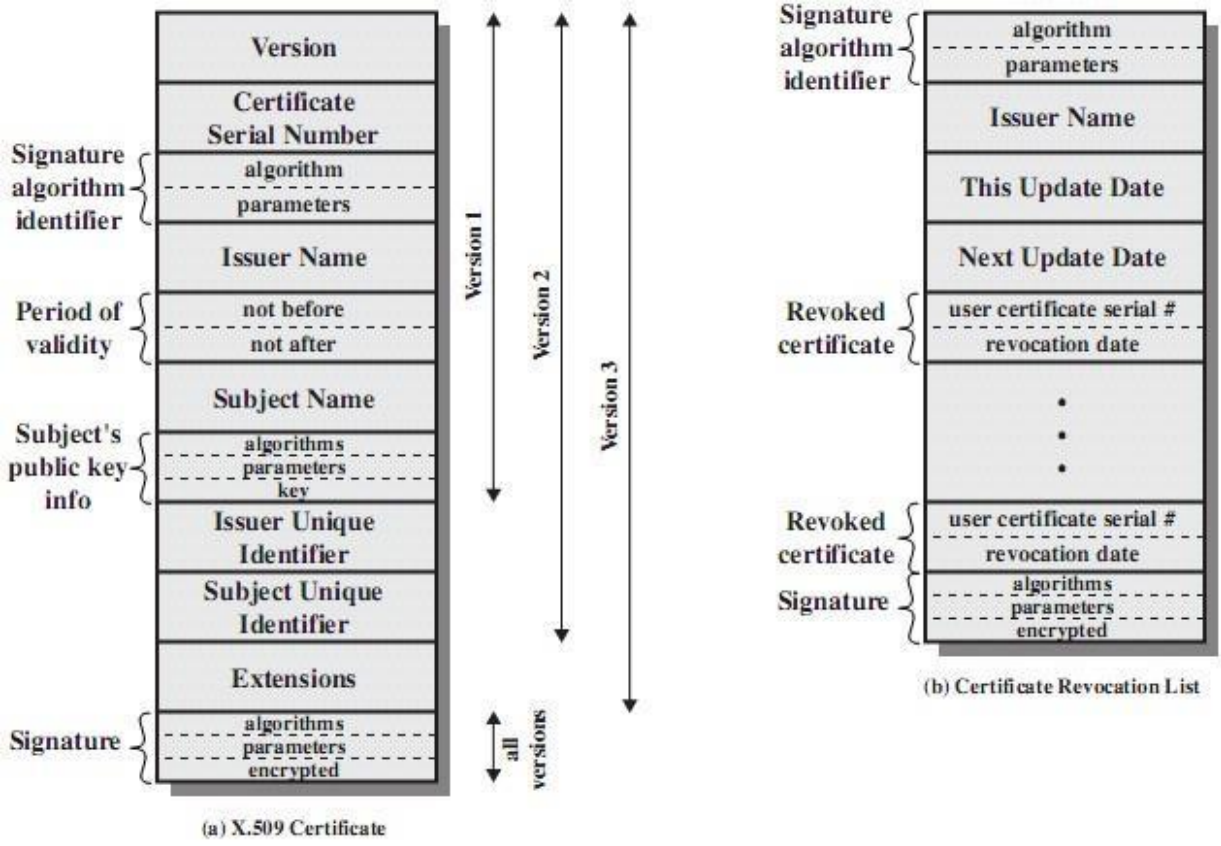


Figure 14.4 X.509 Formats

- **Version:** Differentiates among successive versions of the certificate format; the default isversion 1. If the issuer unique identifier or subject unique identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.
- **Serial number:** An integer value unique within the issuing CA that is unambiguouslyassociated with this certificate.
- **Signature algorithm identifier:** The algorithm used to sign the certificate together withany associated parameters. Because this information is repeated in the signature field at the end of the certificate, this field has little, if any, utility.
- **Issuer name:** X.500 is the 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 isvalid.
- **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 ofthe 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 theissuing 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 thesubject 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 3and are discussed later in this section.
- **Signature:** Covers all of the other fields of the certificate; it contains the hash code of theother fields encrypted with the CA's private key. This field includes the signature algorithm identifier.

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used. The standard uses the following notation to define a certificate:

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

where

$Y\langle\langle X\rangle\rangle$ = 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

CAA = name of user A

UA = optional unique identifier of the user

AAp = public key of user A

T^A = period of validity of the certificate

Obtaining a Certificate

User certificates generated by a CA have the following characteristics:

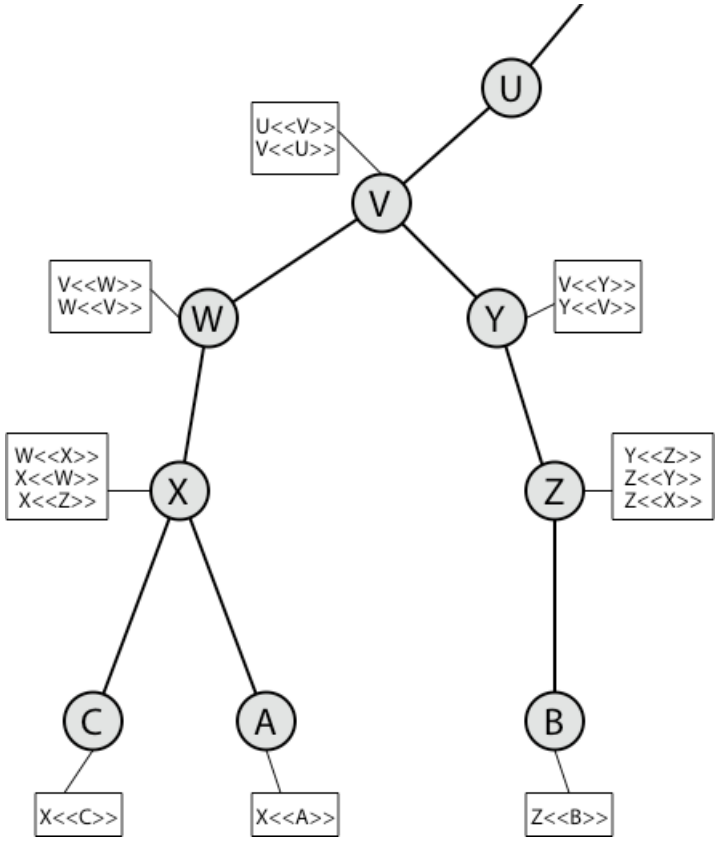
- Any user with access to the public key of the CA can verify the user public keythat was certified.
- No party other than the certification authority can modify the certificate without this being detected.

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 ofA's certificate, B has confidence that messages it encrypts with A's public key will be securefrom eavesdropping and that messages signed with A's private key are unforgeable.

CA Hierarchy:

If both parties use the same CA, they know its public key and can verify others certificates. If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Hence there has to be some means to form a chain of certificationsbetween the CA's used by the two parties, by the use of client and parent certificates. 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. It is assumed that each client trusts its parent's certificates.



Above Figure illustrates the use of an X.509 hierarchy to mutually verify clients certificates. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. Thedirectory 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, we can track chains of certificates as follows:

A acquires B certificate using chain

X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

B acquires A certificate using chain:

Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>

Certificate Revocation:

A certificate includes a period of validity. 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 a range of 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.

To support this, each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, known as the certificate revocation list (CRL). Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes the issuer's name, the date the listwas 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, by checking the directory CRL each time a certificate is received, this often does not happen in practice.

X.509 Version 3

The X.509 version 2 format does not convey all of the information. Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. X.509 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 indicatorindicates whether an extension can be safely ignored or not.

Certificate Extensions

The certificate extensions fall into three main categories:

- **Key and policy information** - convey additional information about the subject andissuer 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.
- **Subject and issuer attributes** - support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject; eg. postal address, email address, or picture image
- **Certification path constraints** - allow constraint specifications to be included in certificates issued for CA's by other CA's that may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

USER AUTHENTICATION

An authentication process consists of two steps:

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

In essence, identification is the means by which a user provides a claimed identity to the system; user authentication is the means of establishing the validity of the claim. Note that user authentication is distinct from message authentication.

There are four general means of authenticating a user's identity, which can be usedalone or in combination:

• **Something the individual knows:** Examples includes a password, a personalidentification number (PIN), or answers to a prearranged set of questions.

• **Something the individual possesses:** Examples include electronic keycards, smartcards, and physical keys. This type of authenticator is referred to as a *token.*

• **Something the individual is (static biometrics):** Examples include recognition by fingerprint, retina, and face.

• **Something the individual does (dynamic biometrics):** Examples include recognitionby voice pattern, handwriting characteristics, and typing rhythm.

All of these methods, properly implemented and used, can provide secure user authentication. However, each method has problems. An adversary may be able to guess or steal a password. Similarly, an adversary may be able to forge or steal a token. A user may forget a password or lose a token. Further, there is a significant administrative overhead for managing password and token information on systems and securing such information on systems. With respect to biometric authenticators, there are a variety of problems, including dealing with false positives and false negatives, user acceptance,cost, and convenience.

Authentication Protocols

- An important application area is that of **mutual authentication** protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other'sidentity and to exchange session keys. There, the focus was key distribution.
- Central to the problem of authenticated key exchange are two issues: **confidentiality and timeliness.**
- To prevent masquerade and to preventcompromise of session keys, essential identification and session key information must be communicated in encrypted form. The second issue, timeliness, is important because of the threat of message replays.

Replay Attacks: are where a valid signed message is copied and later resent. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

Examples of replay attacks:

Simple replay: The opponent simply copies a message and replays it later.

Repetition that can be logged: An opponent can replay a timestamped message withinthe valid time window.

Repetition that cannot be detected: This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only thereplay message arrives.

Backward replay without modification: This is a replay back to the message sender.This attack

is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis ofcontent.

Possible countermeasures include the use of:

• **Sequence numbers** (generally impractical since must remember last number used with every communicating party)

• **Timestamps** (needs synchronized clocks amongst all parties involved, which can beproblematic)

• **Challenge/response** (using unique, random, unpredictable nonce, but not suitable for connectionless applications because of handshake overhead)

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it. Accordingly, the e-mail message should be encrypted suchthat the mail- handling system is not in possession of the decryption key. A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

REMOTE USER-AUTHENTICATION USING SYMMETRICENCRYPTION

Mutual Authentication

As discussed earlier, A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. Usually involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret master key with the KDC.

The KDC is responsible for generating session keys, and for distributing those keys to the parties involved, using the master keys to protect these session keys.

Needham-Schroeder Protocol

The Needham-Schroeder Protocol is the original, basic key exchange protocol. Used by 2 parties who both trusted a common key server, it gives one party the info needed to establish a session key with the other.

Note that all communications is between A&KDC and A&B, B&KDC don't talk directly (though indirectly a message passes from KDC via A to B, encrypted in B's key so that A is unable to read or alter it). Other variations of key distribution protocols can involve direct communications between B&KDC.

1. A→KDC: $ID_A \| ID_B \| N_1$

2. KDC → A: $E(K_a,[K_s \| ID_B \| N_1 \| E(K_b,[K_s \| ID_A])])$

3. A → B: $E(K_b, [K_s \| ID_A])$

4. B → A: $E(K_s, [N_2])$

5. A → B: $E(K_s, [f(N_2)])$

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B.

There is a critical flaw in the protocol, as shown. The message in step 3 can be decrypted, and hence understood only by B. But if an opponent, X, has been able to compromise an old session key, then X can impersonate A and trick B into using the old key by simply replaying step 3. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3.

Denning proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, K_a and K_b are secure, and it consists of the

following steps.

1. A→ KDC: $ID_A || ID_B$

2. KDC→ A: $E(K_a, [K_S || ID_B || T || E(K_b, [K_S || ID_A || T])])$

3. A→ B: $E(K_b, [K_S || ID_A || T])$

4. B→A: $E(K_S, N1)$

5. A→B: $E(K_S, f(N1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange.

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

The problem occurs **when a sender's clock is ahead of the intended recipient's clock.** In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks.**

REMOTE USER AUTHENTICATION USING ASYMMETRICENCRYPTION

Mutual Authentication

This protocol assumes that each ofthe two parties is in possession of the current

public key of the other. It may not be practical to require this assumption.

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

A protocol using timestamps is provided in that uses a central system, referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestampsprotect against replays of compromised keys. See text for details. This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam, makes use of nonces.

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

Note the authors themselves spotted a flaw in it and submitted a revised version of the algorithm in. In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for

confidentiality, authentication, or both. These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication).

If confidentiality is the primary concern, then better to encrypt the message with a one-time secret key, and also encrypt this one-time key with B's public key. If authentication is the primary concern, then a digital signature may suffice (but could be replaced by an opponent). To counter such a scheme, both the message and signature can be encrypted with the recipient's public key. The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate.

KERBEROS (Important Topic)

Kerberos is an authentication service developed as part of Project Athena at MIT, and is one of the best known and most widely implemented **trusted third party** key distribution systems.

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Unlike most other authentication schemes, Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption. Two versions of Kerberos are in common use: v4 & v5.

In a more open environment, in which network connections to other machines are supported, an approach that requires the user to prove his or her identity for each service invoked, and also require that servers prove their identity to clients, is needed to protect user information and resources housed at the server. Kerberos supports this approach, and assumes a distributed client/server architecture that employs one or more Kerberos servers to provide an authentication service. The first published report on Kerberos[STEI88] listed the following requirements:

• **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user.

• **Reliable**: For all services that rely on Kerberos for access control, lack of availability of the Kerberos service means lack of availability of the supported services. Hence, Kerberos should be highly reliable and should employ a distributed server architecture, with one system able to back up another.

• **Transparent**: Ideally, the user should not be aware that authentication is taking place, beyond the requirement to enter a password.

• **Scalable**: The system should be capable of supporting large numbers of clients and servers. This suggests a modular, distributed architecture.

To support these requirements, Kerberos is a trusted third-party authentication service that uses a protocol based on that proposed by Needham and Schroeder which was discussed earlier in this chapter.

The Version 4 Authentication Dialogue

- Kerberos V4 is a basic third-party authentication scheme.
- The core of Kerberos is the **Authentication server** (AS) and **Ticket Granting Servers (TGS)** – these are trusted by all users and servers and must be securely administered.
- The protocol includes a sequence of interactions between the client, AS, TGT and desired server. Version 4 of Kerberos makes use of DES, in a rather elaborate protocol, to provide the authentication service

The heart of the first problem is the lifetime associated with the ticket-granting ticket. If this lifetime is very short (e.g., minutes), then the user will be repeatedly asked for a password. If the lifetime is long (e.g., hours), then an opponent has a greateropportunity for replay. Similarly, if an opponent captures a service-granting ticket and uses it before it

expires, the opponent has access to the corresponding service.

The second problem is that there may be a requirement for servers to authenticate themselves to users.

First, consider the problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information. Table 15.1a shows the technique for distributing the session key.

Table 14.1 Summary of Kerberos Version 4 Message Exchanges

(1) C → AS $ID_c \| ID_{tgs} \| TS_1$

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

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

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

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

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

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

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

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

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

(5) C → V $Ticket_v \| Authenticator_c$

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

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

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

(c) Client/Server Authentication Exchange to obtain service

Table a shows the technique for distributing the session key. As before, the client sends a message to the AS requesting access to the TGS. The AS responds with a message,

encrypted with a key derived from the user's password (Kc) that contains the ticket. The encrypted message also contains a copy of the session key,(Kc,tgs), where thesubscripts indicate that this is a session key for C and TGS. Because this session key is inside the message encrypted with (Kc), only the user's client can read it.The same session key is included in the ticket, which can be read only by the TGS. Thus, the session key has been securely delivered to both C and the TGS.

Note that several additional pieces of information have been added to this first phase of the dialogue. Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity.

C sends the TGS a message that includes the ticket plus the ID of the requested service (message (3) in Table b). In addition, C transmits an authenticator, which includes the ID and address of C's user and a timestamp. Unlike the ticket, which is reusable, the authenticator is intended for use only once and has a very short lifetime. TheTGS can decrypt the ticket with the key that it shares with the AS. This ticket indicates that user C has been provided with the session key Kc,tgs. In effect, the ticket says,"Anyone who uses Kc,tgs must be C."The TGS uses the session key to decrypt the authenticator.

The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator. The server can decrypt the ticket, recover the session key, and decrypt the authenticator. If mutual authentication is required, the server can reply as shown in message (6).

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

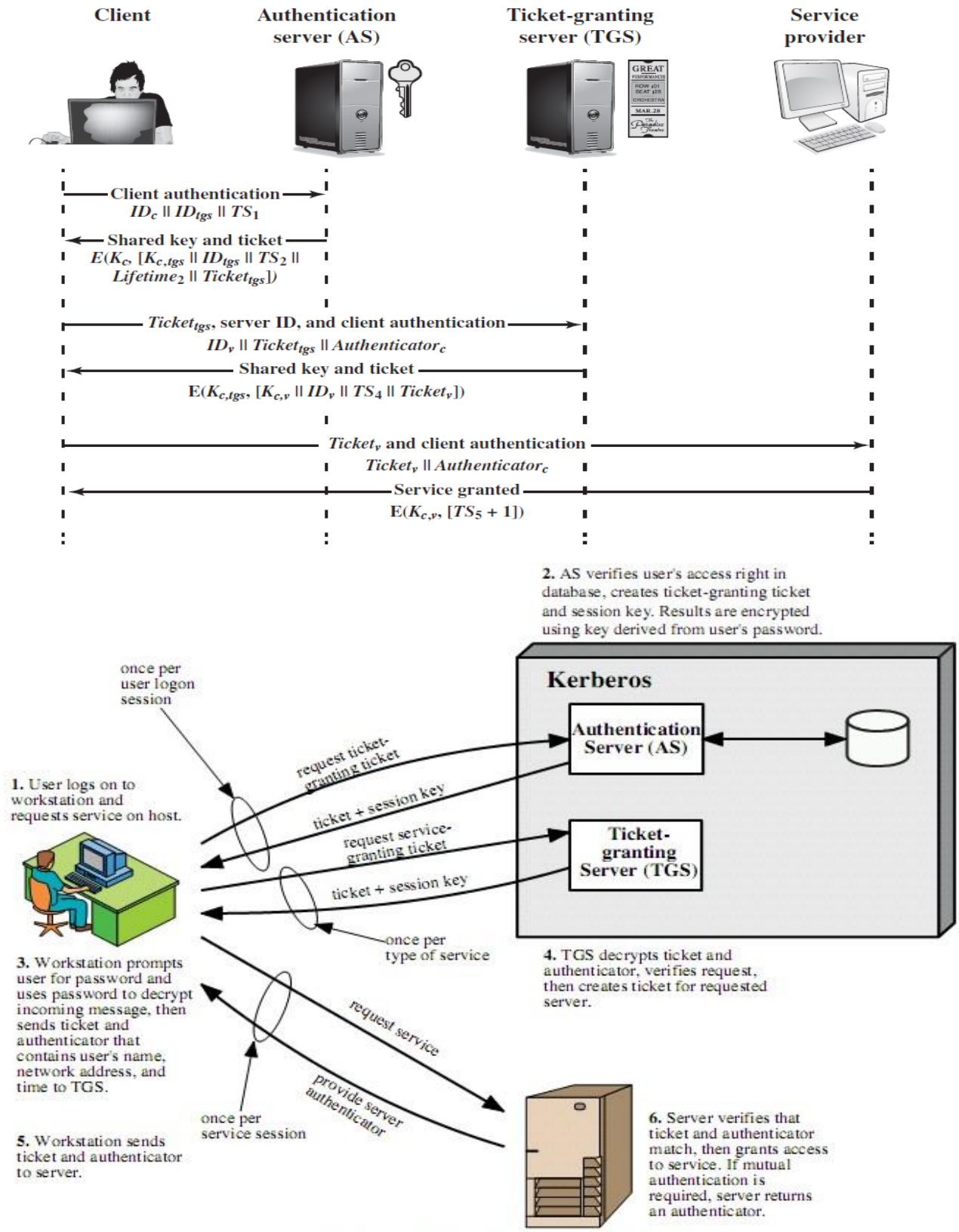


Figure 14.1 Overview of Kerberos

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

(a) Authentication Service Exchange

Message (3)	Client requests service-granting ticket.
ID_V	Tells TGS that user requests access to server V.
$Ticket_{tgs}$	Assures TGS that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (4)	TGS returns service-granting ticket.
$K_{c,tgs}$	Key shared only by C and TGS protects contents of message (4).
$K_{c,v}$	Copy of session key accessible to client created by TGS to permit secure exchange between client and server without requiring them to share a permanent key.
ID_V	Confirms that this ticket is for server V.
TS_4	Informs client of time this ticket was issued.
$Ticket_V$	Ticket to be used by client to access server V.
$Ticket_{tgs}$	Reusable so that user does not have to reenter password.
K_{tgs}	Ticket is encrypted with key known only to AS and TGS, to prevent tampering.

$K_{c,tgs}$	Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_{tgs}	Assures server that it has decrypted ticket properly.
TS_2	Informs TGS of time this ticket was issued.
$Lifetime_2$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures TGS that the ticket presenter is the same as the client for whom the ticket was issued has very short lifetime to prevent replay.
$K_{c,tgs}$	Authenticator is encrypted with key known only to client and TGS, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_3	Informs TGS of time this authenticator was generated.

(b) Ticket-Granting Service Exchange

Message (5)	Client requests service.
$Ticket_V$	Assures server that this user has been authenticated by AS.
$Authenticator_c$	Generated by client to validate ticket.
Message (6)	Optional authentication of server to client.
$K_{c,v}$	Assures C that this message is from V.
$TS_5 + 1$	Assures C that this is not a replay of an old reply.
$Ticket_v$	Reusable so that client does not need to request a new ticket from TGS for each access to the same server.
K_v	Ticket is encrypted with key known only to TGS and server, to prevent tampering.
$K_{c,v}$	Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket.
ID_C	Indicates the rightful owner of this ticket.
AD_C	Prevents use of ticket from workstation other than one that initially requested the ticket.
ID_V	Assures server that it has decrypted ticket properly.

TS_4	Informs server of time this ticket was issued.
$Lifetime_4$	Prevents replay after ticket has expired.
$Authenticator_c$	Assures server that the ticket presenter is the same as the client for whom the ticket was issued; has very short lifetime to prevent replay.
$K_{c,v}$	Authenticator is encrypted with key known only to client and server, to prevent tampering.
ID_C	Must match ID in ticket to authenticate ticket.
AD_C	Must match address in ticket to authenticate ticket.
TS_5	Informs server of time this authenticator was generated.

(c) Client/Server Authentication Exchange

KERBEROS REALMS:

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database.All users are registered with the Kerberos server.

2. The Kerberos server must share a secret key with each server.All servers are registered with the Kerberos server.

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

A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers is referred to as a Kerberos realm. A Kerberos realm is a set of managed nodes that share the same Kerberos database, and are part of the same administrative domain. If have multiple realms, their Kerberos servers must share keys and trust each other.

The details of the exchanges illustrated in below Figure are as follows

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

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

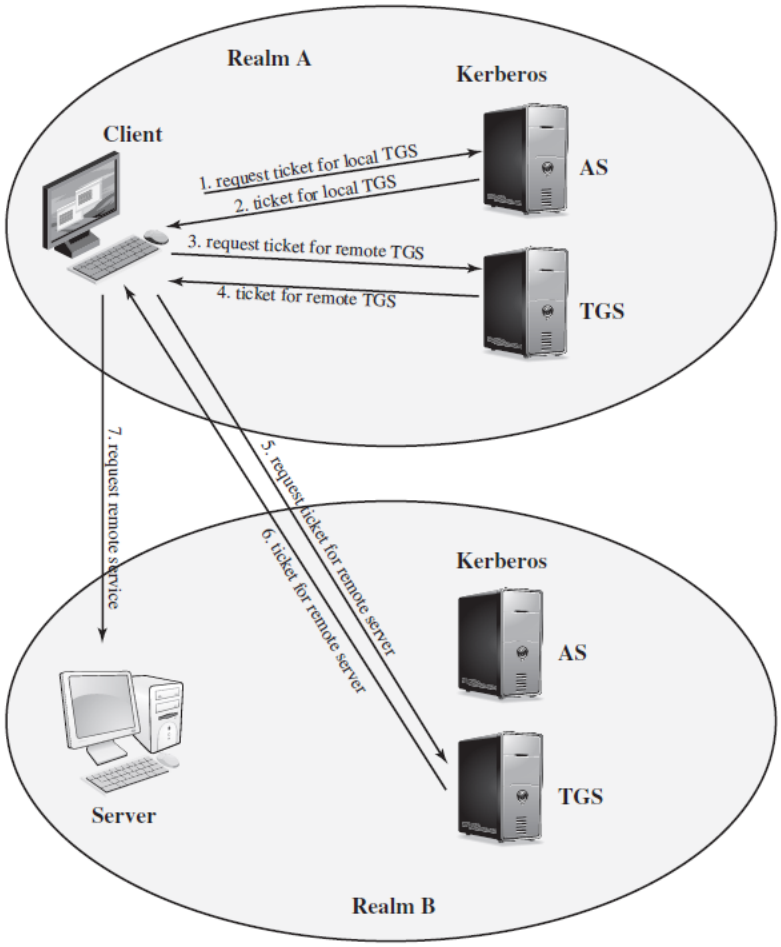
(3) $C \rightarrow TGS$: $ID_{tgsrem} \| Ticket_{tgs} \| Authenticator_c$

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

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

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

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



The Version 5 Authentication Dialogue

Table 15.3 Summary of Kerberos Version 5 Message Exchanges

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(c) Client/Server Authentication Exchange to obtain service

INITIAL	This ticket was issued using the AS protocol and not issued based on a ticket-granting ticket.
PRE-AUTHENT	During initial authentication, the client was authenticated by the KDC before a ticket was issued.
HW-AUTHENT	The protocol employed for initial authentication required the use of hardware expected to be possessed solely by the named client.
RENEWABLE	Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
MAY-POSTDATE	Tells TGS that a postdated ticket may be issued based on this ticket-granting ticket.
POSTDATED	Indicates that this ticket has been postdated; the end server can check the authtime field to see when the original authentication occurred.
INVALID	This ticket is invalid and must be validated by the KDC before use.
PROXIABLE	Tells TGS that a new service-granting ticket with a different network address may be issued based on the presented ticket.
PROXY	Indicates that this ticket is a proxy.
FORWARDABLE	Tells TGS that a new ticket-granting ticket with a different network address may be issued based on this ticket-granting ticket.
FORWARDED	Indicates that this ticket has either been forwarded or was issued based on authentication involving a forwarded ticket-granting ticket.

- Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new elements are added:
  - Realm: Indicates realm of user
  - Options: Used to request that certain flags be set in the returned ticket
  - Times: Used by the client to request the following time settings in the ticket:

  —from: the desired start time for the requested ticket

  —till: the requested expiration time for the requested ticket

  —rtime: requested renew-till time
  - Nonce: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent.
- Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password. This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information. The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options. These flags introduce significant new functionality to version 5.
- Message (3) for both versions includes an authenticator, a ticket, and the name of the requested service. In addition, version 5 includes requested times and options for the ticket and a nonce—all with functions similar to those of message 1).
- Message (4) has the same structure as message (2). It returns a ticket plus information needed by the client, with the information encrypted using the session key now shared by the client and the TGS.
- Finally, for the client/server authentication exchange, several new features appear in version 5. In message (5), the client may request as an option that mutual authentication is required. The authenticator includes several new fields:
  - Subkey: The client's choice for an encryption key to be used to protect this specific application session. If this field is omitted, the session key from the ticket (Kc,v) is used.
  - Sequence number: An optional field that specifies the starting sequence number to be used by the server for messages sent to the client during this session. Messages may be sequence numbered to detect replays.

DIFFERENCES BETWEEN VERSIONS 4 AND 5

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

Environmental shortcomings.

1. Encryption system dependence: Version 4 requires the use of DES. Export restrictionon DES as well as doubts about the strength of DES were thus of concern. In version 5, ciphertext is tagged with an encryption-type identifier so that any encryption technique may be used.

2. Internet protocol dependence: Version 4 requires the use of Internet Protocol (IP) addresses. Other address types, such as the ISO network address, are not accommodated. Version 5 network addresses are tagged with type and length, allowing any network address type to be used.

3. Message byte ordering: In version 4, the sender of a message employs a byte orderingof its own choosing and tags the message to indicate least significant byte in lowest address or most significant byte in lowest address. This techniques works but does not follow established conventions. In version 5, all message structures are defined using Abstract Syntax Notation One (ASN.1) andBasic Encoding Rules (BER), which provide an unambiguous byte ordering.

4. Ticket lifetime: Lifetime values in version 4 are encoded in an 8-bit quantity in units of five minutes. Thus, the maximum lifetime that can be expressed is $2^8 * 5 = 1280$ minutes (a little over 21 hours).This may be inadequate for some applications. In version 5, tickets include an explicit start time and end time, allowing tickets with arbitrary lifetimes.

5. Authentication forwarding: Version 4 does not allow credentials issued to one client to be forwarded to some other host and used by some other client. This capability would enable a client to access a server and have that server access another server on behalf of the client. For example, a client issues a request to a print server that then accesses the client's file from a file server, using the client's credentials for access. Version 5 providesthis capability.

6. Interrealm authentication: In version 4, interoperability among N realms requires on the order of N^2 Kerberos-to-Kerberos relationships, as described earlier. Version 5 supports a method that requires fewer relationships, as described shortly.

Technical deficiencies:

1. Double encryption: Note in Table 15.1 [messages (2) and (4)] that tickets provided to clients are encrypted twice - once with the secret key of the target server and then again with a secret key known to the client. The second encryption is not necessary and is computationally wasteful.

2. PCBC encryption: Encryption in version 4 makes use of a nonstandard mode of DES known as propagating cipher block chaining (PCBC).It has been demonstrated that this mode is vulnerable to an attack involving the interchange of ciphertext blocks. Version 5 provides explicit integrity mechanisms, a checksum or hash code is attached to themessage prior to encryption using CBC.

3. Session keys: Each ticket includes a session key that is used by the client to encryptthe authenticator sent to the service associated with that ticket. In addition, the session key may subsequently be used by the client and the server to protect messages passed during that session. However, because the same ticket may be used repeatedly to gain service from a particular server, there is the risk that an opponent will replay messages from an old session to the client or the server. In version 5, it is possible for a client and server to negotiate a subsession key, which is to be used only for that one connection.

4. Password attacks: Both versions are vulnerable to a password attack. The message from the AS to the client includes material encrypted with a key based on the client's password. An opponent can capture this message and attempt to decrypt it by trying various passwords. If the result of a test decryption is of the proper form, then the opponent has discovered the client's password and may subsequently use it to gain authentication credentials from Kerberos. Version 5 does provide a mechanism known as pre authentication, which should make password attacks more difficult, but it does not prevent them.

ELECTRONIC MAIL SECURITY

- In virtually all distributed environments, electronic mail is the most heavily used network-based application.
- Users expect to be able to, and do, send e-mail to others who are connected directly or indirectly to the Internet, regardless of host operating system or communications suite.

Pretty Good Privacy

- ✓ PGP is an open-source, freely available software package for e-mail security.
- ✓ It provides ***authentication*** through the use of ***digital signature***, ***confidentiality*** through the use of ***symmetric block encryption***, ***compression*** using the ***ZIP algorithm***, and ***e-mail compatibility*** using the ***radix-64 encoding scheme.***
- ✓ PGP was developed by **Phil Zimmermann.**
- ✓ Zimmermann has done the following:

  1. Selected the best available **cryptographic algorithms** as building blocks.
  2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.
  3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line).
  4. Entered into an agreement with a company (Via crypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

PGP has grown explosively and is now widely used because of following reasons:

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more.
2. It is based on algorithms that are considered extremely secure such as **RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.**
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization.
5. PGP is now on an Internet standards track (RFC 3156; *MIME Security withOpenPGP*).

Notations

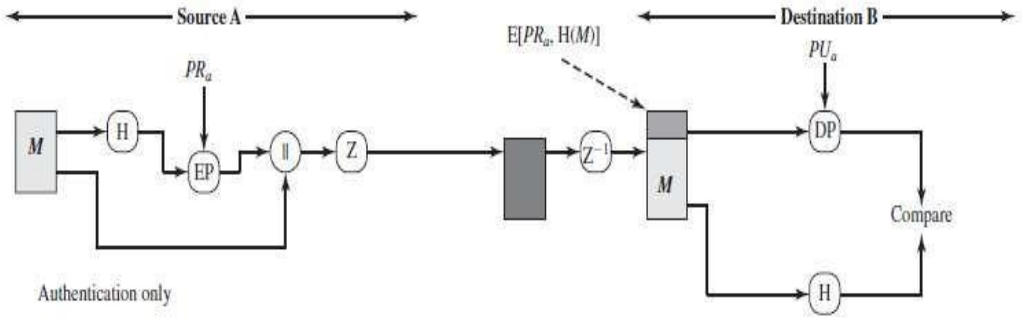
K_S : Session key used in Symmetric Encryption Scheme
PR_a : Private key of User A, used in Public Encryption Scheme
PU_a : Public key of User A, used in Public Encryption
Scheme EP : Public Encryption
DP : Public Decryption
EC : Symmetric
Encryption DC : Symmetric
Decryption H : Hash
Function
|| : Concatenation
Z : Compression
R64 : Radix 64 conversion

Operational Description

The actual operation of PGP consists of **five series:**

- **Authentication**
- **Confidentiality**
- **Compression**
- **E-mail compatibility**
- **Segmentation**

1.Authentication



Authentication

- ✓ It is provided through the digital signatures. The sequence is as follows
  1. The sender creates a message.
  2. **SHA-1** is used to generate a **160-bit hash code of the message.**
  3. The **hash code is encrypted with RSA** using the **sender's private key,** and the result is prepended to the message.
  4. The **receiver uses RSA with the sender's public key** to decrypt and recover

the hash code.

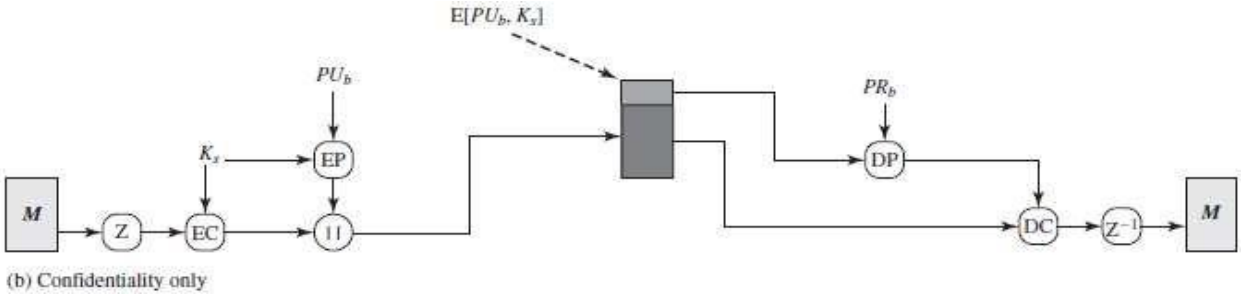
5. The receiver generates a **new hash code for the message and compares it with the decrypted hash code.** If the two match, the message is accepted as authentic.

- ✓ The combination of SHA-1 and RSA provides an effective digital signature scheme.
- ✓ Because of the strength of RSA, the recipient is assured that only the possessor of the matching private key can generate the signature.
- ✓ Because of the strength of SHA-1, the recipient is assured that no one else could generate a new message that matches the hash code and, hence, the signature of the original message
- ✓ Detached signatures are supported. A detached signature may be stored and transmitted separately from the message it signs.
- ✓ Finally, detached signatures can be used when more than one party must sign a document, such as a legal contract.

2. Confidentiality

It is provided by encrypting messages to be transmitted or to be stored locally as files. The sequence can be described as follows.

1. The sender generates a message and a random **128-bit number** to be used as a **session key** for this message only.
2. The message is encrypted using **CAST-128 (or IDEA or 3DES) with the session key**.
3. The **session key is encrypted with RSA** using the **recipient's public key** and is prepended to the message.
4. The receiver uses RSA with its private key to decrypt and recover the session key.
5. The session key is used to decrypt the message.



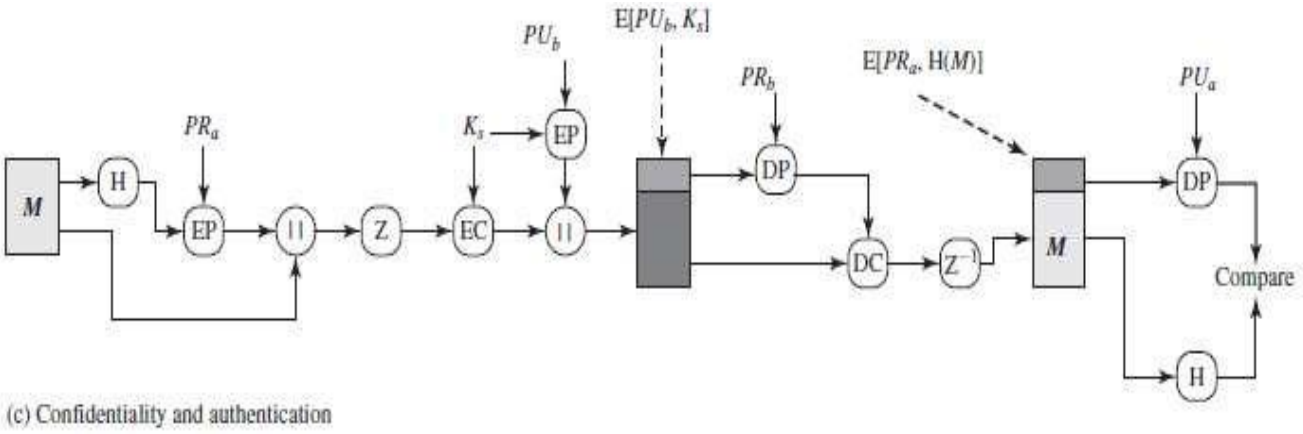
Confidentiality

- ✓ Instead of using the RSA also Diffie Hellman can be used.
- ✓ Diffie- Hellman is a key exchange algorithm.
- ✓ In fact, PGP uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal.

3. Confidentiality and Authentication

1. **Here both services(**Confidentiality and Authentication)**may be used for the same message.**
2. **First, a signature is generated for the plaintext message and prepended to the message.**

3.Then the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA (or ElGamal).



(c) Confidentiality and authentication

Confidentiality and Authentication

4. Compression

- ✓ PGP compresses the message after applying the signature but before encryption .
- ✓ The signature is generated before compression for two reasons:
- ✓ It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification.
- ✓ If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.
- ✓ However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version.
- ✓ Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm.
- ✓ Message encryption is applied after compression to strengthen cryptographic security. Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult. The compression algorithm used is ZIP

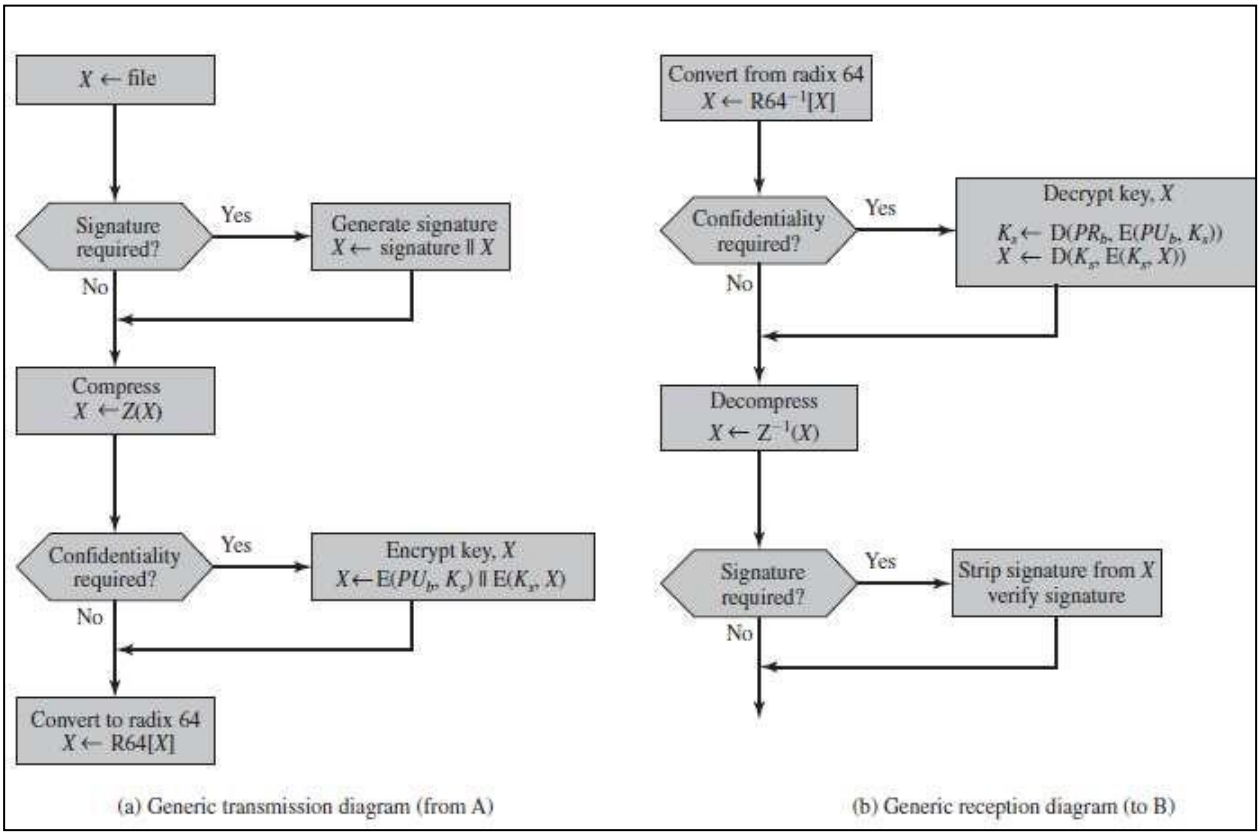
5. Email Compatibility

1. When PGP is used, at least part of the block to be transmitted is encrypted.
2. If only the **signature service is used, then the message digest(hash code) is encrypted** (with the sender's private key).
3. If the confidentiality service is used, the **message plus signature (if present) are encrypted (with a one-time symmetric key).** Thus, part or the entire resulting block consists of a stream of arbitrary **8-bit octets**.
4. PGP provides the service of **converting the raw 8-bit binary stream to a stream of printable ASCII characters.**
5. The scheme used for this purpose is **radix-64 conversion**.
6. Each group of **three octets of binary data is mapped into four ASCII characters**.
7. The **use of radix 64 expands a message by 33%.**
8. Each group of three octets of binary data is mapped into four ASCII characters.

6. Segmentation and Reassembly

1. Email facilities are often restricted to a **maximum message length.**
2. To accommodate this restriction, the PGP subdivides the message that is too large into segments that are small enough to send via email.
3. Thus the session key component, and signature component appear only once at the beginning of the first segment.
4. At the receiving end, PGP strips off all email headers and reassemble the entire original block of data.

The below Figure shows the relationship among **the four services**



Transmission and Reception of PGP Message

Transmission

- ✓ **On transmission** (if it is required), a signature is generated using a hash code of the uncompressed plaintext.
- ✓ Then the plaintext (plus signature if present) is compressed.
- ✓ Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-key encrypted symmetric encryption key.
- ✓ Finally, the entire block is converted to radix-64 format.

Reception

- ✓ On reception, the incoming block is first converted back from radix-64 format to binary.
- ✓ Then, if the message is encrypted, the recipient recovers the session key and decrypts the message.
- ✓ The resulting block is then decompressed.
- ✓ If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.

S/MIME (Security/Multipurpose Internet Mail Extension)

S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages.

- ✓ S/MIME (Secure/Multipurpose Internet Mail Extensions) is a security enhancement to the MIME internet e-mail format standard, based on RSA data security.
- ✓ S/MIME provides the following cryptographic security services for electronic messaging applications:
  - ➢ Authentication
  - ➢ Message integrity
  - ➢ Non-repudiation of origin using digital signatures
  - ➢ Data confidentiality using encryption

RFC 822

- ✓ RFC 822 defines a format for text messages that are sent using electronic mail. It has been the standard for Internet-based text mail messages and remains in common use.
- ✓ In the RFC 822 context, messages are viewed as having an envelope and contents.
- ✓ The envelope contains whatever information is needed to accomplish transmission and delivery.
- ✓ The overall structure of a message that conforms to RFC 822 is very simple.
- ✓ A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line
- ✓ A header line consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines.
- ✓ The most frequently used keywords are From, To, Subject, and Date. Here is an example message.

Example:

Date: October 8, 2009 2:15:49 PM EDT
From: William Stallings <ws@shore.net>
Subject: The Syntax in RFC822
To: Smith@Other-host.com
Cc: Jones@Yet-Another-Host.com

Hello. This section begins the **actual message body, which is delimited from the message heading by a blank line.**

Multipurpose Internet Mail Extensions (MIME)

- ✓ Multipurpose Internet Mail Extension (MIME) is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use

of Simple Mail Transfer Protocol (SMTP).

- ✓ They are

  1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/Uudecode scheme. However, none of these is a standard or even a de facto standard.
  2. SMTP cannot transmit text data that includes national language characters, because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
  3. SMTP servers may reject mail message over a certain size.
  4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
  5. SMTP gateways to X.400 electronic mail networks cannot handle non textual data included in X.400 messages.
  6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821

- ✓ Common problems include:
  - Deletion, addition, or reordering of carriage return and linefeed
  - Truncating or wrapping lines longer than 76 characters
  - Removal of trailing white space (tab and space characters)
  - Padding of lines in a message to the same length
  - Conversion of tab characters into multiple space characters.

OVERVIEW

The MIME specification includes the following elements

1. Five new message header fields are defined, which may be included in an RFC 822 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

(Explain the header fields of MIME protocol…)

The five header fields defined in MIME are

- **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
- **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.

- **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
- **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.
- **Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

(Explain **MIME** Content types ?)

MIME CONTENT TYPES:

- ✓ The bulk of the MIME specification is concerned with the definition of a variety of content types.
- ✓ There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.
- ✓ For the text type of body, no special software is required to get the full meaning of the text aside from support of the indicated character set.
- ✓ The primary subtype is plain text, which is simply a string of ASCII characters or ISO 8859 characters. The enriched subtype allows greater formatting flexibility.
- ✓ The multipart type indicates that the body contains multiple, independent parts.
- ✓ The Content-Type header field includes a parameter (called a boundary) that defines the delimiter between body parts.
- ✓ Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens.
- ✓ Within each part, there may be an optional ordinary MIME header.

Example:

From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Sample message
MIME-Version: 1.0
Content-type: multipart/mixed; boundary="simple boundary"
This is the preamble. It is to be ignored, though it is a handy place for mail composers to include an explanatory note to non-MIME conformant readers.
—simple boundary
This is implicitly typed plain ASCII text. It does NOT end with a line break.
—simple boundary
Content-type: text/plain; charset=us-ascii
This is explicitly typed plain ASCII text. It DOES end with a linebreak.

—simple boundary—

Table 7.3 MIME Content Types

Type	Subtype	Description
Text	Plain	Unformatted text; may be ASCII or ISO 8859.
	Enriched	Provides greater format flexibility.
Multipart	Mixed	The different parts are independent but are to be transmitted together. They should be presented to the receiver in the order that they appear in the mail message.
	Parallel	Differs from Mixed only in that no order is defined for delivering the parts to the receiver.
	Alternative	The different parts are alternative versions of the same information. They are ordered in increasing faithfulness to the original, and the recipient's mail system should display the "best" version to the user.
	Digest	Similar to Mixed, but the default type/subtype of each part is message/rfc822.
Message	rfc822	The body is itself an encapsulated message that conforms to RFC 822.
	Partial	Used to allow fragmentation of large mail items, in a way that is transparent to the recipient.
	External-body	Contains a pointer to an object that exists elsewhere.
Image	jpeg	The image is in JPEG format, JFIF encoding.
	gif	The image is in GIF format.
Video	mpeg	MPEG format.
Audio	Basic	Single-channel 8-bit ISDN mu-law encoding at a sample rate of 8 kHz.
Application	PostScript	Adobe Postscript format.
	octet-stream	General binary data consisting of 8-bit bytes.

- ✓ There are **four subtypes of the multipart type**, all of which have the same overall syntax.

1. The **multipart/mixed subtype** is used when there are multiple independent body parts that need to be bundled in a particular order.
2. **The multipart/parallel subtype**, the order of the parts is not significant. If the recipient's system is appropriate, the multiple parts can be presented in parallel.
3. For the **multipart/alternative subtype**, the various parts are different representations of the same information.

4. The **multipart/digest subtype** is used when each of the body parts is interpreted as an RFC 822 message with headers. This subtype enables the construction of a message whose parts are individual messages.

Example of multipart/alternative subtype:

From: Nathaniel Borenstein <nsb@bellcore.com>
To: Ned Freed <ned@innosoft.com>
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/alternative;
boundary=boundary42
—boundary42
Content-Type: text/plain; charset=us-ascii
...plain text version of message goes here....
—boundary42
Content-Type: text/enriched
.... RFC 1896 text/enriched version of same message
goes here ...
—boundary42—

There are three sub types in **message type** in MIME.

1. The **message/rfc822 subtype:** indicates that the body is an entire message, including header and body.
2. **The message/partial subtype** enables fragmentation of a large message into a number of parts, which must be reassembled at the destination. For this subtype, three parameters are specified in the Content-Type: Message/Partial field: an id common to all fragments of the same message, a sequence number unique to each fragment, and the total number of fragments.
3. **The message/external-body subtype**: indicates that the actual data to be conveyed in this message are not contained in the body. Instead, the body contains the information needed to access the data

- ✓ **The application type** refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application. The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters.

(Explain MIME Transfer Encodings ?)

MIME TRANSFER ENCODINGS

- ✓ The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies.

- ✓ The Content-Transfer-Encoding field can actually take on **six values**, as listed in below Table However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data.
- ✓ For SMTP transfer, it is safe to use the 7bit form.
- ✓ The 8bit and binary forms may be usable in other mail transport contexts.
- ✓ x-token Encoding , which indicates that some other encoding scheme is used for which a name is to be supplied. This could be a vendor-specific or application-specific scheme.

- ✓ The **two actual encoding schemes defined are quoted-printable and base64.**
- ✓ Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.
- ✓ The **quoted-printable** transfer encoding is useful when the data consists largely of **octets** that correspond to **printable ASCII characters.**
- ✓ In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.
- ✓ The **base64 transfer encoding,** also known as **radix-64 encoding,** is a common one for **encoding arbitrary binary data** in such a way as to be invulnerable to the processing by mail-transport programs.

Table 7.4 MIME Transfer Encodings

7bit	The data are all represented by short lines of ASCII characters.
8bit	The lines are short, but there may be non-ASCII characters (octets with the high-order bit set).
binary	Not only may non-ASCII characters be present, but the lines are not necessarily short enough for SMTP transport.
quoted-printable	Encodes the data in such a way that if the data being encoded are mostly ASCII text, the encoded form of the data remains largely recognizable by humans.
base64	Encodes data by mapping 6-bit blocks of input to 8-bit blocks of output, all of which are printable ASCII characters.
x-token	A named nonstandard encoding.

S/MIME Functionality:

(Describe S/MIME functionality.)

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability.

FUNCTIONS S/MIME provides the following functions.

- **Enveloped data**: This consists of encrypted content of any type and encrypted content encryption keys for one or more recipients.
- **Signed data:** A digital signature is formed by taking the message digest of the content

to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.

- **Clear-signed data**: As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64.As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
- **Signed and enveloped data**: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

CRYPTOGRAPHIC ALGORITHM

- S/MIME uses the following terminology to specify requirement level:

MUST: The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.

SHOULD: There may exist valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

- S/MIME incorporates three public-key algorithms.
  - The Digital Signature Standard (DSS) is the preferred algorithm for digital signature.
  - S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys
  - RSA, can be used for both signatures and session key encryption.

Table 7.6 Cryptographic Algorithms Used in S/MIME

Function	Requirement
Create a message digest to be used in forming a digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility.
Encrypt message digest to form a digital signature.	Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with a message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with a one-time session key.	Sending and receiving agents MUST support encryption with tripleDES. Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code.	Receiving agents MUST support HMAC with SHA-1. Sending agents SHOULD support HMAC with SHA-1.

- For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME.
- For message encryption, three-key triple DES (tripleDES) is recommended, but compliant implementations must support 40-bit RC2.
- The following rules, in the following order, should be followed by a sending agent.

  1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.

  2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.

  3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use triple DES.

  4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

S/MIME Messages

(Explain S/MIME content types)

- ✓ S/MIME makes use of a number of new MIME content types shown in table.
- ✓ All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories.

Table 7.7 S/MIME Content Types

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity.
	pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

Securing a MIME entity:

- ✓ **S/MIME** secures a MIME entity with a signature encryption, or both. A MIME entity may be an entire message , or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message.
- ✓ Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object.
- ✓ A PKCS object is then treated as message content and wrapped in MIME. In all cases, the message to be sent is converted to canonical form.

Enveloped Data

- ✓ An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique s/mime-type parameter.
- ✓ The steps for preparing an enveloped Data MIME entity are:

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or triple DES).
2. For each recipient, encrypt the session key with the recipient's public RSA key.
3. For each recipient, prepare a block known as **RecipientInfo** that contains an identifier of the recipient's public-key certificate, an identifier of the algorithm used to encrypt the session key, and the encrypted session key.
4. Encrypt the message content with the session key.

- ✓ The **RecipientInfo** blocks followed by the encrypted content constitute the enveloped Data.

Signeddata

- ✓ The signedData s/mime-type can be used with one or more signers. For clarity, we

confine our description to the case of a single digital signature.

The steps for preparing a signedData MIME entity are:

1. Select a message digest algorithm (SHA or MD5).
2. Compute the message digest (hash function) of the content to be signed.
3. Encrypt the message digest with the signer's private key.
4. Prepare a block known as SignerInfo that contains the signer's public key certificate, an identifier of the message digest algorithm, an identifier of the algorithm used to encrypt the message digest, and the encrypted message digest.

- ✓ The signedData entity consists of a series of blocks, including a message digest algorithm identifier, the message being signed, and SignerInfo.

Clear Signing

- ✓ Clear signing is achieved using the multipart content type with a signed subtype.
- ✓ A multipart/signed message has two parts. The first part can be any MIME type but must be prepared so that it will not be altered during transfer from source to destination.
- ✓ This second part has a MIME content type of application and a subtype of pkcs7-signature.

Registration Request

An application or user will apply to a certification authority for a public-key certificate. The application/pkcs10 S/MIME entity is used to transfer a certification request.

CERTIFICATES-ONLY MESSAGE

A message containing only certificates or a certificate revocation list (CRL) can be sent in response to a registration request. The message is an application/pkcs7-mime type/subtype with an s/mime-type parameter of degenerate.

S/MIME Certificate Processing

S/MIME uses public-key certificates that confirm to version 3 of X.509.

USER AGENT ROLE:

An S/MIME user has several key-management functions to perform.

- **Key generation**: The user of some related administrative utility MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of a non deterministic random input. A user agent SHOULD generate RSA key pairs with a length in the range of 768 – 1024 bits and MUST NOT generate less than 512 bits.
- **Registration**: A user's public key must be registered with a certification authority

in order to receive an X.509 public-key certificate.

- **Certificate storage and retrieval**: A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages.

VeriSign Certificates

- ✓ There are several companies that provide certification authority (CA) services. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications.
- ✓ VeriSign issues X.509 certificates with the product name VeriSign Digital ID.
- ✓ The information contained in a Digital ID depends on the type of Digital ID and its use.
- ✓ At a minimum, each Digital ID contains:
  - Owner's public key
  - Owner's name or alias
  - Expiration date of the Digital ID
  - Serial number of the Digital ID
  - Name of the certification authority that issued the Digital ID
  - Digital signature of the certification authority that issued the Digital ID.
- ✓ Digital IDs can also contain user supplied information :
  - Address
  - Email address
  - Basic registration information (Country ZIP code, age and gender)

- ✓ VeriSign provides three levels, or classes, of security for public-key certificates.
  - For Class 1 Digital IDs, VeriSign confirms the user's e-mail address by sending a PIN and Digital ID pick-up information to the e-mail address provided in the application.
  - For Class 2 Digital IDs, VeriSign verifies the information in the application through an automated comparison with a consumer database in addition to performing all of the checking associated with a Class 1 Digital ID.
  - For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance.

Enhanced Security Services:

The three services are:

- **Signed receipts**: A signed receipt may be requested in a SignedData object. e. In essence, the recipient signs the entire original message plus the original (sender's) signature and appends the new signature to form a new S/MIME messages.
- **Security labels:** A security label may be included in the authenticated attributes of a SignedData object. The labels may be used for access control, by indicating which users are permitted access to an object.

- **Secure mailing lists**: When a user sends a message to multiple recipients, a certain amount of per-recipient processing is required, including the use of each recipient's public key. The user can be relieved of this work by employing the services of an S/MIME Mail List Agent (MLA). An MLA can take a single incoming message, perform the recipient-specific encryption for each recipient, and forward the message. The originator of a message need only send the message to the MLA with encryption performed using the MLA's public key.

DomainKeys Identified Mail (DKIM)

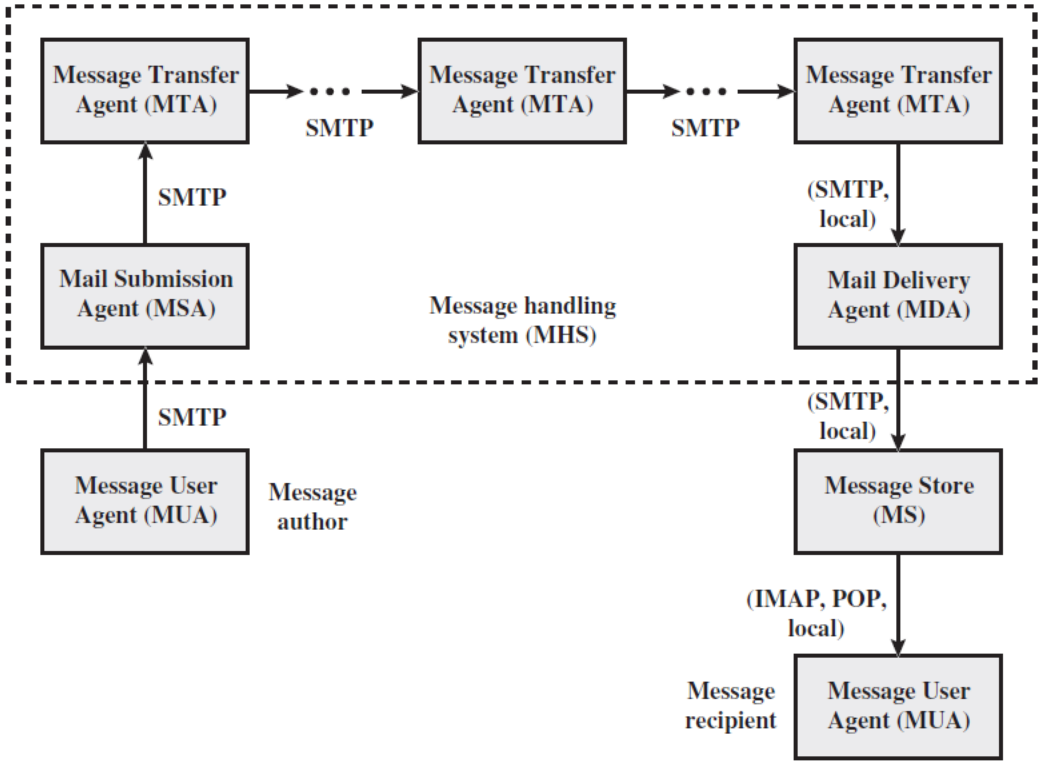
- DomainKeys Identified Mail (DKIM) is a specification for cryptographically signing e-mail messages, permitting a signing domain to claim responsibility for a message in the mail stream.
- Message recipients (or agents acting in their behalf) can verify the signature by querying the signer's domain directly to retrieve the appropriate public key and thereby can confirm that the message was attested to by a party in possession of the private key for the signing domain.
- DKIM has been widely adopted by a range of e-mail providers, including corporations, government agencies, gmail, yahoo, and many Internet Service Providers (ISPs).

Internet Mail Architecture

- **Message User Agent (MUA):** Operates on behalf of user actors and user applications. It is their representative within the e-mail service. Typically, this function is housed in the user's computer and is referred to as a client e-mail program or a local network e-mail server. The author MUA formats a message and performs initial submission into the MHS via a MSA. The recipient MUA processes received mail for storage and/or display to the recipient user.
- **Mail Submission Agent (MSA):** Accepts the message submitted by an MUA and enforces the policies of the hosting domain and the requirements of Internet standards. This function may be located together with the MUA or as a separate functional model. In the latter case, the Simple Mail Transfer Protocol (SMTP) is used between the MUA and the MSA.
- **Message Transfer Agent (MTA):** Relays mail for one application-level hop. It is like a packet switch or IP router in that its job is to make routing assessments and to move the message closer to the recipients. Relaying is performed by a sequence of MTAs until the

message reaches a destination MDA. An MTA also adds trace information to the message header. SMTP is used between MTAs and between an MTA and an MSA or MDA.

- **Mail Delivery Agent (MDA):** Responsible for transferring the message from the MHS to the MS.
- **Message Store (MS):** An MUA can employ a long-term MS. An MS can be located on a remote server or on the same machine as the MUA. Typically, an MUA retrieves messages from a remote server using POP (Post Office Protocol) or IMAP (Internet Message Access Protocol).



E-mail Threats

Characteristics

1. At the low end are attackers who simply want to send e-mail that a recipient does not want to receive. The attacker can use one of a number of commercially available tools that allow the sender to falsify the origin address of messages. This makes it difficult for the receiver to filter spam on the basis of originating address or domain.
2. At the next level are professional senders of bulk spam mail. These attackers often operate as commercial enterprises and send messages on behalf of third parties. They

employ more comprehensive tools for attack, including Mail Transfer Agents (MTAs) and registered domains and networks of compromised computers (zombies) to send messages and (in some cases) to harvest addresses to which to send.

3. The most sophisticated and financially motivated senders of messages are those who stand to receive substantial financial benefit, such as from an e-mail-based fraud scheme. These attackers can be expected to employ all of the above mechanisms and additionally may attack the Internet infrastructure itself, including DNS cache-poisoning attacks and IP routing attacks.

Capabilities

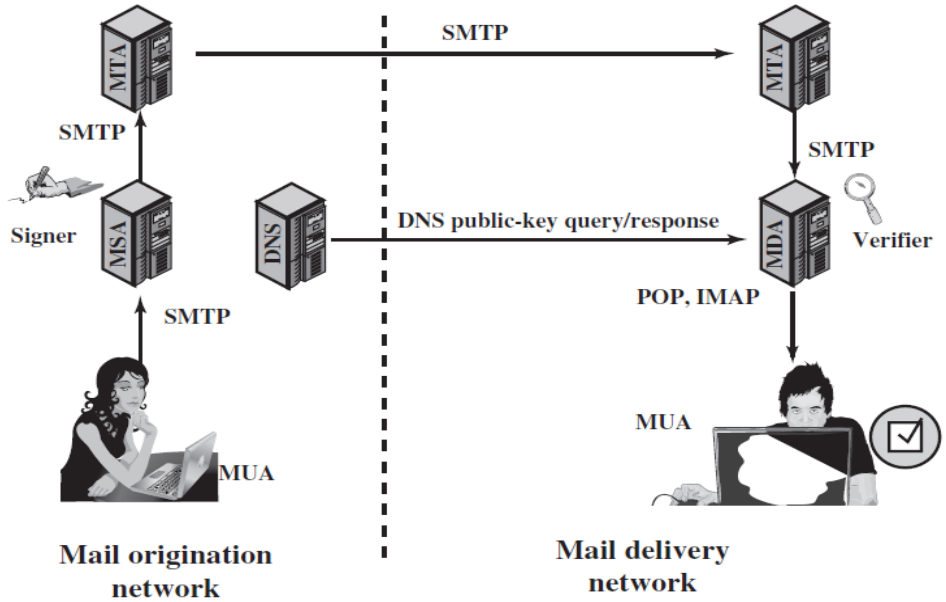
1. Submit messages to MTAs and Message Submission Agents (MSAs) at multiple locations in the Internet.
2. Construct arbitrary Message Header fields, including those claiming to be mailing lists, resenders, and other mail agents.
3. Sign messages on behalf of domains under their control.
4. Generate substantial numbers of either unsigned or apparently signed messages that might be used to attempt a denial-of-service attack.
5. Resend messages that may have been previously signed by the domain.
6. Transmit messages using any envelope information desired.
7. Act as an authorized submitter for messages from a compromised computer.
8. Manipulation of IP routing. This could be used to submit messages from specific
9. IP addresses or difficult-to-trace addresses, or to cause diversion of messages to a specific domain.
10. Limited influence over portions of DNS using mechanisms such as cache poisoning.
11. This might be used to influence message routing or to falsify advertisements of DNS-based keys or signing practices.
12. Access to significant computing resources, for example, through the conscription of worm-infected “zombie” computers. This could allow the “bad actor” to perform various types of brute-force attacks.
13. Ability to eavesdrop on existing traffic, perhaps from a wireless network.

Location

DKIM focuses primarily on attackers located outside of the administrative units of the claimed originator and the recipient. These administrative units frequently correspond to the protected portions of the network adjacent to the originator and recipient.

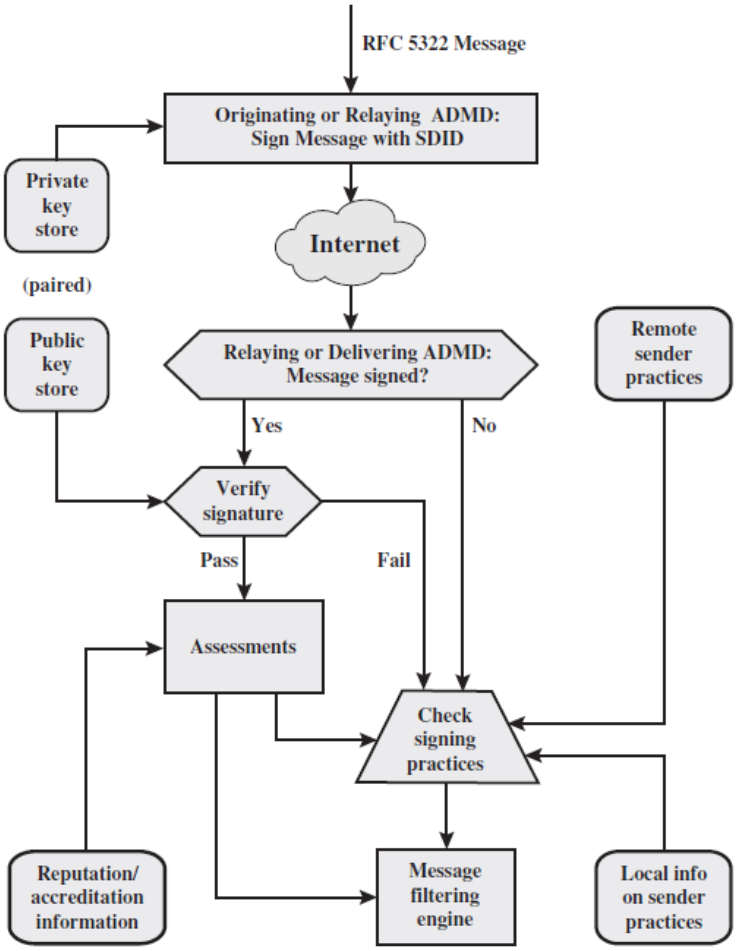
DKIM Strategy

1. S/MIME depends on both the sending and receiving users employing S/MIME. For almost all users, the bulk of incoming mail does not use S/MIME, and the bulk of the mail the user wants to send is to recipients not using S/MIME.
2. S/MIME signs only the message content. Thus, RFC 5322 header information concerning origin can be compromised.
3. DKIM is not implemented in client programs (MUAs) and is therefore transparent to the user; the user need take no action.
4. DKIM applies to all mail from cooperating domains.
5. DKIM allows good senders to prove that they did send a particular message and to prevent forgers from masquerading as good senders.



DNS = domain name system
MDA = mail delivery agent
MSA = mail submission agent
MTA = message transfer agent
MUA = message user agent

DKIM Functional Flow



Basic message processing is divided between a signing Administrative Management Domain (ADMD) and a verifying ADMD. At its simplest, this is between the originating ADMD and the delivering ADMD, but it can involve other ADMDs in the handling path.

Signing is performed by an authorized module within the signing ADMD and uses private information from a Key Store. Within the originating ADMD, this might be performed by the MUA, MSA, or an MTA. Verifying is performed by an authorized module within the verifying ADMD. Within a delivering ADMD, verifying might be performed by an MTA, MDA, or MUA. The module verifies the signature or determines whether a particular signature was required. Verifying the signature uses public information from the Key Store. If the signature passes, reputation information is used to assess the signer and that information is passed to the message filtering system. If the signature fails or there is no signature using the author's domain,

information about signing practices related to the author can be retrieved remotely and/or locally, and that information is passed to the message filtering system. For example, if the sender (e.g., Gmail) uses DKIM but no DKIM signature is present, then the message may be considered fraudulent.

Module 5
IP SECURITY

IP-level security encompasses three functional areas:

- Authentication
- Confidentiality
- Key management.

5.1 IP SECURITY OVERVIEW

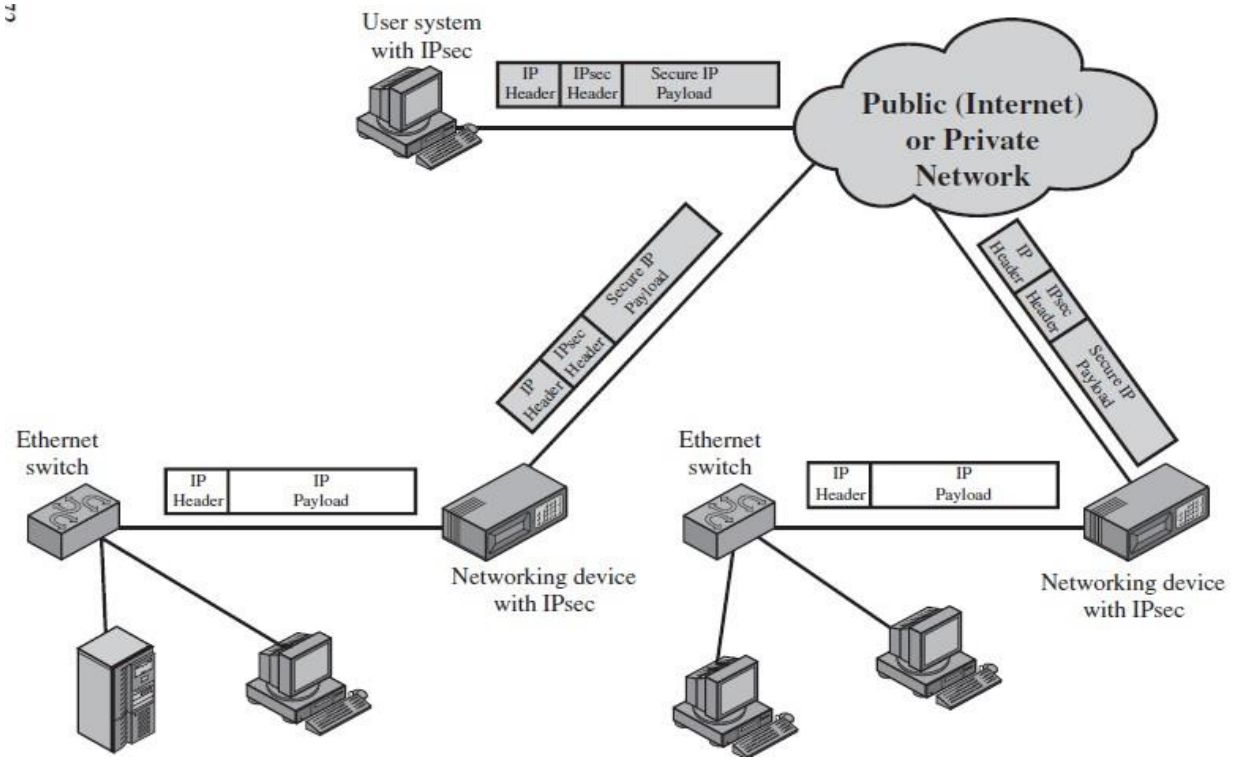
IPsec provides the capability to secure communications across a LAN, across private and public WANs, and across the Internet. Examples of its use include the following:

- ***Secure branch office connectivity over the Internet:*** A company can build a secure virtual private network over the Internet or over a public WAN. This enables a business to rely heavily on the Internet and reduce its need for private networks, saving costs and network management overhead.
- ***Secure remote access over the Internet*****:** An end user whose system is equipped with IP security protocols can make a local call to an Internet Service Provider (ISP) and gain secure access to a company network. This reduces the cost of toll charges for traveling employees and telecommuters.
- ***Establishing extranet and intranet connectivity with partners:*** IPsec can be used to secure communication with other organizations, ensuring authentication and confidentiality and providing a key exchange mechanism.
- ***Enhancing electronic commerce security:*** Even though some Web and electronic commerce applications have built-in security protocols, the use of IPsec enhances that security. IPsec guarantees that all traffic designated by the network administrator is both encrypted and authenticated, adding an additional layer of security to whatever is provided at the application layer.

The below diagram shows typical scenario of IPsec usage.

- An organization maintains LANs at dispersed locations. Nonsecure IP traffic is conducted on each LAN.
- For traffic offsite, through some sort of private or public WAN, IPsec protocols are used.
- These protocols operate in networking devices, such as a router or firewall that connect each LAN to the outside world.
- The IPsec networking device will typically encrypt and compress all traffic going into the WAN and decrypt and decompress traffic coming from the WAN; these operations are transparent to workstations and servers on the LAN

- Secure transmission is also possible with individual users who dial into the WAN. Such user workstations must implement the IPsec protocols to provide security.



An IP Security Scenario

Benefits of IPsec

Some of the benefits of IPsec are:

- When IPsec is implemented in a firewall or router, it provides strong security that can be applied to all traffic crossing the perimeter.
- IPsec in a firewall is resistant to bypass if all traffic from the outside must use IP and the firewall is the only means of entrance from the Internet into the organization.
- IPsec is below the transport layer (TCP, UDP) and so is transparent to applications.
- IPsec can be transparent to end users. There is no need to train users on security mechanisms, issue keying material on a per-user basis, or revoke keying material when users leave the organization.
- IPsec can provide security for individual users if needed.

Routing Applications

IPsec can assure that

- A router advertisement comes from an authorized router.
- A neighbor advertisement comes from an authorized router.
- A redirect message comes from the router to which the initial IP packet was sent.
- A routing update is not forged.

5.2 IP SECURITY ARCHITECTURE

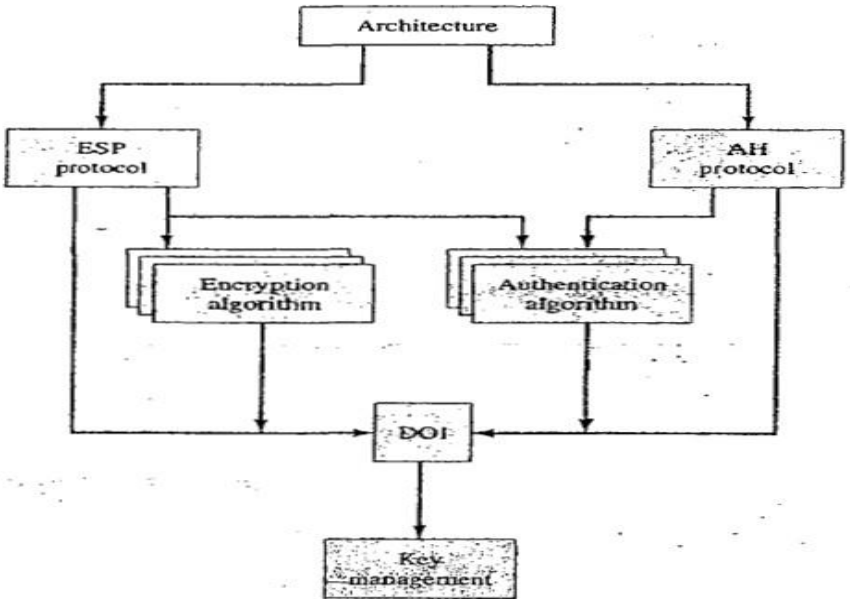
IPsec Documents :

- ✓ IPsec encompasses three functional areas: authentication, confidentiality, and key management. The best way to grasp the scope of IPsec is to consult the latest version of the IPsec document roadmap, which as of this writing is [FRAN09].

The documents can be categorized into the following groups.

- **Architecture:**
  Covers the general concepts, security requirements, definitions, and mechanisms defining IPsec technology.
- **Encapsulating Security Payload (ESP):**
  ESP consists of an encapsulating header and trailer used to provide encryption or combined encryption/authentication.
- **Authentication Header (AH):**
  AH is an extension header to provide message authentication. **Encryption Algorithm:** A set of documents that describe how various encryption algorithms are used for ESP
- **Authentication Algorithm:**
  A set of documents that describe how various authentication algorithms are used for AH and for the authentication option of ESP
- **Key Management:** document that describes key management schemes.
- **Domain of Interpretation(DOI):** contains values needed for the other documents to relate to each other.

The below diagram shows this concept



IPSec Document Overview

IPsec Services

- ✓ IPsec provides security services at the IP layer by enabling a system to select required security protocols, determine the algorithm(s) to use for the service(s), and put in place any cryptographic keys required to provide the requested services.
- ✓ Two protocols are used to provide security:
  - ➢ An authentication protocol designated by the header of the protocol, Authentication Header (AH)
  - ➢ A combined encryption/ authentication protocol designated by the format of the packet for that protocol, Encapsulating Security Payload (ESP).
- ✓ The services are:
  - ➢ Access control
  - ➢ Connectionless integrity
  - ➢ Data origin authentication
  - ➢ Rejection of replayed packets (a form of partial sequence integrity)
  - ➢ Confidentiality (encryption)
  - ➢ Limited traffic flow confidentiality

Security Associations

- ✓ An association is a one-way logical connection between a sender and a receiver that affords security services to the traffic carried on it.
- ✓ Security association (SA) is the key concept for both authentication and confidentiality mechanism for IP.
- ✓ A security association is uniquely identified by three parameters.
  - ➢ **Security Parameters Index (SPI):** A bit string assigned to this SA and having local significance only. The SPI is carried in AH and ESP headers to enable the receiving system to select the SA under which a received packet will be processed.
  - ➢ **IP Destination Address:** This is the address of the destination endpoint of the SA, which may be an end-user system or a network system such as a firewall or router.
  - ➢ **Security Protocol Identifier:** This field from the outer IP header indicates whether the association is an AH or ESP security association.

SA Parameter(Security Association Database)

- ✓ A security association is defines the parameter associated with SA.
- ✓ A security association is normally defined by the following parameters in an SAD entry.
  - ➢ **Security Parameter Index:** A 32-bit value selected by the receiving end of an SA to uniquely identify the SA.
  - ➢ **Sequence Number Counter:** A 32-bit value used to generate the Sequence Number field in AH or ESP headers.

- **Sequence Counter Overflow:** A flag indicating whether overflow of the Sequence Number Counter should generate an auditable event and prevent further transmission of packets on this SA
- **Anti-Replay Window:** Used to determine whether an inbound AH or ESP packet is a replay.
- **AH Information:** Authentication algorithm, keys, key lifetimes, and related parameters being used with AH
- **ESP Information:** Encryption and authentication algorithm, keys, initialization values, key lifetimes, and related parameters being used with ESP
- **Lifetime of this Security Association:** A time interval or byte count after which an SA must be replaced with a new SA or terminated, plus an indication of which of these actions should occur.
- **IPsec Protocol Mode:** Tunnel, transport, or wildcard.
- **Path MTU:** Any observed path maximum transmission unit and aging variables

SA Selectors (Security Policy Database)

- ✓ IPSec provides the user with considerable flexibility in the way in which IPSec services are applied.
- ✓ IP traffic is related to specific SAs (or no SA in the case of traffic allowed to bypass IPsec) is the nominal Security Policy Database (SPD).
- ✓ In its simplest form, an SPD contains entries, each of which defines a subset of IP traffic and points to an SA for that traffic.
- ✓ In more complex environments, there may be multiple entries that potentially relate to a single SA or multiple SAs associated with a single SPD entry.
  - ✓ Each SPD entry is defined by a set of IP and upper-layer protocol field values, called *selectors*.
- ✓ These selectors are used to filter outgoing traffic in order to map it into a particular SA.

Outbound processing obeys the following general sequence for each IP packet:

1. Compare the values of the appropriate fields in the packet (the selector fields) against the SPD to find a matching SPD entry, which will point to zero or more SAs.
2. Determine the SA if any for this packet and its associated SPI.
3. Do the required IPsec processing (i.e., AH or ESP processing).

The following selectors determine an SPD entry:

- **Remote IP Address (destination IP address):** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address.
- **Source IP Address:** This may be a single IP address, an enumerated list or range of addresses, or a wildcard (mask) address.
- **UserID:** a user identifier from the operating system

- **Data sensitivity level:** used for systems providing information flow security
- **Transport layer protocol:** this may be an individual protocol number, a list of protocol numbers, or a range of protocol numbers
- **Source and Destination ports:** these may be individual TCP or UDP port values, an enumerated list of ports or a wildcard port

Transport and Tunnel Modes

- ✓ Both AH and ESP support two modes of use
  - Transport mode
  - Tunnel mode
- ✓ The below table shows the functionality of transport mode and tunnel mode

Tunnel Mode and Transport Mode Functionality

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

Transport Mode:

- ✓ Transport mode provides protection primarily for upper-layer protocols.
- ✓ Transport mode protection extends to the payload of an IP packet.

Example: TCP or UDP segment or an ICMP packet

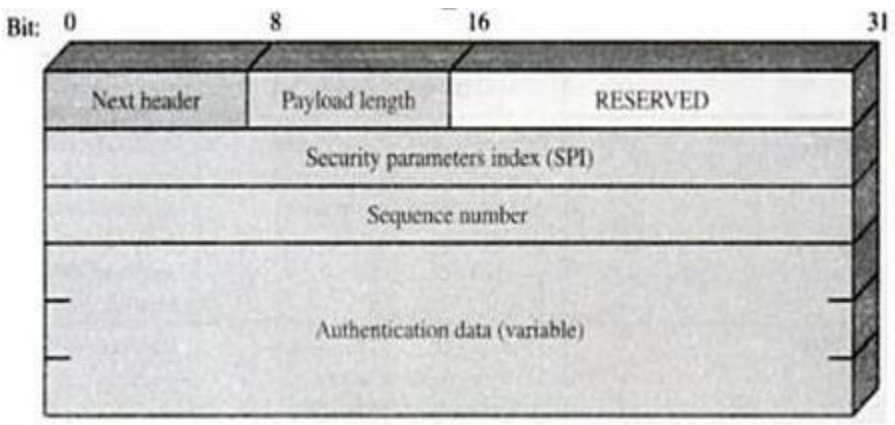
- ✓ Transport mode is used for end-to-end communication between two hosts (e.g., a client and a server, or two workstations).
- ✓ When a host runs AH or ESP over IPv4, the payload is the data that normally follow the IP header.
- ✓ For IPv6, the payload is the data that normally follow both the IP header and any IPv6 extensions headers that are present, with the possible exception of the destination options header, which may be included in the protection.
- ✓ ESP in transport mode encrypts and optionally authenticates the IP payload but not the IP header.
- ✓ AH in transport mode authenticates the IP payload and selected portions of the IP header.

Tunnel Mode:

- ✓ Tunnel mode provides protection to the entire IP packet. To achieve this, after the AH or ESP fields are added to the IP packet, the entire packet plus security fields is treated as the payload of new outer IP packet with a new outer IP header.
- ✓ The entire original, inner, packet travels through a tunnel from one point of an IP network to another; no routers along the way are able to examine the inner IP header.
- ✓ Because the original packet is encapsulated, the new, larger packet may have totally different source and destination addresses, adding to the security.
- ✓ Tunnel mode is used when one or both ends of a security association (SA) are a security gateway, such as a firewall or router that implements IPsec.

5.3 AUTHENTICATION HEADER

- ✓ The Authentication Header provides support for data integrity and authentication of IP packets
- ✓ The data integrity feature ensures that undetected modification to a packet's content in transit is not possible
- ✓ The authentication feature enables an end system or network device to authenticate the user or application and filter traffic accordingly
- ✓ It also prevents the address spoofing attacks and guards against the replay attack.
- ✓ The below diagram shows IPSec authentication header



IPSec Authentication Header

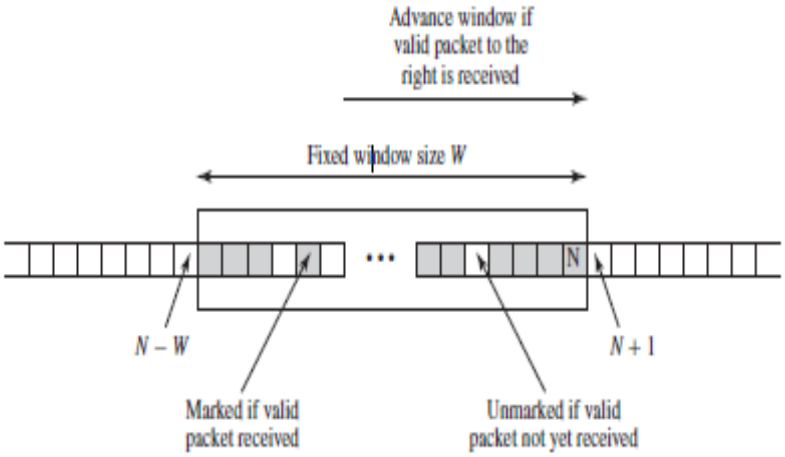
The authentication header consists of the following fields:

- ✓ **Next Header (8 bits):** Identifies the type of the header immediately following this header.
- ✓ **Payload Length(8 bits):** Length of authentication header in 32-bit words, minus 2.
- ✓ **Reserved(16 bits):** For future use.
- ✓ **Security Parameters Index(32 bits):** Identifies a security association.

- ✓ **Sequence Number(32 bits):** A monotonically increasing counter value
- ✓ **Authentication Data(variable):** A variable-length field that contains the Integrity Check value(ICV),or MAC for the packet.

Anti-Replay service

- ✓ A replay attack is one in which an attacker obtains a copy of an authenticated packet and later transmits it to the intended destination
- ✓ The sequence number field is designed to thwart such attacks
- ✓ When a new SA is established, the sender initializes a sequence number counter to 0
- ✓ Each time a packet is sent on this SA, the sender increments the counter and places the value in sequence number field .Thus, the first value to be used is 1.
- ✓ If anti-replay is enabled, the sender must not allow the sequence number to cycle past $2^{32} – 1$ back to 0 otherwise, there would be multiple valid packets with the same sequence number.
- ✓ If the limit of 232 – 1 is reached, the sender should terminate this SA and negotiate a new SA with a new key.
- ✓ Because IP is a connectionless, unreliable service, the protocol does not guarantee that packets will be delivered in order and does not guarantee that all packets will be delivered.
- ✓ The IPSec authentication document dictates that the receiver should implement a window of size W with a default of W=64.
- ✓ The right edge of the window represents the highest sequence number, , so far received for a valid packet.
- ✓ For any packet with a sequence number in the range from to that has been correctly received (i.e., properly authenticated), the corresponding slot in the window is marked .



Anti-replay Mechanism

- ✓ Inbound processing proceeds as follows when a packet is received:

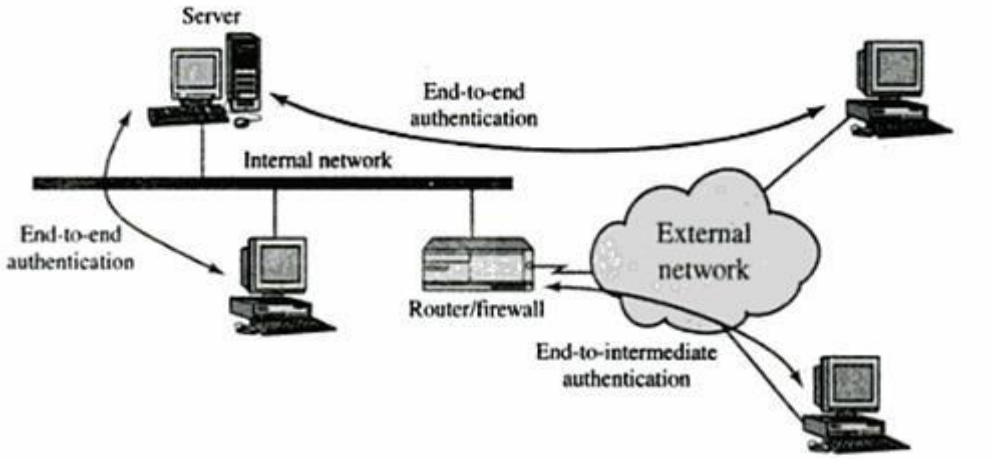
1. If the received packet falls within the window and is new, the MAC is checked. If the packet is authenticated, the corresponding slot in the window is marked.
2. If the received packet is to the right of the window and is new, the MAC is checked. If the packet is authenticated, the window is advanced so that this sequence number is the right edge of the window, and the corresponding slot in the window is marked.
3. If the received packet is to the left of the window, or if authentication fails, the packet is discarded; this is an auditable event.

Integrity Check Value

- ✓ The ICV is a message authentication code or truncated version of a code produced by a MAC algorithm.
- ✓ The MAC is calculated over
  1. IP header fields that either do not change or that are predictable in value upon arrival at the endpoint for the AH SA>
  2. The AH header other than the authentication data field.
  3. The entire upper-level protocol data, which is assumed to be immutable in transist.

Transport and Tunnel Modes

- ✓ The below diagram shows two ways in which the IPSec authentication service can be used.



End-to-End versus End-to-Intermediate Authentication

Transport Mode:

- ✓ In this case authentication is provided directly between a server and client workstations; the workstations can be either on the same network as the server or on an external network.

- ✓ As long as the workstation and the server share a protected secret key, the authentication process is secure.

Tunnel Mode:

- ✓ In this case a remote workstation authenticates itself to the corporate firewall, either for access to the entire internal network or because the requested server does not support the authentication feature.
- ✓
- ✓ For **transport mode** AH using IPv4, the AH is inserted after the original IP header and before the IP payload
- ✓ Authentication covers the entire packet, excluding mutable fields in the IPv4 header that are set to zero for MAC calculation
- ✓ For **tunnel mode** AH, the entire original IP packet is authenticated and the AH is inserted between the original IP header and a new outer IP header
- ✓ The inner IP header carries the ultimate source and destination address
- ✓ The outer IP header may contain different IP addresses
- ✓ The below diagram shows Scope of AH Encryption and Authentication scheme

Original IPv4 Header (any options)	TCP	Data

IPv4----After applying AH (transport mode)

Original IPv4 Header (any options)	AH	TCP	Data

IPv4----After applying AH (tunnel mode)

New IPv4 Header (any options)	AH	Orig IPv4 Header (any options)	TCP	Data

(a)

IPv6----Before applying AH

Original IPv6 Header	Extension Header if present	TCP	Data

IPv6----After applying AH (transport mode)

Original IPv6 Header	Extension Header if present *	AH	Dest. *	TCP	Data

*: If destination extension header presents, it could be before AH, after AH or both

IPv6----After applying AH (tunnel mode)

New IPv6 Header	New Extension Headers if present	AH	Orig IPv6 Header	Extension Headers if present	TCP	Data

(b)

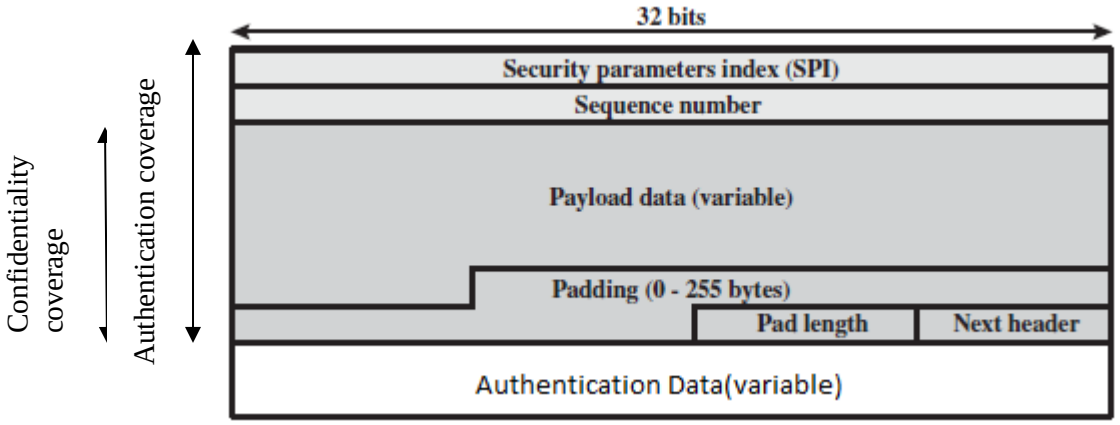
Scope of AH Authentication

5.4 ENCAPSULATING SECURITY PAYLOAD(ESP)

- ✓ ESP can be used to provide confidentiality, data origin authentication, connectionless integrity, an anti-replay service (a form of partial sequence integrity), and (limited) traffic flow confidentiality
- ✓ The set of services provided depends on options selected at the time of Security Association (SA) establishment and on the location of the implementation in a network topology.
- ✓ ESP can work with a variety of encryption and authentication algorithms, including authenticated encryption algorithms such as GCM.

ESP Format

In below Figure shows the format of an ESP packet. It contains the following fields.



- **Security Parameters Index (32 bits):** Identifies a security association.
- **Sequence Number (32 bits):** A monotonically increasing counter value; this provides an anti-replay function, as discussed for AH.
- **Payload Data (variable):** This is a transport-level segment (transport mode) or IP packet (tunnel mode) that is protected by encryption.
- **Padding (0–255 bytes):** The purpose of this field is discussed later.
- **Pad Length (8 bits):** Indicates the number of pad bytes immediately preceding this field.
- **Next Header (8 bits):** Identifies the type of data contained in the payload data field by identifying the first header in that payload (for example, an extension header in IPv6, or an upper-layer protocol such as TCP).
- **Authentication Data Variable:** A variable Length field that contains Integrity Check value computed over the ESP Packet minus the Authentication Data field.

Encryption and Authentication Algorithms

- ✓ The Payload Data, Padding, Pad Length, and Next Header fields are encrypted by the ESP service. These include:
  - Three-key triple DES
  - RC5
  - IDEA
  - Three-key triple IDEA
  - CAST
  - Blowfish

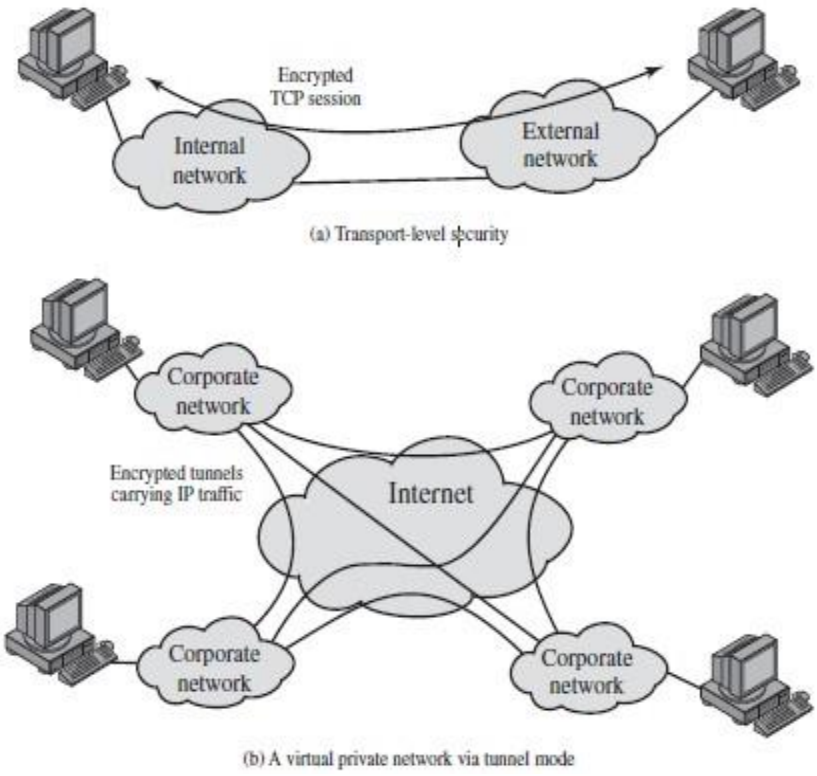
Padding

The Padding field serves several purposes:

- If an encryption algorithm requires the plaintext to be a multiple of some number of bytes, the Padding field is used to expand the plaintext to the required length.
- The ESP format requires that the Pad Length and Next Header fields be right aligned within a 32-bit word.
- Additional padding may be added to provide partial traffic-flow confidentiality by concealing the actual length of the payload.

Transport and Tunnel Modes

The below Figure shows two ways in which the IPSec ESP service can be used.



Transport-Mode versus Tunnel-Mode Encryption

- ✓ In Transport mode Encryption is provided directly between two hosts.
- ✓ In Tunnel mode operation can be used to set up a ***virtual private network***.
- ✓ In the above diagram, an organization has four private networks interconnected across the Internet. Hosts on the internal networks use the Internet for transport of data but do not interact with other Internet-based hosts. By terminating the tunnels at the security gateway to each internal network, the configuration allows the hosts to avoid implementing the security capability.

Transport Mode ESP

- ✓ Transport mode ESP is used to encrypt and optionally authenticate the data carried by IP.
- ✓ The IPv6, ESP is viewed as an end-to-end payload; that is, it is not examined or processed by intermediate routers.
- ✓ Therefore, the ESP header appears after the IPv6 base header and the hop-by-hop, routing, and fragment extension headers.

- ✓ Transport mode operation is summarized as follows.

  1. At the source, the block of data consisting of the ESP trailer plus the entire transport-layer segment is encrypted and the plaintext of this block is replaced with its ciphertext to form the IP packet for transmission. Authentication is added if this option is selected.

  2. The packet is then routed to the destination. Each intermediate router needs to examine and process the IP header plus any plaintext IP extension headers but does not need to examine the cipher text.

  3. The destination node examines and processes the IP header plus any plaintext IP extension headers.

Tunnel Mode ESP

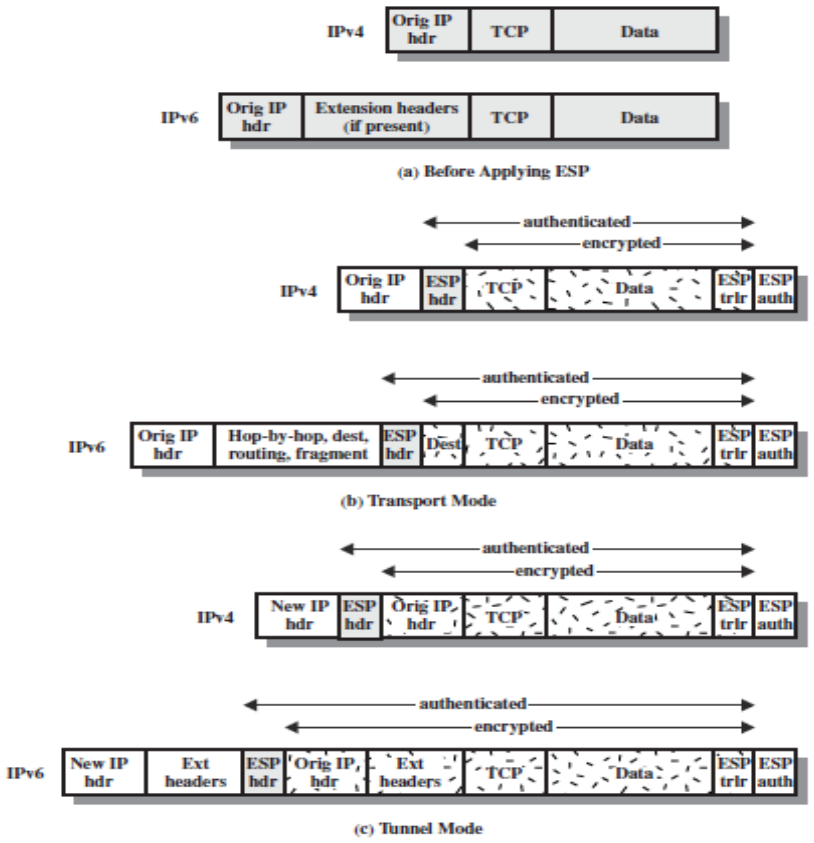
- ✓ Tunnel mode ESP is used to encrypt an entire IP packet. For this mode, the ESP header is prefixed to the packet and then the packet plus the ESP trailer is encrypted.
- ✓ This method can be used to counter traffic analysis.
- ✓ Tunnel mode ESP is used to encrypt an entire IP packet.
- ✓ The following steps occur for transfer of a transport layer segment from the external host to the internal host.

1. The source prepares an inner IP packet with a destination address of the target internal host.

2. The outer packet is routed to the destination firewall.

3. The destination firewall examines and processes the outer IP header plus any outer IP extension headers. This packet is then transmitted in the internal network.

4. The inner packet is routed through zero or more routers in the internal network to the destination host.



Scope of ESP Encryption and Authentication

5.5 COMBINING SECURITY ASSOCIATIONS

- ✓ The term security association bundle refers to a sequence of SAs through which traffic must be processed to provide a desired set of IPsec services.
- ✓ The SAs in a bundle may terminate at different endpoints or at the same endpoints. Security associations may be combined into bundles in two ways:
  - ➢ **Transport adjacency:** Refers to applying more than one security protocol to the same IP packet without invoking tunneling. This approach to combining AH and ESP allows for only one level of combination; further nesting yields no added benefit since the processing is performed at one IPsec instance: the (ultimate) destination.
  - ➢ **Iterated tunneling:** Refers to the application of multiple layers of security protocols effected through IP tunneling. This approach allows for multiple levels of nesting, since each tunnel can originate or terminate at a different IPsec site along the path.

Authentication Plus Confidentiality

Encryption and authentication can be combined in order to transmit an IP packet that has both confidentiality and authentication between hosts.

- ✓ **ESP with Authentication Option** There are two subcases:
  - ➢ **Transport mode ESP:** Authentication and encryption apply to the IP payload delivered to the host, but the IP header is not protected.
  - ➢ **Tunnel mode ESP:** The entire inner IP packet is protected by the privacy mechanism for delivery to the inner IP destination.

Transport Adjacency

- ✓ Another way to apply authentication after encryption is to use two bundled transport SAs, with the inner being an ESP SA and the outer being an AH SA
- ✓ Here ESP is used without its authentication option. Because the inner SA is a transport SA, encryption is applied to the IP payload.
- ✓ The resulting packet consists of an IP header followed by an ESP.
- ✓ AH is then applied in transport mode, so that authentication covers the ESP plus the original IP header except for mutable fields.

Advantage

- ➢ This approach over simply using a single ESP SA with the ESP authentication option is that the authentication covers more fields, including the source and destination IP addresses.

Disadvantage

- ➢ The disadvantage is the overhead of two SAs versus one SA.

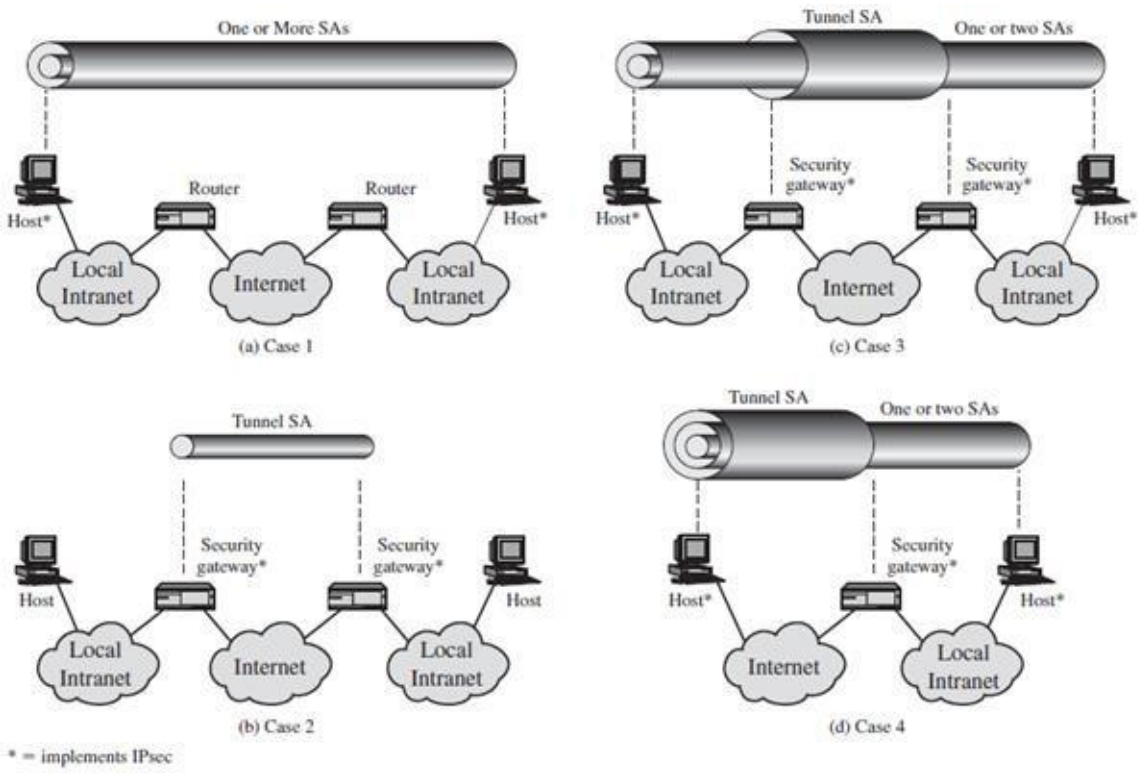
Transport-Tunnel Bundle

- ✓ The use of authentication prior to encryption might be preferable for several reasons.
  1. Because the authentication data are protected by encryption, it is impossible for anyone to intercept the message and alter the authentication data without detection.
  2. It may be desirable to store the authentication information with the message at the destination for later reference.

Basic Combinations of Security Associations

- ✓ The IPsec Architecture document lists four examples of combinations of SAs that must be supported by compliant IPsec hosts (e.g., workstation, server) or security gateways (e.g. firewall, router).

- ✓ These are illustrated in below Figure.



Basic Combinations of Security Associations

- ✓ The lower part of each case in the figure represents the physical connectivity of the elements; the upper part represents logical connectivity via one or more nested SAs.
- ✓ Each SA can be either AH or ESP. For host-to-host SAs, the mode may be either transport or tunnel; otherwise it must be tunnel mode.

Case 1.

- ➢ All security is provided between end systems that implement IPsec.
- ➢ For any two end systems to communicate via an SA, they must share the appropriate secret keys.
- ➢ Among the possible combinations are
  - **a.** AH in transport mode
  - **b.** ESP in transport mode
  - **c.** ESP followed by AH in transport mode (an ESP SA inside an AH SA)
  - **d.** Any one of a, b, or c inside an AH or ESP in tunnel mode

Case 2.

- ➢ Security is provided only between gateways (routers, firewalls, etc.) and no hosts implement IPsec.
- ➢ This case illustrates simple virtual private network support.
- ➢ The security architecture document specifies that only a single tunnel SA is needed for this case.
- ➢ The tunnel could support AH, ESP, or ESP with the authentication option.

- Nested tunnels are not required, because the IPsec services apply to the entire inner packet.

Case 3.

- This builds on case 2 by adding end-to-end security.
- The same combinations applies for cases 1 and 2 are allowed here.
- The gateway-to-gateway tunnel provides either authentication, confidentiality, or both for all traffic between end systems.
- When the gateway-to-gateway tunnel is ESP, it also provides a limited form of traffic confidentiality.
- Individual hosts can implement any additional IPsec services required for given applications or given users by means of end-to-end SAs.

Case 4.

- This provides support for a remote host that uses the Internet to reach an organization's firewall and then to gain access to some server or workstation behind the firewall.
- Only tunnel mode is required between the remote host and the firewall.
- As in case 1, one or two SAs may be used between the remote host and the local host.

5.6 KEY MANAGEMENT

- ✓ The key management of IPsec involves the determination and distribution of secret keys.
- ✓ A typical requirement is four keys for communication between two applications; transmit and receive pairs for both integrity and confidentiality.
- ✓ The IPsec Architecture document mandates support for two types of key management:
  - **Manual:** A system administrator manually configures each system with its own keys and with the keys of other communicating systems.
  - **Automated:** An automated system enables the on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system with an evolving configuration.
- ✓ The default automated key management protocol for IPsec is referred to as ISAKMP/Oakley and consists of the following elements:
  - **Oakley Key Determination Protocol:** Oakley is a key exchange protocol based on the Diffie-Hellman algorithm but providing added security. Oakley is generic in that it does not dictate specific formats.
  - **Internet Security Association and Key Management Protocol (ISAKMP):** ISAKMP provides a framework for Internet key management and provides the specific protocol support, including formats, for negotiation of security attributes.

Oakley Key Determination Protocol

- ✓ It is a refinement of the Diffie-Hellman key exchange algorithm.
- ✓ The Diffie-Hellman algorithm has two attractive features:

- Secret keys are created only when needed. There is no need to store secret keys for a long period of time, exposing them to increased vulnerability.
- The exchange requires no pre-existing infrastructure other than an agreement on the global parameters. The oakley key determination algorithm is characterized by five important features:

Disadvantages of Diffie-Hellman key exchange algorithm

1. It does not provide any information about the identities of the parties.
2. It is subject to a man-in-the-middle attack, in which a third party C impersonates B while communicating with A and impersonates A while communicating with B. Both A and B end up negotiating a key with C, which can then listen to and pass on traffic
3. B computes a secret key K_1 based on B's private key and Y_E. A computes a secret key K_2 based on A's private key and Y_E. E computes K_1 using E's secret key X_E and Y_B and computers K_2 using X_E and Y_A.
4. E is able to relay messages from A to B and from B to A, appropriately changing their encipherment en route in such a way that neither A nor B will know that they share their communication with E.

Features of Oakley

The IKE key determination algorithm is characterized by five important features:

1. It employs a mechanism known as cookies to thwart clogging attacks.
2. It enables the two parties to negotiate a *group*; this, in essence, specifies the global parameters of the Diffie-Hellman key exchange.
3. It uses nonces to ensure against replay attacks.
4. It enables the exchange of Diffie-Hellman public key values.
5. It authenticates the Diffie-Hellman exchange to thwart man-in-the-middle attacks.

ISAKMP mandates that cookie generation satisfy three basic requirements:

- The cookie must depend on the specific parties.
- It must not be possible for anyone other than the issuing entity to generate cookies that will be accepted by that entity
- The cookie generation and verification methods must be fast to thwart attacks intended to sabotage processor resources

Three different **authentication** methods can be used with Oakley:

- **Digital signatures:** The exchange is authenticated by signing a mutually obtainable hash; each party encrypts the hash with its private key. The hash is generated over important parameters, such as user IDs and nonces.
- **Public-key encryption:** The exchange is authenticated by encrypting parameters such as IDs and nonces with the sender's private key.

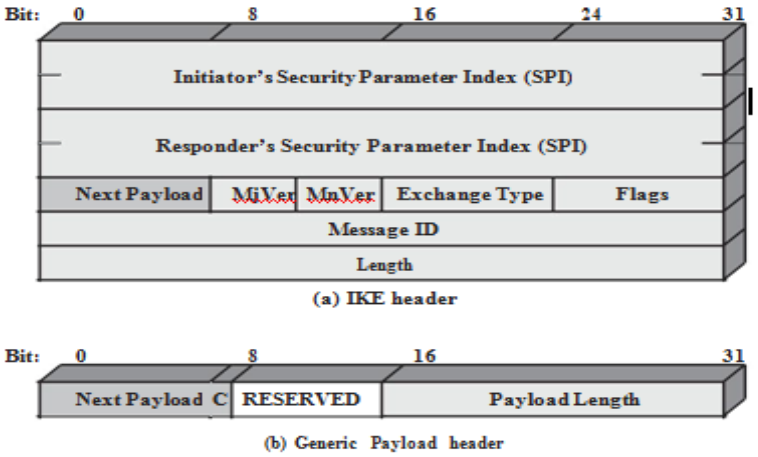
- **Symmetric-key encryption:** A key derived by some out-of-band mechanism can be used to authenticate the exchange by symmetric encryption of exchange parameters.

ISAKMP

- ✓ It defines procedures and packet formats to establish, negotiate, modify, and delete security associations.
- ✓ As part of SA establishment, IKE defines payloads for exchanging key generation and authentication data.
- ✓ These payload formats provide a consistent framework independent of the specific key exchange protocol, encryption algorithm, and authentication mechanism.

ISAKMP Header Format

- ✓ An ISAKMP message consists of an ISAKMP header followed by one or more payloads.
- ✓ All of this is carried in a transport protocol. The specification dictates that implementations must support the use of UDP for the transport protocol.
- ✓ The below Figure shows the header format for an ISAKMP message.



IKE Formats

- ✓ It consists of the following fields.
  - **Initiator SPI (64 bits):** A value chosen by the initiator to identify a unique ISAKMP security association (SA).
  - **Responder SPI (64 bits):** A value chosen by the responder to identify a unique ISAKMP SA.
  - **Next Payload (8 bits):** Indicates the type of the first payload in the message
  - **Major Version (4 bits):** Indicates major version of ISAKMP in use.

- **Minor Version (4 bits):** Indicates minor version in use. • Exchange Type (8 bits): Indicates the type of exchange
- **Flags (8 bits):** Indicates specific options set for this ISAKMP exchange.
- **Message ID (32 bits):** Used to control retransmission of lost packets and matching of requests and responses.
- **Length (32 bits):** Length of total message (header plus all payloads) in octets.

ISAKMP Payload Types

- ✓ IKE payloads begin with the same generic payload header shown in Figure IKE Format(b).
- ✓ The Next Payload field has a value of 0 if this is the last payload in the message
- ✓ The Payload Length field indicates the length in octets of this payload, including the generic payload header.
- ✓ The critical bit is 0 if the sender wants the recipient to skip this payload if it does not understand the payload type code in the Next Payload field of the previous payload.
- ✓ It is set to 1 if the sender wants the recipient to reject this entire message if it does not understand the payload type.
- ✓ The below Table summarizes the payload types defined for IKE and lists the fields, or parameters, that are part of each payload.

IKE Payload Types

Type	Parameters
Security Association	Proposals
Key Exchange	DH Group #, Key Exchange Data
Identification	ID Type, ID Data
Certificate	Cert Encoding, Certificate Data
Certificate Request	Cert Encoding, Certification Authority
Authentication	Auth Method, Authentication Data
Nonce	Nonce Data
Notify	Protocol-ID, SPI Size, Notify Message Type, SPI, Notification Data
Delete	Protocol-ID, SPI Size, # of SPIs, SPI (one or more)
Vendor ID	Vendor ID
Traffic Selector	Number of TSs, Traffic Selectors
Encrypted	IV, Encrypted IKE payloads, Padding, Pad Length, ICV
Configuration	CFG Type, Configuration Attributes
Extensible Authentication Protocol	EAP Message

1. Security Association:

- ✓ The **SA payload** is used to begin the establishment of an SA. The payload has a complex, hierarchical structure.
- ✓ The payload may contain multiple proposals. Each proposal may contain multiple protocols.
- ✓ Each protocol may contain multiple transforms. And each transform may contain multiple attributes.
- ✓ These elements are formatted as substructures within the payload as follows.
  - ➢ **Proposal:** This substructure includes a proposal number, a protocol ID (AH, ESP, or IKE), an indicator of the number of transforms, and then a transform substructure.
  - ➢ **Transform:** The transforms are used primarily to define cryptographic algorithms to be used with a particular protocol.
  - ➢ **Attribute:** Each transform include attributes that modify or complete the specification of the transform. An example is key length.

2. Key Exchange payload:

- ✓ The **Key Exchange payload** can be used for a variety of key exchange techniques, including Oakley, Diffie-Hellman, and the RSA-based key exchange used by PGP.
- ✓ The Key Exchange data field contains the data required to generate a session key and is dependent on the key exchange algorithm used.

3. Identification Payload:

- ✓ The **Identification payload** is used to determine the identity of communicating peers and may be used for determining authenticity of information. Typically the ID Data field will contain an IPv4 or IPv6 address.

4. Certificate payload :

- ✓ The **Certificate payload** transfers a public-key certificate. The Certificate Encoding field indicates the type of certificate or certificate-related information, which include the following:
  - PKCS #7 wrapped X.509 certificate
  - PGP certificate
  - DNS signed key
  - X.509 certificate—signature
  - X.509 certificate—key exchange
  - Kerberos tokens
  - Certificate Revocation List (CRL)
  - Authority Revocation List (ARL)
  - SPKI certificate

5. Certificate Request:

- ✓ At any point in an IKE exchange, the sender may include a **Certificate Request** payload to request the certificate of the other communicating entity.

- ✓ The payload may list more than one certificate type that is acceptable and more than one certificate authority that is acceptable.

6. Authentication payload:

- ✓ The **Authentication** payload contains data used for message authentication purposes.The authentication method types so far defined are RSA digital signature, shared-key message integrity code, and DSS digital signature.

7. Nonce payload :

- ✓ The **Nonce** payload contains random data used to guarantee liveness during an exchange and to protect against replay attacks.

8. Notify payload:

- ✓ The **Notify** payload contains either error or status information associated with this SA or this SA negotiation.

9. Delete payload:

- ✓ The **Delete** payload indicates one or more SAs that the sender has deleted from its database and that therefore are no longer valid.

10. Vendor ID payload:

- ✓ The **Vendor ID** payload contains a vendor-defined constant. The constant issued by vendors to identify and recognize remote instances of their implementations. This mechanism allows a vendor to experiment with new features while maintaining backward compatibility.

11.Traffic Selector payload:

- ✓ The **Traffic Selector** payload allows peers to identify packet flows for processing by IPsec services.

12. Traffic Selector payload:

- ✓ The **Encrypted** payload contains other payloads in encrypted form. The encrypted payload format is similar to that of ESP. It may include an IV if the encryption algorithm requires it and an ICV if authentication is selected.

13.Configuration payload:

- ✓ The **Configuration** payload is used to exchange configuration information between IKE peers.

14.Extensible Authentication Protocol (EAP) payload: The **Extensible Authentication Protocol (EAP)** payload allows IKE SAs to be authenticated using EAP,

MODULE 1

CLASSICAL ENCRYPTION TECHNIQUES

Symmetric encryption, also referred to as conventional encryption or single-key encryption, was the only type of encryption in use prior to the development of public-key encryption in the 1970s. It remains by far the most widely used of the two types of encryption. Part One examines a number of symmetric ciphers. In this chapter, we begin with a look at a general model for the symmetric encryption process; this will enable us to understand the context within which the algorithms are used. Next, we examine a variety of algorithms in use before the computer era. Finally, we look briefly at a different approach known as steganography. Chapter 3 examines the most widely used symmetric cipher: DES.

Before beginning, we define some terms. An original message is known as the **plaintext**, while the coded message is called the **ciphertext**. The process of converting from plaintext to ciphertext is known as **enciphering** or **encryption**; restoring the plaintext from the ciphertext is **deciphering** or **decryption**. The many schemes used for encryption constitute the area of study known as **cryptography**. Such a scheme is known as a **cryptographic system** or a **cipher**. Techniques used for deciphering a message without any knowledge of the enciphering details fall into the area of **cryptanalysis**. Cryptanalysis is what the layperson calls "breaking the code." The areas of cryptography and cryptanalysis together are called **cryptology**.

Symmetric Cipher Model

A symmetric encryption scheme has five ingredients (Figure 2.1):

Plaintext: This is the original intelligible message or data that is fed into the algorithm as input.

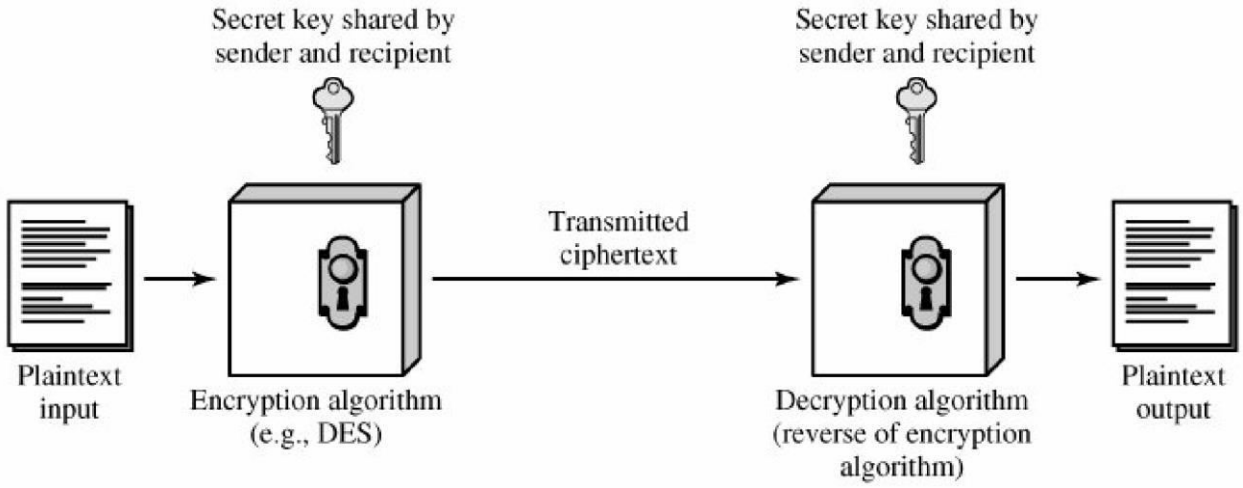
Encryption algorithm: The encryption algorithm performs various substitutions and transformations on the plaintext.

Secret key: The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time. The exact substitutions and transformations performed by the algorithm depend on the key.

Cipher text: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different cipher texts. The cipher text is an apparently random stream of data and, as it stands, is unintelligible.

Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the cipher text and the secret key and produces the original plaintext.

Figure 2.1. Simplified Model of Conventional Encryption



There are two requirements for secure use of conventional encryption:

1. We need a strong encryption algorithm. At a minimum, we would like the algorithm to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key. This requirement is usually stated in a stronger form: The opponent should be unable to decrypt ciphertext or discover the key even if he or she is in possession of a number of ciphertexts together with the plaintext that produced each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the ciphertext *plus* knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption is

what makes it feasible for widespread use. The fact that the algorithm need not be kept secret means that manufacturers can and have developed low-cost chip implementations of data encryption algorithms. These chips are widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Let us take a closer look at the essential elements of a symmetric encryption scheme, using Figure 2.2. A source produces a message in plaintext, $X = [X1, X2, ..., XM]$. The M elements of X are letters in some finite alphabet. Traditionally, the alphabet usually consisted of the 26 capital letters. Nowadays, the binary alphabet {0, 1} is typically used. For encryption, a key of the form $K = [K1, K2, ..., KJ]$ is generated. If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel. Alternatively, a third party could generate the key and securely deliver it to both source and destination.

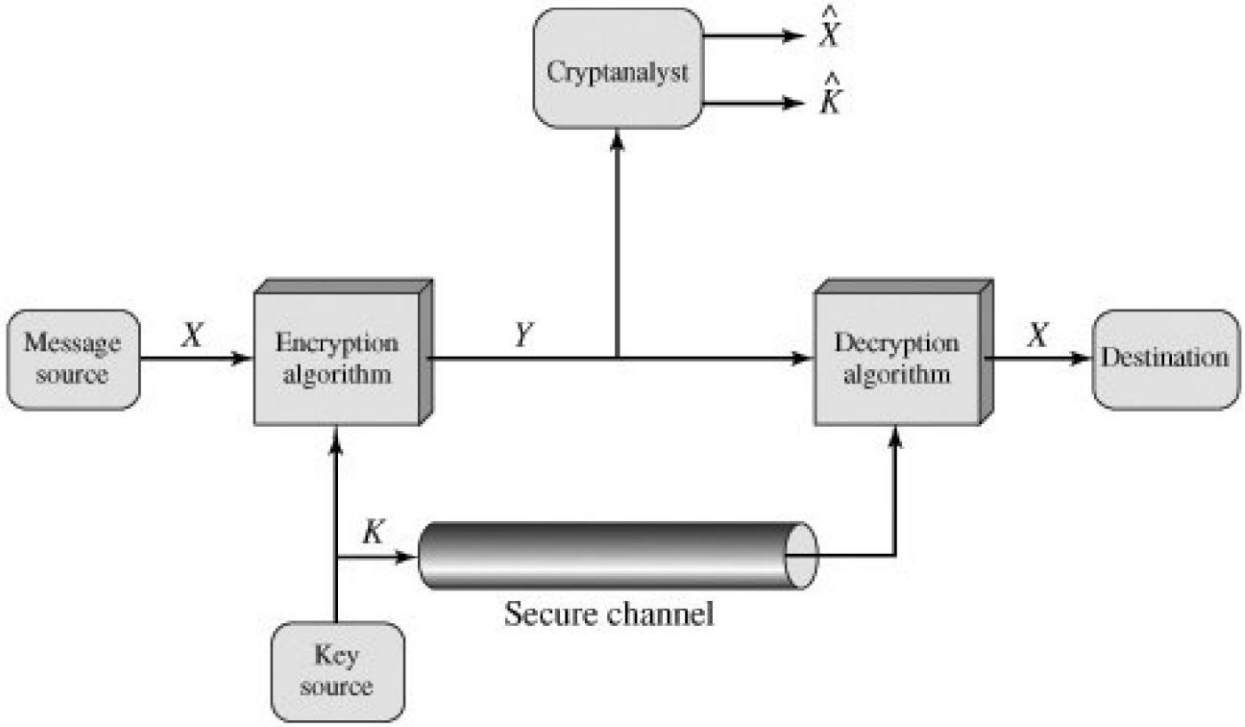


Figure 2.2. Model of Conventional Cryptosystem

With the message X and the encryption key K as input, the encryption algorithm forms the ciphertext $Y = [Y1, Y2, ..., YN]$. We can write this As $Y = E(K, X)$

This notation indicates that Y is produced by using encryption algorithm E as a function of the plaintex$X$t , with the specific function determined by the value of the key K.

The intended receiver, in possession of the key, is able to invert the transformation:

$X = D(K, Y)$

An opponent, observing Y but not having access to K or X, may attempt to recover X or K or both X and K. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. If the opponent is interested in only this particular message, then the focus of the effort is to recover X by generating a plaintext estimate . Often, however, the opponent is interested in being able to read future messages as well, in which case an attempt is made to recover K by generating an estimate .

Cryptography

Cryptographic systems are characterized along three independent dimensions:

1. The type of operations used for transforming plaintext to ciphertext. All encryption algorithms are based on two general principles: substitution, in which each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element, and transposition, in which elements in the plaintext are rearranged. The fundamental requirement is that no information be lost (that is, that all operations are reversible). Most systems, referred to as *product systems*, involve multiple stages of substitutions and transpositions.

2. The number of keys used. If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption. If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.

3.The way in which the plaintext is processed. A *block cipher* processes the input one block of elements at a time, producing an output block for each input block. A *stream cipher* processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis

Typically, the objective of attacking an encryption system is to recover the key in use rather then simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis:** Cryptanalytic attacks rely on the nature of the algorithm plus perhaps some knowledge of the general characteristics of the plaintext or even some sample plaintext-ciphertext pairs. This type of attack exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.

- **Brute-force attack:** The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success. If either type of attack succeeds in deducing the key, the effect is catastrophic: All future and past messages encrypted with that key are compromised.

Table 2.1 summarizes the various types of **cryptanalytic attacks**, based on the amount of information known to the cryptanalyst. The most difficult problem is presented when all that is available is the *cipher text only*. In some cases, not even the encryption algorithm is known, but in general we can assume that the opponent does know the algorithm used for encryption. One possible attack under these circumstances is the brute-force approach of trying all possible keys. If the key space is very large, this becomes impractical. Thus, the opponent must rely on an analysis of the cipher text itself, generally applying various statistical tests to it.

Table 2.1. Types of Attacks on Encrypted Messages

Type of Attack Known to Cryptanalyst

Cipher text only

- Encryption algorithm
- Cipher text

Known plaintext

- Encryption algorithm
- Cipher text
- One or more plaintext-cipher text pairs formed with the secret key

Chosen plaintext

- Encryption algorithm
- Cipher text
- Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key

Chosen cipher text

- Encryption algorithm
- Cipher text

- Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

Chosen text

- Encryption algorithm
- Ciphertext
- Plaintext message chosen by cryptanalyst, together with its corresponding
- ciphertext generated with the secret key
- Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key

A **brute-force attack** involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

Results are shown for four binary key sizes. The 56-bit key size is used with the DES (Data Encryption Standard) algorithm, and the 168-bit key size is used for triple DES. The minimum key size specified for AES (Advanced Encryption Standard) is 128 bits. Results are also shown for what are called substitution codes that use a 26-character key (discussed later), in which all possible permutations of the 26 characters serve as keys. For each key size, the results are shown assuming that it takes 1 *m*s to perform a single decryption, which is a reasonable order of magnitude for today's machines. With the use of massively parallel organizations of microprocessors, it may be possible to achieve processing rates many orders of magnitude greater.

Substitution Techniques

In this section and the next, we examine a sampling of what might be called classical encryption techniques. A study of these techniques enables us to illustrate the basic approaches to symmetric encryption used today and the types of cryptanalytic attacks that must be anticipated.

The two basic building blocks of all encryption techniques are substitution and transposition. We examine these in the next two sections. Finally, we discuss a system that combines both substitution and transposition. A substitution technique is one in which the letters of plaintext are replaced by other letters or by numbers or symbols. If the plaintext is viewed as a sequence of bits, then substitution involves replacing plaintext bit patterns with ciphertext bit patterns

Caesar Cipher

The earliest known use of a substitution cipher, and the simplest, was by Julius Caesar. The Caesar cipher involves replacing each letter of the alphabet with the letter standing three places further down the alphabet. For example,

plain: meet me after the toga party

Cipher: PHHW PH DIWHU WKH WRJD SDUWB

Note that the alphabet is wrapped around, so that the letter following Z is A. We can define the transformation by listing all possibilities, as follows:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

cipher: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

Let us assign a numerical equivalent to each letter:

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Then the algorithm can be expressed as follows. For each plaintext letter p, substitute the ciphertext letter C:

$C = \mathrm{E}(3, p) = (p + 3) \bmod 26$

A shift may be of any amount, so that the general Caesar algorithm is

$C = \mathrm{E}(k, p) = (p + k) \bmod 26$

where k takes on a value in the range 1 to 25. The decryption algorithm is simply

$p = \mathrm{D}(k, C) = (C\ k) \bmod 26$

If it is known that a given ciphertext is a Caesar cipher, then a brute-force cryptanalysis is easily performed: Simply try all the 25 possible keys. Figure 2.3 shows the results of applying this strategy to the example ciphertext. In this case, the plaintext leaps out as occupying the third line.

KEY	PHHW PH DIWHU WKH WRJD SDUWB
1	oggv og chvgt vjg vqic rctva
2	nffu nf bgufs uif uphb qbsuz
3	meet me after the toga party
4	ldds ld zesdq sgd snfz ozqsx
5	kccr kc ydrcp rfc rmey nyprw
6	jbbq jb xcqbo qeb qldx mxoqv
7	iaap ia wbpan pda pkcw lwnpu
8	hzzo hz vaozm ocz ojbv kvmot
9	gyyn gy uznyl nby niau julns
10	fxxm fx tymxk max mhzt itkmr
11	ewwl ew sxlwj lzw lgys hsjlq
12	dvvk dv rwkvi kyv kfxr grikp
13	cuuj cu qvjuh jxu jewq fqhjo
14	btti bt puitg iwt idvp epgin
15	assh as othsf hvs hcuo dofhm
16	zrrg zr nsgre gur gbtn cnegl
17	yqqf yq mrfqd ftq fasm bmdfk
18	xppe xp lqepc esp ezrl alcej
19	wood wo kpdob dro dyqk zkbdi
20	vnnc vn jocna cqn cxpj yjach
21	ummb um inbmz bpm bwoi xizbg
22	tlla tl hmaly aol avnh whyaf
23	skkz sk glzkx znk zumg vgxze
24	rjjy rj fkyjw ymj ytlf ufwyd
25	qiix qi ejxiv xli xske tevxc

Three important characteristics of this problem enabled us to use a brute-force cryptanalysis:

1. The encryption and decryption algorithms are known.

2. There are only 25 keys to try.

3. The language of the plaintext is known and easily recognizable

In most networking situations, we can assume that the algorithms are known. What generally makes brute-force cryptanalysis impractical is the use of an algorithm that employs a large number of keys. For example, the triple DES algorithm, examined in Chapter 6, makes use of a 168-bit key, giving a key space of 2 168 or greater than 3.7 x 10^5 possible keys. The third characteristic is also significant. If the language of the plaintext is unknown, then plaintext output may not be recognizable. Furthermore, the input may be abbreviated or compressed in some fashion, again making recognition difficult.

Monoalphabetic Ciphers

With only 25 possible keys, the Caesar cipher is far from secure. A dramatic increase in the key space can be achieved by allowing an arbitrary substitution. Recall the assignment for the Caesar cipher:

plain: a b c d e f g h i j k l m n o p q r s t u v w x y z

cipher: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters, then there are 26! or greater than 4 x 10^26 possible keys. This is 10 orders of magnitude greater than the key space for DES and would seem to eliminate brute-force techniques for cryptanalysis. Such an approach is referred to as a **monoalphabetic substitution cipher**, because a single cipher alphabet (mapping from plain alphabet to cipher alphabet) is used per message. There is, however, another line of attack. If the cryptanalyst knows the nature of the plaintext (e.g., noncompressed English text), then the analyst can exploit the regularities of the language. To see how such a cryptanalysis might proceed. The ciphertext to be solved is

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ

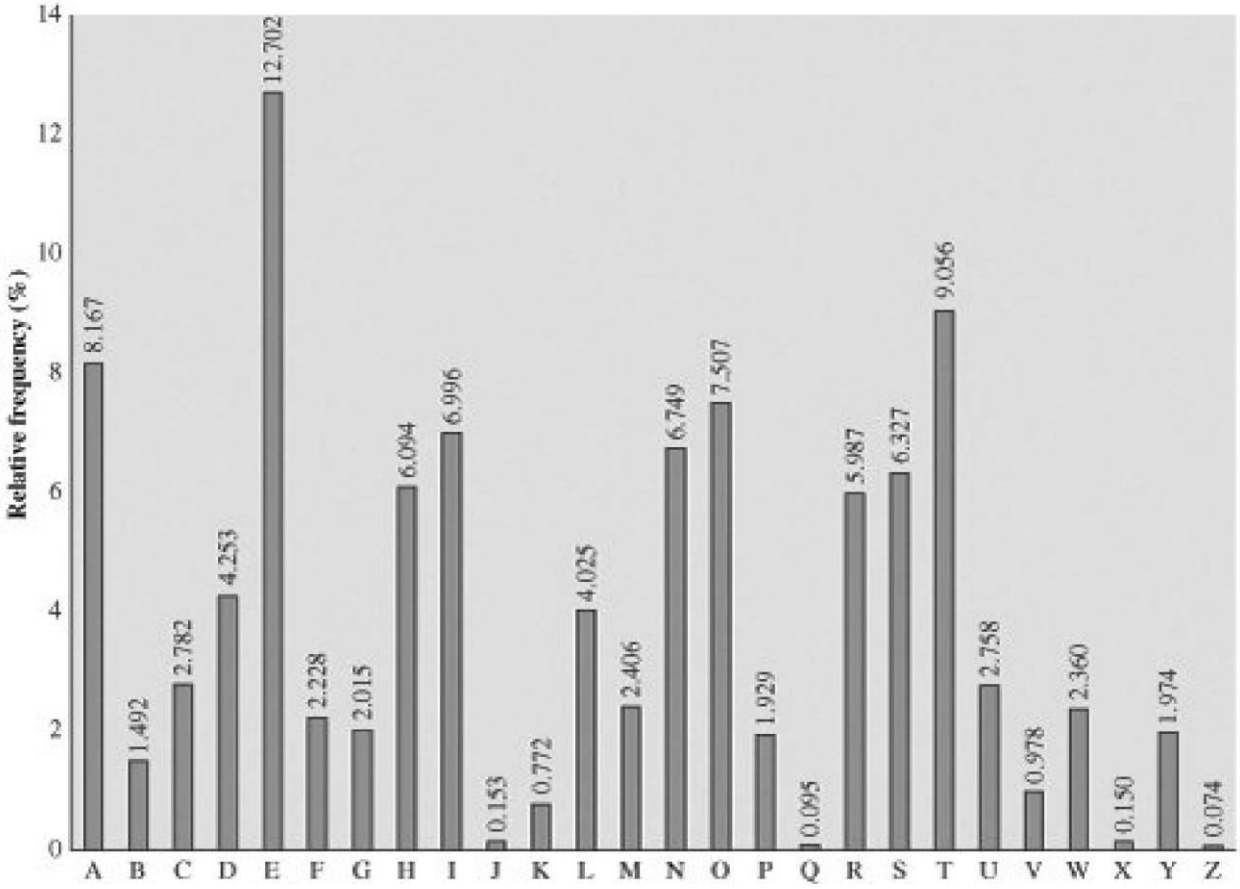
VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX

EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

As a first step, the relative frequency of the letters can be determined and compared to a standard frequency distribution for English, such as is shown in Figure 2.5. If the message were long enough, this technique alone might be sufficient, but because this is a relatively short message, we cannot expect an exact match. In any case, the relative frequencies of the letters in the ciphertext (in percentages) are as follows:

P 13.33	H 5.83	F 3.33	B 1.67	C 0.00
Z 11.67	D 5.00	W 3.33	G 1.67	K 0.00
S 8.33	E 5.00	Q 2.50	Y 1.67	L 0.00
U 8.33	V 4.17	T 2.50	I 0.83	N 0.00
O 7.50	X 4.17	A 1.67	J 0.83	R 0.00
M 6.67				

Figure 2.5. Relative Frequency of Letters in English Text



Comparing this breakdown with Figure 2.5, it seems likely that cipher letters P and Z are the equivalents of plain letters e and t, but it is not certain which is which. The letters S, U, O, M, and H are all of relatively high frequency and probably correspond to plain letters from the set {a, h, i, n, o, r, s}.The letters with the lowest frequencies (namely, A, B, G, Y, I, J) are likely included in the set {b, j, k, q, v, x, z}. There are a number of ways to proceed at this point. We could make some tentative assignments and start to fill in the plaintext to see if it looks like a

reasonable "skeleton" of a message. A more systematic approach is to look for other regularities. For example, certain words may be known to be in the text. Or we could look for repeating sequences of cipher letters and try to deduce their plaintext equivalents.

A powerful tool is to look at the frequency of two-letter combinations, known as digrams. A table similar to Figure 2.5 could be drawn up showing the relative frequency of digrams. The most common such digram is th. In our cipher text, the most common digram is ZW, which appears three times. So we make the correspondence of Z with t and W with h. Then, by our earlier hypothesis, we can equate P with e. Now notice that the sequence ZWP appears in the cipher text, and we can translate that sequence as "the." This is the most frequent trigram (three-letter combination) in English, which seems to indicate that we are on the right track. Next, notice the sequence ZWSZ in the first line. We do not know that these four letters form a complete word, but if they do, it is of the form th_t. If so, S equates with a. So far, then, we have

UZQSOVUOHXMOPVGPOZPEVSGZWSZOPFPESXUDBMETSXAIZ

t a e e te a that e e a a

VUEPHZHMDZSHZOWSFPAPPDTSVPQUZWYMXUZUHSX

e t ta t ha e ee a e th t a

EPYEPOPDZSZUFPOMBZWPFUPZHMDJUDTMOHMQ

e e e tat e the t

Only four letters have been identified, but already we have quite a bit of the message. Continued analysis of frequencies plus trial and error should easily yield a solution from this point. The complete plaintext, with spaces added between words, follows: it was disclosed yesterday that several informal but direct contacts have been made with political representatives of the viet cong in Moscow Monoalphabetic ciphers are easy to break because they reflect the frequency data of the original alphabet. A countermeasure is to provide multiple substitutes, known as homophones, for a single letter. For example, the letter e could be assigned a number of different cipher symbols, such as 16, 74, 35, and 21, with each homophone used in rotation, or randomly. If the number of symbols assigned to each letter is proportional to the relative frequency of that letter, then single-letter frequency information is completely obliterated. The great mathematician Carl Friedrich Gauss believed that he had devised an unbreakable cipher using homophones. However, even with homophones, each element of plaintext affects only one element of ciphertext, and multiple-letter patterns (e.g., digram frequencies) still survive in the

ciphertext, making cryptanalysis relatively straightforward. Two principal methods are used in substitution ciphers to lessen the extent to which the structure of the plaintext survives in the ciphertext: One approach is to encrypt multiple letters of plaintext, and the other is to use multiple cipher alphabets. We briefly examine each.

Playfair Cipher

The best-known multiple-letter encryption cipher is the Playfair, which treats digrams in the plaintext as single units and translates these units into ciphertext digrams.

The Playfair algorithm is based on the use of a 5 x 5 matrix of letters constructed using a keyword. Here is an example, solved by Lord Peter Wimsey in Dorothy Sayers's *Have His Carcase*.

M O N A R

C H Y B D

E F G I/J K

L P Q S T

U V W X Z

In this case, the keyword is *monarchy*. The matrix is constructed by filling in the letters of the keyword (minus duplicates) from left to right and from top to bottom, and then filling in the remainder of the matrix with the remaining letters in alphabetic order. The letters I and J count as one letter. Plaintext is encrypted two letters at a time, according to the following rules:

1. Repeating plaintext letters that are in the same pair are separated with a filler letter, such as x, so that balloon would be treated as ba lx lo on.

2. Two plaintext letters that fall in the same row of the matrix are each replaced by the letter to the right, with the first element of the row circularly following the last. For example, ar is encrypted as RM.

3. Two plaintext letters that fall in the same column are each replaced by the letter beneath, with the top element of the column circularly following the last. For example, mu is encrypted as CM.

4. Otherwise, each plaintext letter in a pair is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter. Thus, hs becomes BP and ea becomes IM (or JM, as the encipherer wishes).

The Playfair cipher is a great advance over simple monoalphabetic ciphers. For one thing, whereas there are only 26 letters, there are 26 x 26 = 676 digrams, so that identification of

individual digrams is more difficult. Furthermore, the relative frequencies of individual letters exhibit a much greater range than that of digrams, making frequency analysis much more difficult. For these reasons, the Playfair cipher was for a long time considered unbreakable. It was used as the standard field system by the British Army in World War I and still enjoyed considerable use by the U.S. Army and other Allied forces during World War II. Despite this level of confidence in its security, the Playfair cipher is relatively easy to break because it still leaves much of the structure of the plaintext language intact. A few hundred letters of ciphertext are generally sufficient.

Hill Cipher

Another interesting multiletter cipher is the Hill cipher, developed by the mathematician Lester Hill in 1929. The encryption algorithm takes *m* successive plaintext letters and substitutes for them *m* ciphertext letters. The substitution is determined by *m* linear equations in which each character is assigned a numerical value (a = 0, b = 1 ... z = 25). For *m* = 3, the system can be described as follows:

c1 = ($k11P1 + k12P2 + k13P3$) mod 26

c2 = ($k21P1 + k22P2 + k23P3$) mod 26

c3 = ($k31P1 + k32P2 + k33P3$) mod 26

This can be expressed in term of column vectors and matrices:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \bmod 26$$

or

C = **KP** mod 26

where **C** and **P** are column vectors of length 3, representing the plaintext and ciphertext, and **K** is a 3 x 3 matrix, representing the encryption key. Operations are performed mod 26. For example, consider the plaintext "paymoremoney" and use the encryption key

$$\mathbf{K} = \begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}$$

The first three letters of the plaintext are represented by the vector $\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix}$. Then $\mathbf{K}\begin{pmatrix} 15 \\ 0 \\ 24 \end{pmatrix} = \begin{pmatrix} 375 \\ 819 \\ 486 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 13 \\ 18 \end{pmatrix} =$ LNS. Continuing in this fashion, the ciphertext for the entire plaintext is LNSHDLEWMTRW.

Decryption requires using the inverse of the matrix **K**. The inverse **K-**1 of a matrix **K** is defined by the equation **KK-**1 = **K-**1**K** = **I**, where **I** is the matrix that is all zeros except for ones along the main diagonal from upper left to lower right. The inverse of a matrix does not always exist, but when it does, it satisfies the preceding equation. In this case, the inverse

$$\mathbf{K}^{-1} = \begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix}$$

This is demonstrated as follows:

$$\begin{pmatrix} 17 & 17 & 5 \\ 21 & 18 & 21 \\ 2 & 2 & 19 \end{pmatrix}\begin{pmatrix} 4 & 9 & 15 \\ 15 & 17 & 6 \\ 24 & 0 & 17 \end{pmatrix} = \begin{pmatrix} 443 & 442 & 442 \\ 858 & 495 & 780 \\ 494 & 52 & 365 \end{pmatrix} \bmod 26 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

It is easily seen that if the matrix **K-**1 is applied to the ciphertext, then the plaintext is recovered. To explain how the inverse of a matrix is determined, we make an exceedingly brief excursion into linear algebra. For any square matrix (*m* x *m*) the **determinant** equals the sum of all the products that can be formed by taking exactly one element from each row and exactly one element from each column, with certain of the product terms preceded by a minus sign. For a 2*2 matrix,

$$\begin{pmatrix} k_{11} & k_{12} \\ k_{21} & k_{22} \end{pmatrix}$$

the determinant is $k11k22$ $k12k21$. For a 3 x 3 matrix, the value of the determinant is$k$ $11k22k33$ + $k21k32k13$ + $k31k12k23$ $k31k22k13$

$k21k12k33$ $k11k32k23$. If a square matrix**A** has a nonzero determinant, then the inverse of the matrix is computed as$\boldsymbol{A^1}ij$ =$(1)^{I+J}$(Dij)/ded(**A**), where (Dij) is the subdeterminant formed by deleting the ith row and the jth column of **A** and det(**A**) is the determinant of **A**. For our purposes, all arithmetic is done mod 26.

In general terms, the Hill system can be expressed as follows:

C = E(**K, P**) = **KP** mod 26

P = D(**K, P**) = $\boldsymbol{K^1}$**C** mod 26 = $\boldsymbol{K^1}$ **KP** = **P**

As with Playfair, the strength of the Hill cipher is that it completely hides single-letter frequencies. Indeed, with Hill, the use of a larger

matrix hides more frequency information. Thus a 3 x 3 Hill cipher hides not only single-letter but also two-letter frequency information.

Although the Hill cipher is strong against a cipher text-only attack, it is easily broken with a known plaintext attack. For an m x m Hill cipher,

suppose we have m plaintext-ciphe r text pairs, each of length m. We label the pairs

$$\mathbf{P}_j = \begin{pmatrix} p_{1j} \\ p_{2j} \\ \vdots \\ p_{mj} \end{pmatrix} \text{ and } \mathbf{C}_j = \begin{pmatrix} c_{1j} \\ c_{2j} \\ \vdots \\ c_{mj} \end{pmatrix} \text{ such that } \mathbf{C}_j = \mathbf{KP}_j \text{ for } 1 \le j \le m \text{ and for some}$$

unknown key matrix **K**. Now define two m x m matrices **X** = (Pij) and **Y** = (Cij). Then we can form the matrix equation **Y** = **KX**. If **X** has an inverse, then we can determine **K** = **YX**1.If **X** is not invertible, then a new version of **X** can be formed with additional plaintext-ciphertext pairs until an invertible **X** is obtained. Suppose that the plaintext "friday" is encrypted using a 2 x 2 Hill cipher to yield the ciphertext PQCFKU. Thus, we know that

$$\mathbf{K}\begin{pmatrix} 5 \\ 17 \end{pmatrix} \bmod 26 = \begin{pmatrix} 15 \\ 16 \end{pmatrix}; \mathbf{K}\begin{pmatrix} 8 \\ 3 \end{pmatrix} \bmod 26 = \begin{pmatrix} 2 \\ 5 \end{pmatrix}; \quad \text{and} \quad \mathbf{K}\begin{pmatrix} 0 \\ 24 \end{pmatrix} \bmod 26 = \begin{pmatrix} 10 \\ 20 \end{pmatrix}$$

Using the first two plaintext-ciphertext pairs, we have

$$\begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix} = \mathbf{K}\begin{pmatrix} 5 & 8 \\ 17 & 3 \end{pmatrix} \bmod 26$$

The inverse of **X** can be computed:

$$\mathbf{K} = \begin{pmatrix} 15 & 2 \\ 16 & 5 \end{pmatrix}\begin{pmatrix} 9 & 2 \\ 1 & 15 \end{pmatrix} = \begin{pmatrix} 137 & 60 \\ 149 & 107 \end{pmatrix} \bmod 26 = \begin{pmatrix} 7 & 8 \\ 19 & 3 \end{pmatrix}$$

Polyalphabetic Ciphers

Another way to improve on the simple monoalphabetic technique is to use different monoalphabetic substitutions as one proceeds through the plaintext message. The general name for this approach is **polyalphabetic substitution cipher**. All these techniques have the following features in common:

1. A set of related monoalphabetic substitution rules is used.

2. A key determines which particular rule is chosen for a given transformation.

The best known, and one of the simplest, such algorithm is referred to as the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution rules consists of the 26 Caesar ciphers, with shifts of 0 through 25. Each cipher is denoted by a key letter,Mwhich is the ciphertext letter that substitutes for the plaintext letter a. Thus, a Caesar cipher with a shift of 3 is denoted by the key value *d*. To aid in understanding the scheme and to aid in its use, a matrix known as the Vigenère tableau is constructed (Table 2.3). Each of the 26 ciphers is laid out horizontally, with the key letter for each cipher to its left. A normal alphabet for the plaintext runs across the top. The process of encryption is simple: Given a key letter *x* and a plaintext letter y, the ciphertext letter is at the intersection of the row labeled *x* and the column labeled y; in this case the ciphertext is V.

Key	Plaintext																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
b	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
c	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
d	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
e	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
f	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
g	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
h	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
i	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
j	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
k	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
l	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
m	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
n	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
o	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
p	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
r	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
s	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
t	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
u	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
v	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
w	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
x	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Table 2.3. The Modern Vigenère Tableau

To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword. For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

key: *deceptivedeceptivedeceptive*

plaintext: wearediscoveredsaveyourself

ciphertext: ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Decryption is equally simple. The key letter again identifies the row. The position of the ciphertext letter in that row determines the column, and the plaintext letter is at the top of that column. The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword. Thus, the letter frequency information is obscured. However, not all knowledge of the plaintext structure is lost. For example, Figure 2.6 shows the frequency distribution for a Vigenère cipher with a keyword of length 9. An improvement is achieved over the Playfair cipher, but considerable frequency information remains.

It is instructive to sketch a method of breaking this cipher, because the method reveals some of the mathematical principles that apply in cryptanalysis. First, suppose that the opponent believes that the ciphertext was encrypted using either monoalphabetic substitution or a Vigenère cipher.

A simple test can be made to make a determination. If a monoalphabetic substitution is used, then the statistical properties of the ciphertext should be the same as that of the language of the plaintext. Thus, referring to Figure 2.5, there should be one cipher letter with a relative frequency of occurrence of about 12.7%, one with about 9.06%, and so on. If only a single message is available for analysis, we would not expect an exact match of this small sample with the statistical profile of the plaintext language. Nevertheless, if the correspondence is close, we can assume a monoalphabetic substitution. If, on the other hand, a Vigenère cipher is suspected, then progress depends on determining the length of the keyword, as will be seen in a moment. For now, let us concentrate on how the keyword length can be determined. The important insight that leads to a solution is the following: If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length, they will generate identical ciphertext sequences. In the foregoing example, two instances of the sequence "red" are separated by nine character positions. Consequently, in both cases, r is encrypted using key letter *e*, e is encrypted using key letter *p*, and d is encrypted using keyM letter *t*. Thus, in both cases the ciphertext sequence is VTW. An analyst looking at only the ciphertext would detect the repeated sequences VTW at a displacement of 9 and make the assumption that the keyword is either three or nine letters in length. The appearance of VTW twice could be by chance and not reflect identical plaintext letters encrypted with identical key letters. However, if the message is long enough, there will be a number of such repeated cipher text sequences. By looking for common factors in the displacements of the various sequences, the analyst should be able to make a good guess of the keyword length. Solution of the cipher now depends on an important insight. If the keyword length is *N*, then the cipher, in effect, consists of *N* monoalphabetic substitution ciphers. For example, with the keyword DECEPTIVE, the letters in positions 1, 10, 19, and so on are all encrypted with the same monoalphabetic cipher. Thus, we can use the known frequency characteristics of the plaintext language to attack each of the monoalphabetic ciphers separately.

The periodic nature of the keyword can be eliminated by using a nonrepeating keyword that is as long as the message itself. Vigenère proposed what is referred to as an **autokey system**, in which a keyword is concatenated with the plaintext itself to provide a running key.

For our example,

key: *deceptivewearediscoveredsav*

plaintext: wearediscoveredsaveyourself

ciphertext: ZICVTWQNGKZEIIGASXSTSLVVWLA

The ultimate defense against such a cryptanalysis is to choose a keyword that is as long as the plaintext and has no statistical relationship to it. Such a system was introduced by an AT&T engineer named Gilbert Vernam in 1918. His system works on binary data rather than letters. The system can be expressed succinctly as follows:

$ci = pi + ki$

where

pi = ith binary digit of plaintext

ki = ith binary digit of key

ci = ith binary digit of cipher text

'+'= exclusive-or (XOR) operation

Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key. Because of the properties of the XOR, decryption simply involves the same bitwise operation:

$pi = ci + ki$

The essence of this technique is the means of construction of the key. Vernam proposed the use of a running loop of tape that eventually repeated the key, so that in fact the system worked with a very long but repeating keyword. Although such a scheme, with a long key, presents formidable cryptanalytic difficulties, it can be broken with sufficient ciphertext, the use of known or probable plaintext sequences, or both.

One-Time Pad

An Army Signal Corp officer, Joseph Mauborgne, proposed an improvement to the Vernam cipher that yields the ultimate in security. Mauborgne suggested using a random key that is as long as the message, so that the key need not be repeated. In addition, the key is to

be used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message. Such a scheme, known as a **one-time pad**, is unbreakable. It produces random output that bears no statistical relationship to the plaintext. Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code. An example should illustrate our point. Suppose that we are

using a Vigenère scheme with 27 characters in which the twenty-seventh character is the space character, but with a one-time key that is as long as the message. Thus, the tableau of Table 2.3 must be expanded to 27 x 27. Consider the ciphertext

ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

We now show two different decryptions using two different keys:

ciphertext: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

Key: *pxlmvmsydofuyrvzwc tnlebnecvgdupahfzzlmnyih*

Plaintext: mr mustard with the candlestick in the hall

Cipher text: ANKYODKYUREPFJBYOJDSPLREYIUNOFDOIUERFPLUYTS

Key: *mfugpmiydgaxgoufhklllmhsqdqogtewbqfgyovuhwt*

Plaintext: miss scarlet with the knife in the library

Suppose that a cryptanalyst had managed to find these two keys. Two plausible plaintexts are produced. How is the cryptanalyst to decide which is the correct decryption (i.e., which is the correct key)? If the actual key were produced in a truly random fashion, then them cryptanalyst cannot say that one of these two keys is more likely than the other. Thus, there is no way to decide which key is correct and therefore which plaintext is correct.

In fact, given any plaintext of equal length to the ciphertext, there is a key that produces that plaintext. Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which was the intended plaintext.

Therefore, the code is unbreakable. The security of the one-time pad is entirely due to the randomness of the key. If the stream of characters that constitute the key is truly random, then the stream of characters that constitute the ciphertext will be truly random. Thus, there are no patterns or regularities that a cryptanalyst can use to attack the ciphertext. In theory, we need look no further for a cipher. The one-time pad offers complete security but, in practice, has two fundamental difficulties:

1. There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.

2. Even more daunting is the problem of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver. Thus, a mammoth key distribution problem exists.

Because of these difficulties, the one-time pad is of limited utility, and is useful primarily for low-bandwidth channels requiring very high security.

Transposition Techniques

All the techniques examined so far involve the substitution of a ciphertext symbol for a plaintext symbol. A very different kind of mapping is achieved by performing some sort of permutation on the plaintext letters. This technique is referred to as a transposition cipher. The simplest such cipher is the rail fence technique, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows. For example, to encipher the message "meet me after the toga party" with a rail fence of depth 2, we write the following:

m e m a t r h t g p r y

e t e f e t e o a a t

The encrypted message is

MEMATRHTGPRYETEFETEOAAT

This sort of thing would be trivial to cryptanalyze. A more complex scheme is to write the message in a rectangle, row by row, and read the message off, column by column, but permute the order of the columns. The order of the columns then becomes the key to the algorithm. For example,

Key: 4 3 1 2 5 6 7

Plaintext: a t t a c k p

o s t p o n e

d u n t i l t

w o a m x y z

Ciphertext: TTNAAPTMTSUOAODWCOIXKNLYPETZ

A pure transposition cipher is easily recognized because it has the same letter frequencies as the original plaintext. For the type of columnar transposition just shown, cryptanalysis is fairly straightforward and involves laying out the ciphertext in a matrix and playing around with

column positions. Digram and trigram frequency tables can be useful. The transposition cipher can be made significantly more secure by performing more than one stage of transposition. The result is a more complex permutation that is not easily reconstructed. Thus, if the foregoing message is reencrypted using the same algorithm,

Key: 4 3 1 2 5 6 7

Input: t t n a a p t

m t s u o a o

d w c o i x k

n l y p e t z

Output: NSCYAUOPTTWLTMDNAOIEPAXTTOKZ

To visualize the result of this double transposition, designate the letters in the original plaintext message by the numbers designating their position. Thus, with 28 letters in the message, the original sequence of letters is

01 02 03 04 05 06 07 08 09 10 11 12 13 14

15 16 17 18 19 20 21 22 23 24 25 26 27 28

After the first transposition we have

03 10 17 24 04 11 18 25 02 09 16 23 01 08

15 22 05 12 19 26 06 13 20 27 07 14 21 28

which has a somewhat regular structure. But after the second transposition, we have

17 09 05 27 24 16 12 07 10 02 22 20 03 25

15 13 04 23 19 14 11 01 26 21 18 08 06 28

This is a much less structured permutation and is much more difficult to cryptanalyze.

Rotor Machines

The example just given suggests that multiple stages of encryption can produce an algorithm that is significantly more difficult to cryptanalyze. This is as true of substitution ciphers as it is of transposition ciphers. Before the introduction of DES, the most important application of the principle of multiple stages of encryption was a class of systems known as rotor machines.

The basic principle of the rotor machine is illustrated in Figure 2.7. The machine consists of a set of independently rotating cylinders through which electrical pulses can flow. Each cylinder has 26 input pins and 26 output pins, with internal wiring that connects each input pin to a unique output pin. For simplicity, only three of the internal connections in each cylinder are shown.

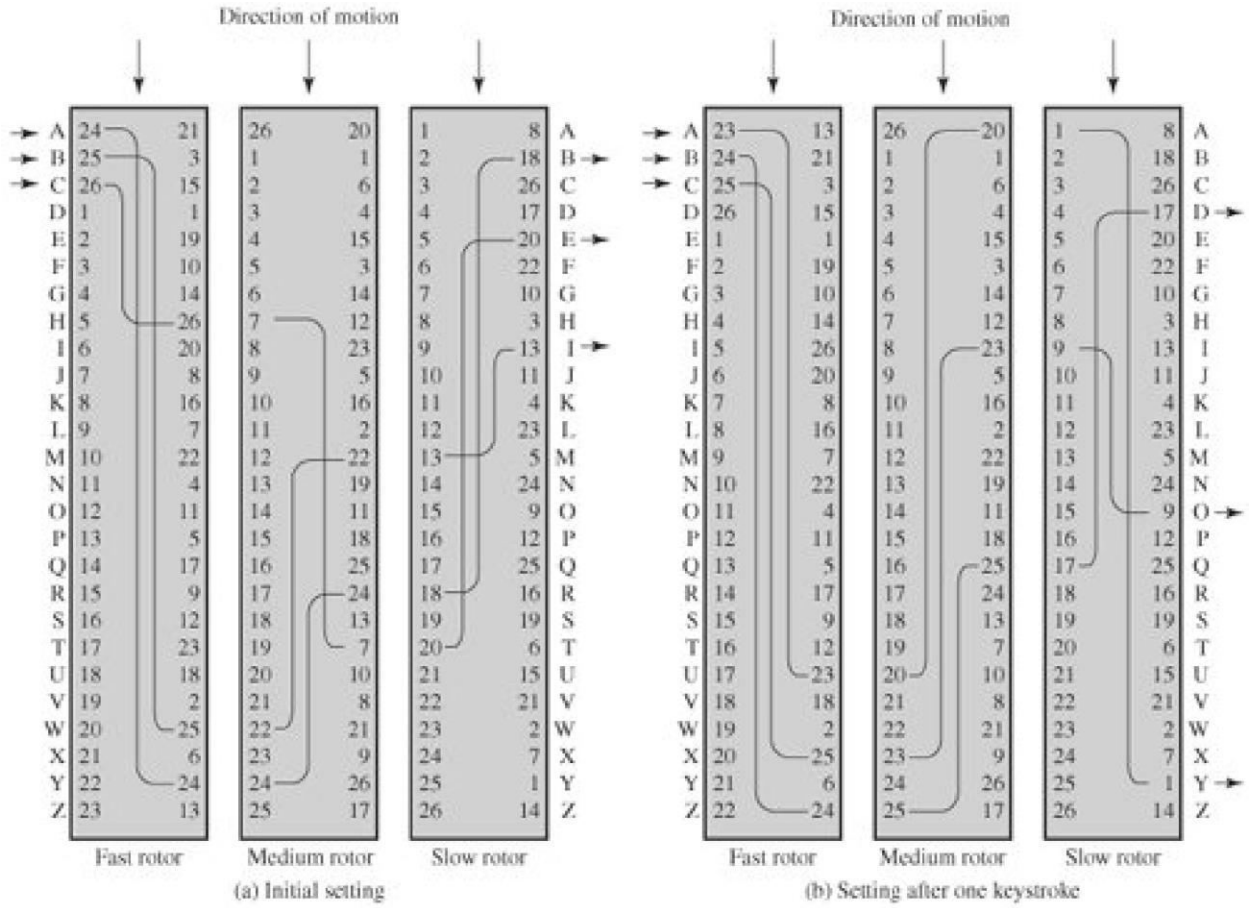


Figure 2.7. Three-Rotor Machine with Wiring Represented by Numbered Contacts

If we associate each input and output pin with a letter of the alphabet, then a single cylinder defines a monoalphabetic substitution. For example, in Figure 2.7, if an operator depresses the key for the letter A, an electric signal is applied to the first pin of the first cylinder and flows through the internal connection to the twenty-fifth output pin. Consider a machine with a single cylinder. After each input key is depressed, the cylinder rotates one position, so that the internal connections are shifted accordingly. Thus, a different monoalphabetic substitution cipher is defined. After 26 letters of plaintext, the cylinder would be back to the initial position. Thus, we have a polyalphabetic substitution algorithm with a period of 26. A single-cylinder system is trivial and does not present a formidable cryptanalytic task. The power of the rotor machine is in the use of multiple cylinders, in which the output pins of one cylinder are connected to the input pins of the next. Figure 2.7 shows a three-cylinder system. The left half of the figure shows a position in which the input from the operator to the first pin (plaintext letter a) is routed through the three cylinders to appear at the output of the second pin (ciphertext letter B).

With multiple cylinders, the one closest to the operator input rotates one pin position with each keystroke. The right half of Figure 2.7 shows the system's configuration after a single keystroke. For every complete rotation of the inner cylinder, the middle cylinder rotates one pin position. Finally, for every complete rotation of the middle cylinder, the outer cylinder rotates one pin position. This is the same type of operation seen with an odometer. The result is that there are 26 x 26 x 26 = 17,576 different substitution alphabets used before the system repeats. The addition of fourth and fifth rotors results in periods of 456,976 and 11,881,376 letters, respectively. As David Kahn eloquently put it, referring to a five-rotor machine [KAHN96, page 413]:

A period of that length thwarts any practical possibility of a straightforward solution on the basis of letter frequency. This general solution would need about 50 letters per cipher alphabet, meaning that all five rotors would have to go through their combined cycle 50 times. The ciphertext would have to be as long as all the speeches made on the floor of the Senate and the House of Representatives in three successive sessions of Congress. No cryptanalyst is likely to bag that kind of trophy in his lifetime; even diplomats, who can be as verbose as politicians, rarely scale those heights of loquacity.

Steganography

We conclude with a discussion of a technique that is, strictly speaking, not encryption, namely, steganography. A plaintext message may be hidden in one of two ways. The methods of steganography conceal the existence of the message, whereas the methods of cryptography render the message unintelligible to outsiders by various transformations of the text.

Steganography was an obsolete word that was revived by David Kahn. A simple form of steganography, but one that is time-consuming to construct, is one in which an arrangement of words or letters within an apparently innocuous text spells out the real message. For example, the sequence of first letters of each word of the overall message spells out the hidden message.

Various other techniques have been used historically; some examples are the following

Character marking: Selected letters of printed or typewritten text are overwritten in pencil. The marks are ordinarily not visible unless the paper is held at an angle to bright light.

Invisible ink: A number of substances can be used for writing but leave no visible trace until heat or some chemical is applied to the paper.

Pin punctures: Small pin punctures on selected letters are ordinarily not visible unless the paper is held up in front of a light.

Typewriter correction ribbon: Used between lines typed with a black ribbon, the results of typing with the correction tape are visible only under a strong light.

Although these techniques may seem archaic, they have contemporary equivalents. [WAYN93] proposes hiding a message by using the least significant bits of frames on a CD. For example, the Kodak Photo CD format's maximum resolution is 2048 by 3072 pixels, with each pixel containing 24 bits of RGB color information. The least significant bit of each 24-bit pixel can be changed without greatly affecting the quality of the image. The result is that you can hide a 2.3-megabyte message in a single digital snapshot. There are now a number of software packages available that take this type of approach to steganography. Steganography has a number of drawbacks when compared to encryption. It requires a lot of overhead to hide a relatively few bits of information, although using some scheme like that proposed in the preceding paragraph may make it more effective. Also, once the system is discovered, it becomes virtually worthless. This problem, too, can be overcome if the insertion method depends on some sort of key (e.g., see Problem 2.11). Alternatively, a message can be first encrypted and then hidden using steganography. The advantage of steganography is that it can be employed by parties who have something to lose should the fact of their secret communication (not necessarily the content) be discovered. Encryption flags traffic as important or secret or may identify the sender or receiver as someone with something to hide.

Block Ciphers and the Data Encryption Standard

Block Cipher Principles

Most symmetric block encryption algorithms in current use are based on a structure referred to as a Feistel block cipher. For that reason, it is important to examine the design principles of the Feistel cipher. We begin with a comparison of stream ciphers and block ciphers. Then we discuss the motivation for the Feistel block cipher structure. Finally, we discuss some of its implications.

Stream Ciphers and Block Ciphers

A **stream cipher** is one that encrypts a digital data stream one bit or one byte at a time. Examples of classical stream ciphers are the autokeyed Vigenère cipher and the Vernam cipher.

A **block cipher** is one in which a block of plaintext is treated as a whole and used to produce a ciphertext block of equal length. Typically, a block size of 64 or 128 bits is used. Using some of the modes of operation, a block cipher can be used to achieve the same effect as a stream cipher. Far more effort has gone into analyzing block ciphers. In general, they seem applicable to a broader range of applications than stream ciphers. The vast majority of network-based symmetric cryptographic applications make use of block ciphers.

The Feistel Cipher

Feistel proposed that we can approximate the ideal block cipher by utilizing the concept of a product cipher, which is the execution of two or more simple ciphers in sequence in such a way that the final result or product is cryptographically stronger than any of the component ciphers. The essence of the approach is to develop a block cipher with a key length of k bits and a block length of n bits, allowing a total of 2^k possible transformations, rather than the $2^n!$ transformations available with the ideal block cipher. In particular, Feistel proposed the use of a cipher that alternates substitutions and permutations. In fact, this is a practical application a proposal by Claude Shannon to develop a product cipher that alternates *confusion* and *diffusion* functions. We look next at these concepts of diffusion and confusion and then present the Feistel cipher.

Diffusion and Confusion

The terms *diffusion* and *confusion* were introduced by Claude Shannon to capture the two basic building blocks for any cryptographic system.Shannon's concern was to thwart cryptanalysis based on statistical analysis. The reasoning is as follows. Assume the attacker has some knowledge of the statistical characteristics of the plaintext. For example, in a human-readable message in some language, the frequency distribution of the various letters may be known. Or there may be words or phrases likely to appear in the message (probable words). If these statistics are in any way reflected in the ciphertext, the cryptanalyst may be able to deduce the encryption key, or part of the key, or at least a set of keys likely to contain the exact key.

Other than recourse to ideal systems, Shannon suggests two methods for frustrating statistical cryptanalysis: diffusion and confusion. In **diffusion**, the statistical structure of the plaintext is

dissipated into long-range statistics of the ciphertext. This is achieved by having each plaintext digit affect the value of many ciphertext digits; generally this is equivalent to having each ciphertext digit be affected by many plaintext digits. An example of diffusion is to encrypt a message $M = m1, m2, m3,...$ of characters with an averaging operation: adding k successive letters to get a ciphertext letter yn. One can show that the statistical structure of the plaintext has been dissipated. Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of cipher text.

$$y_n = \left(\sum_{i=1}^{k} m_{n+i} \right) \bmod 26$$

adding k successive letters to get a ciphertext letter yn. One can show that the statistical structure of the plaintext has been dissipated.

Thus, the letter frequencies in the ciphertext will be more nearly equal than in the plaintext; the digram frequencies will also be more nearly equal, and so on. In a binary block cipher, diffusion can be achieved by repeatedly performing some permutation on the data followed by applying a function to that permutation; the effect is that bits from different positions in the original plaintext contribute to a single bit of cipher text Every block cipher involves a transformation of a block of plaintext into a block of ciphertext, where the transformation depends on the key. The mechanism of diffusion seeks to make the statistical relationship between the plaintext and ciphertext as complex as possible in order to thwart attempts to deduce the key. On the other hand, **confusion** seeks to make the relationship between the statistics of the ciphertext and the value of the encryption key as complex as possible, again to thwart attempts to discover the key. Thus, even if the attacker can get some handle on the statistics of the ciphertext, the way in which the key was used to produce that ciphertext is so complex as to make it difficult to deduce the key. This is achieved by the use of a complex substitution algorithm. In contrast, a simple linear substitution function would add little confusion.

Feistel Cipher Structure

Figure 3.2 depicts the structure proposed by Feistel. The inputs to the encryption algorithm are a plaintext block of length 2*w* bits and a key *K*. The plaintext block is divided into two halves, *L*0 and *R*0. The two halves of the data pass through *n* rounds of processing and then combine to produce the ciphertext block. Each round *i* has as inputs *Li-1* and *Ri-1*, derived from the previous round, as well as a subkey *Ki*, derived from the overall *K*. In general, the subkeys *Ki* are different from *K* and from each other.

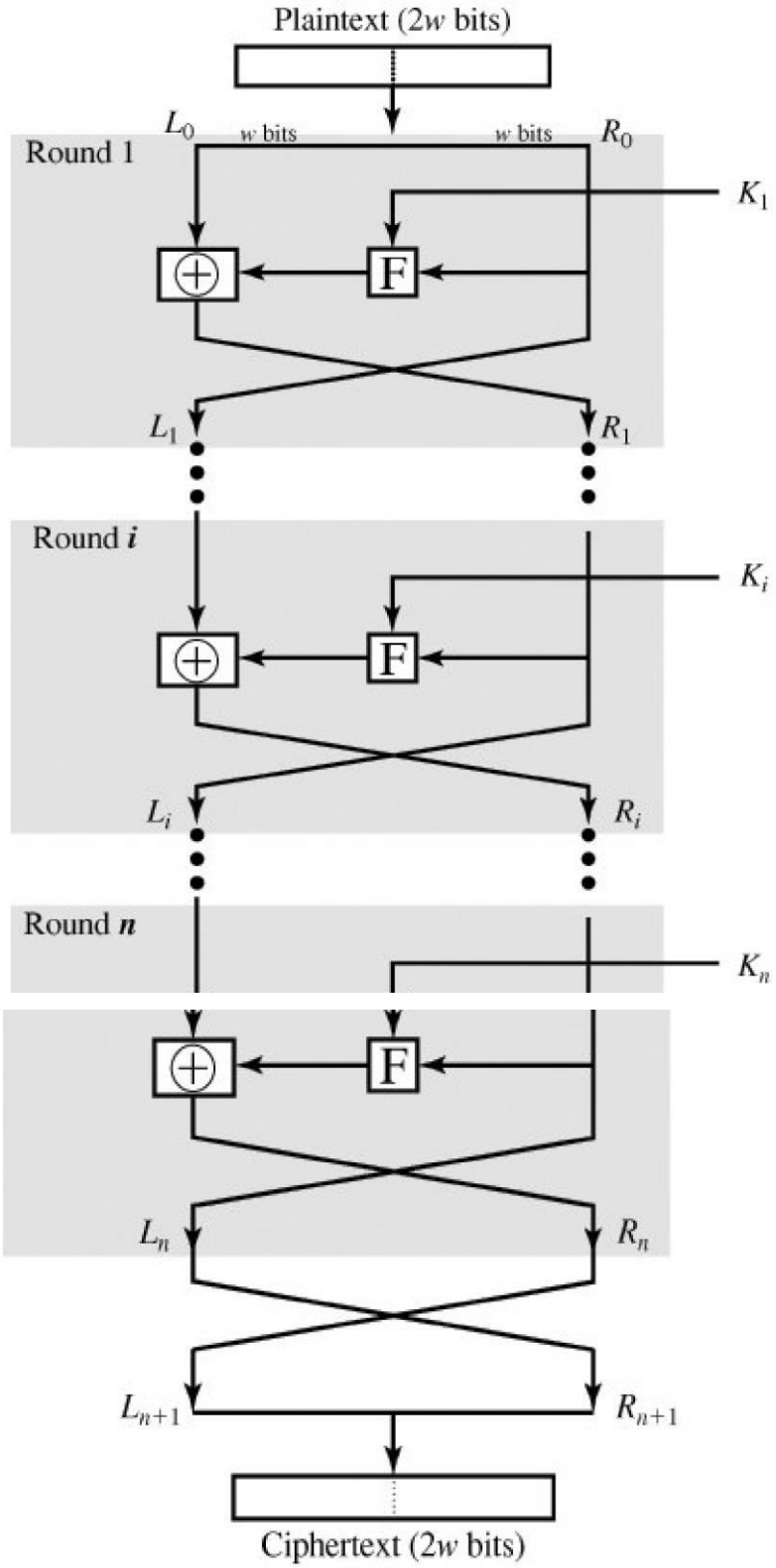


Figure 3.2. Classical Feistel Network

The exact realization of a Feistel network depends on the choice of the following parameters and design features:

Block size: Larger block sizes mean greater security (all other things being equal) but reduced encryption/decryption speed for a given algorithm. The greater security is achieved by greater diffusion Traditionally, a block size of 64 bits has been considered a reasonable tradeoff and was nearly universal in block cipher design. However, the new AES uses a 128-bit block size.

Key size: Larger key size means greater security but may decrease encryption/decryption speed. The greater security is achieved by greater resistance to brute-force attacks and greater confusion. Key sizes of 64 bits or less are now widely considered to be inadequate, and 128 bits has become a common size.

Number of rounds: The essence of the Feistel cipher is that a single round offers inadequate security but that multiple rounds offer increasing security. A typical size is 16 rounds.

Subkey generation algorithm: Greater complexity in this algorithm should lead to greater difficulty of cryptanalysis.

Round function: Again, greater complexity generally means greater resistance to cryptanalysis.

There are two other considerations in the design of a Feistel cipher:

Fast software encryption/decryption: In many cases, encryption is embedded in applications or utility functions in such a way as to preclude a hardware implementation. Accordingly, the speed of execution of the algorithm becomes a concern.

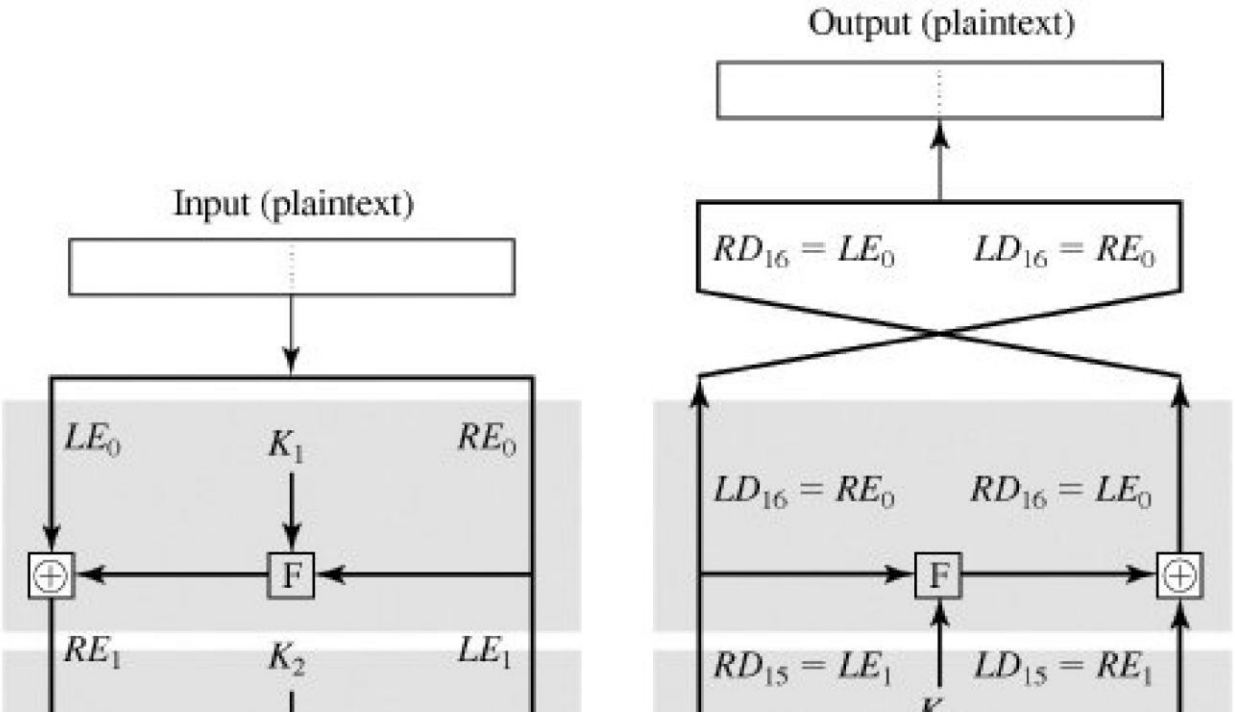
Ease of analysis: Although we would like to make our algorithm as difficult as possible to cryptanalyze, there is great benefit in making the algorithm easy to analyze. That is, if the algorithm can be concisely and clearly explained, it is easier to analyze that algorithm for cryptanalytic vulnerabilities and therefore develop a higher level of assurance as to its strength. DES, for example, does not have an easily analyzed functionality.

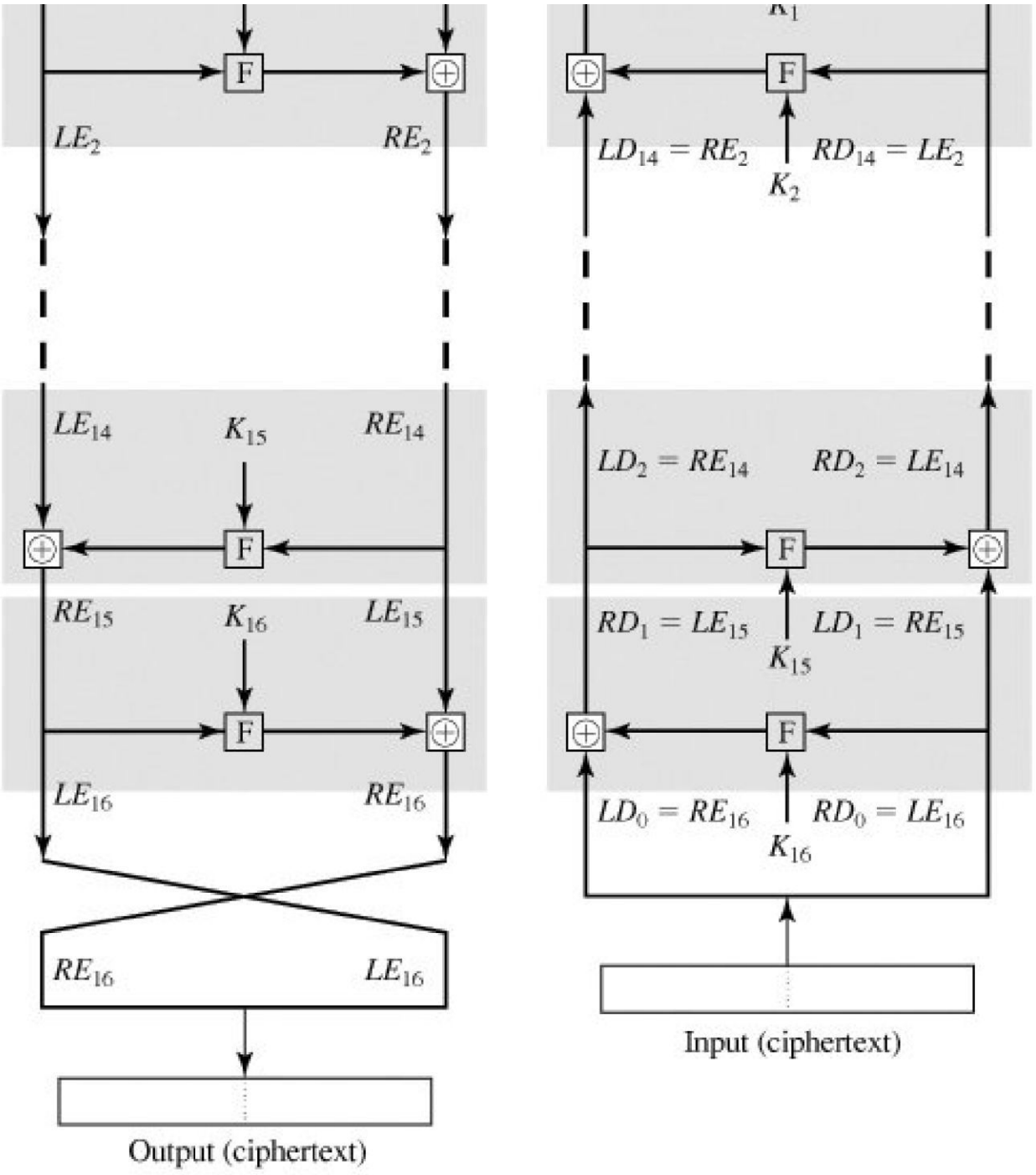
Feistel Decryption Algorithm

The process of decryption with a Feistel cipher is essentially the same as the encryption process. The rule is as follows: Use the cipher text as input to the algorithm, but use the subkeys *Ki* in reverse order. That is, use *Kn* in the first round, *Kn-1* in the second round, and so on until *K*1 is used in the last round. This is a nice feature because it means we need not implement two different algorithms, one for encryption and one for decryption. To see that the same algorithm with a reversed key order produces the correct result, consider Figure 3.3, which shows the

encryption process going down the left-hand side and the decryption process going up the right-hand side for a 16-round algorithm (the result would be the same for any number of rounds). For clarity, we use the notation LE*i* and RE*i* for data traveling through the encryption algorithm and *LDi* and RD*i* for data traveling through the decryption algorithm. The diagram indicates that, at every round, the intermediate value of the decryption process is equal to the corresponding value of the encryption process with the two halves of the value swapped. To put this another way, let the output of the *i*th encryption round be LE*i*||RE*i* (*Li* concatenated with *Ri*). Then the corresponding input to the (16*i*)th decryption round is RE*i*||LE*i* or, equivalently, RD16-*i*||*LD*16-*i*.

Figure 3.3. Feistel Encryption and Decryption





After the last iteration of the encryption process,

the two halves of the output are swapped, so that the ciphertext is RE16||LE16. The output of that round is the ciphertext. Now take that ciphertext and use it as input to the same algorithm. The input to the first round is RE16||LE16, which is equal to the 32-bit swap of the output of the sixteenth round of the encryption process.

Now we would like to show that the output of the first round of the decryption process is equal to a 32-bit swap of the input to the sixteenth round of the encryption process. First, consider the encryption process. We see that

LE16 = RE15

RE16 = LE15 x F(RE15, *K*16)

On the decryption side,

*LD*1 = RD0 = LE16 = RE15

RD1 = LD0 x F(RD0, *K*16)

= RE16 x F(RE15, *K*16)

= [LE15 x F(RE15, *K*16)] x F(RE15, *K*16)

The XOR has the following properties:

[*A* x *B*] x *C* = *A* x [*B* x *C*]

D x *D* = 0

E x 0 = E

Thus, we have *LD*1 = RE15 and RD1 = LE15. Therefore, the output of the first round of the decryption process is LE15||RE15, which is the 32-bit swap of the input to the sixteenth round of the encryption. This correspondence holds all the way through the 16 iterations, as is easily shown. We can cast this process in general terms. For the *i*th iteration of the encryption algorithm,

LE*i* = RE*i-1*

RE*i* =LE*i*-1 x F(RE*i*-1, *Ki*)

Rearranging terms,

RE*i*-1 = LE*i*

LE*i*-1 = RE*i* x F(RE*i*-1, *Ki*2 = RE*i* x F(LE*i*, *Ki*)

The Data Encryption Standard

The most widely used encryption scheme is based on the Data Encryption Standard (DES) adopted in 1977 by the National Bureau of Standards, now the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). The algorithm itself is referred to as the Data Encryption Algorithm (DEA).

For DES, data are encrypted in 64-bit blocks using a 56-bit key. The algorithm transforms 64-bit input in a series of steps into a 64-bit output. The same steps, with the same key, are used to reverse the encryption.

DES Encryption

The overall scheme for DES encryption is illustrated in Figure 3.4. As with any encryption scheme, there are two inputs to the encryption function: the plaintext to be encrypted and the key. In this case, the plaintext must be 64 bits in length and the key is 56 bits in length.

Looking at the left-hand side of the figure, we can see that the processing of the plaintext proceeds in three phases. First, the 64-bit plaintext passes through an initial permutation (IP) that rearranges the bits to produce the *permuted input*. This is followed by a phase consisting of 16 rounds of the same function, which involves both permutation and substitution functions. The output of the last (sixteenth) round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the **preoutput**. Finally, the preoutput is passed through a permutation (IP-1) that is the inverse of the initial permutation function, to produce the 64-bit ciphertext. With the exception of the initial and final permutations, DES has the exact structure of a Feistel cipher, as shown in Figure 3.2.

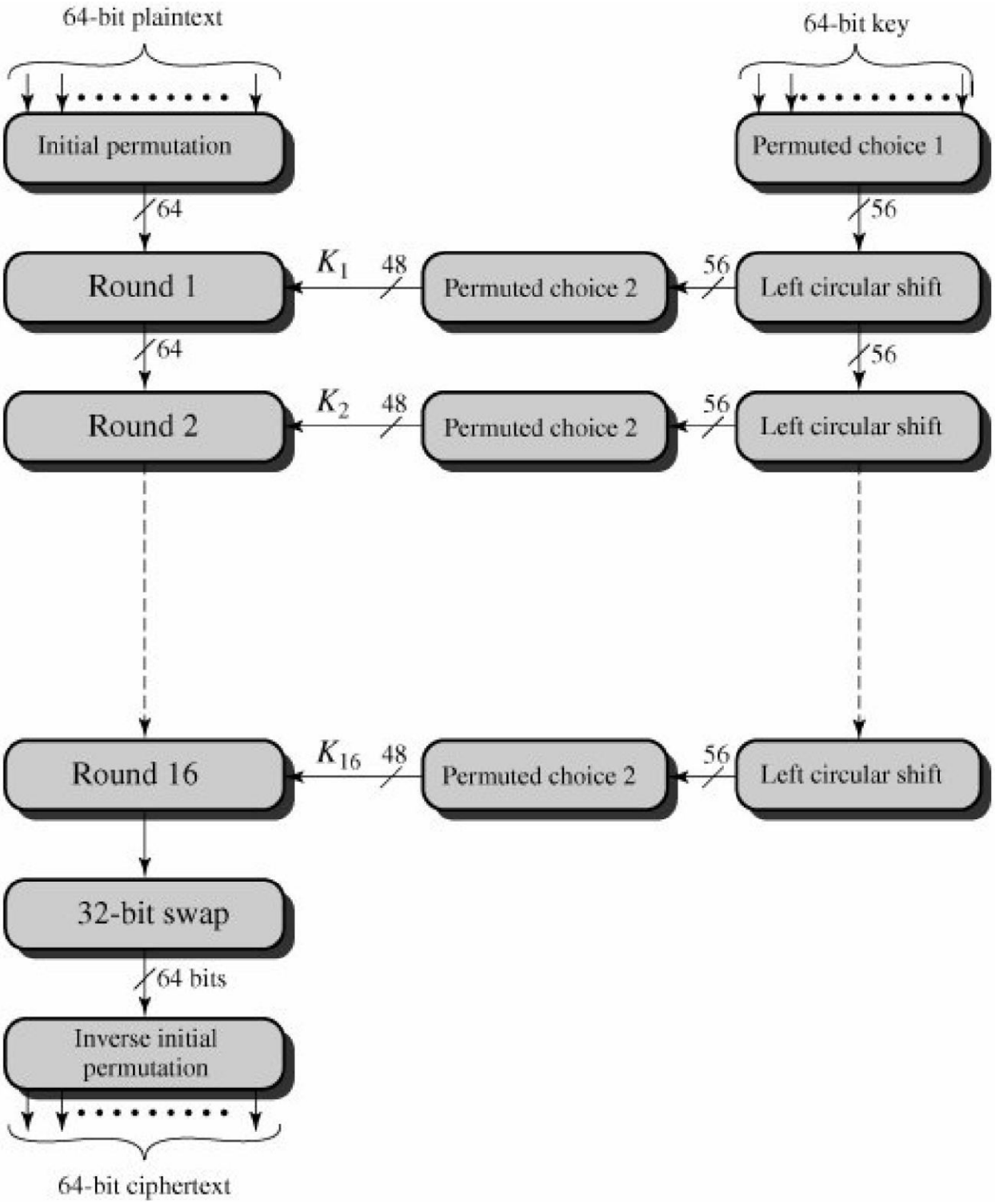


Figure 3.4. General Depiction of DES Encryption Algorithm

Details of Single Round

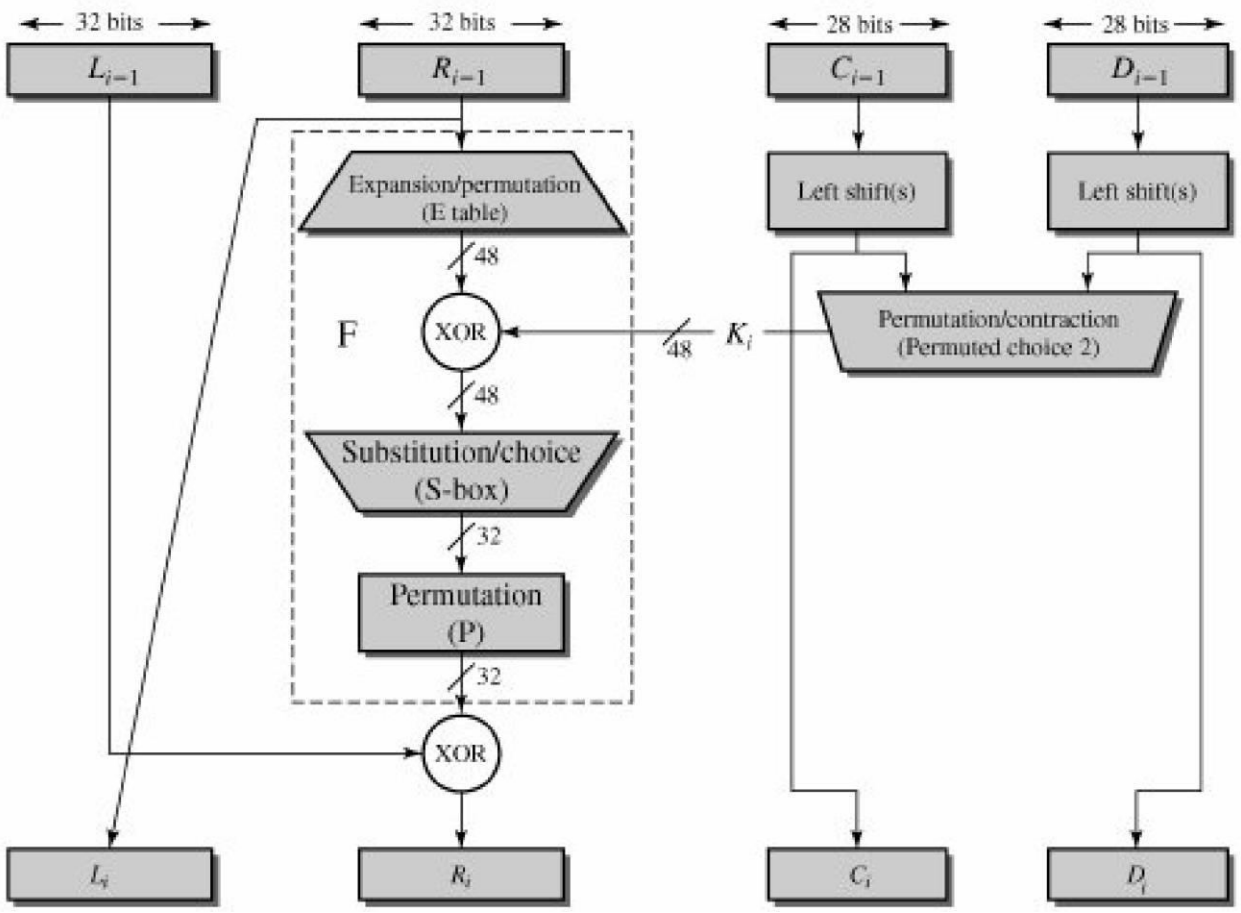
Figure 3.5 shows the internal structure of a single round. Again, begin by focusing on the left-hand side of the diagram. The left and right halves of each 64-bit intermediate value are treated as separate 32-bit quantities, labeled L (left) and R (right). As in any classic Feistel cipher, the overall processing at each round can be summarized in the following formulas:

$L_i = R_{i-1}$

$R_i = L_{i-1} \times F(R_{i-1}, K_i)$

[Page 77]

Figure 3.5. Single Round of DES Algorithm



The round key *Ki* is 48 bits. The *R* input is 32 bits. This *R* input is first expanded to 48 bits by using a table that defines a permutation plus an expansion that involves duplication of 16 of the *R* bits (Table 3.2c). The resulting 48 bits are XORed with *Ki*. This 48-bit result passes through a substitution function that produces a 32-bit output, which is permuted as defined by Table 3.2d.

The role of the S-boxes in the function F is illustrated in Figure 3.6. The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. These transformations are defined in Table 3.3, which is interpreted as follows: The first and last bits of the input to box S*i* form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for S*i*. The middle four bits select one of the sixteen columns. The decimal value in the cell selected by the row and column is then converted to its 4-bit representation to produce the output. For example, in S1 for input 011001, the row is 01 (row 1) and the column is 1100 (column 12). The value in row 1, column 12 is 9, so the output is 1001.

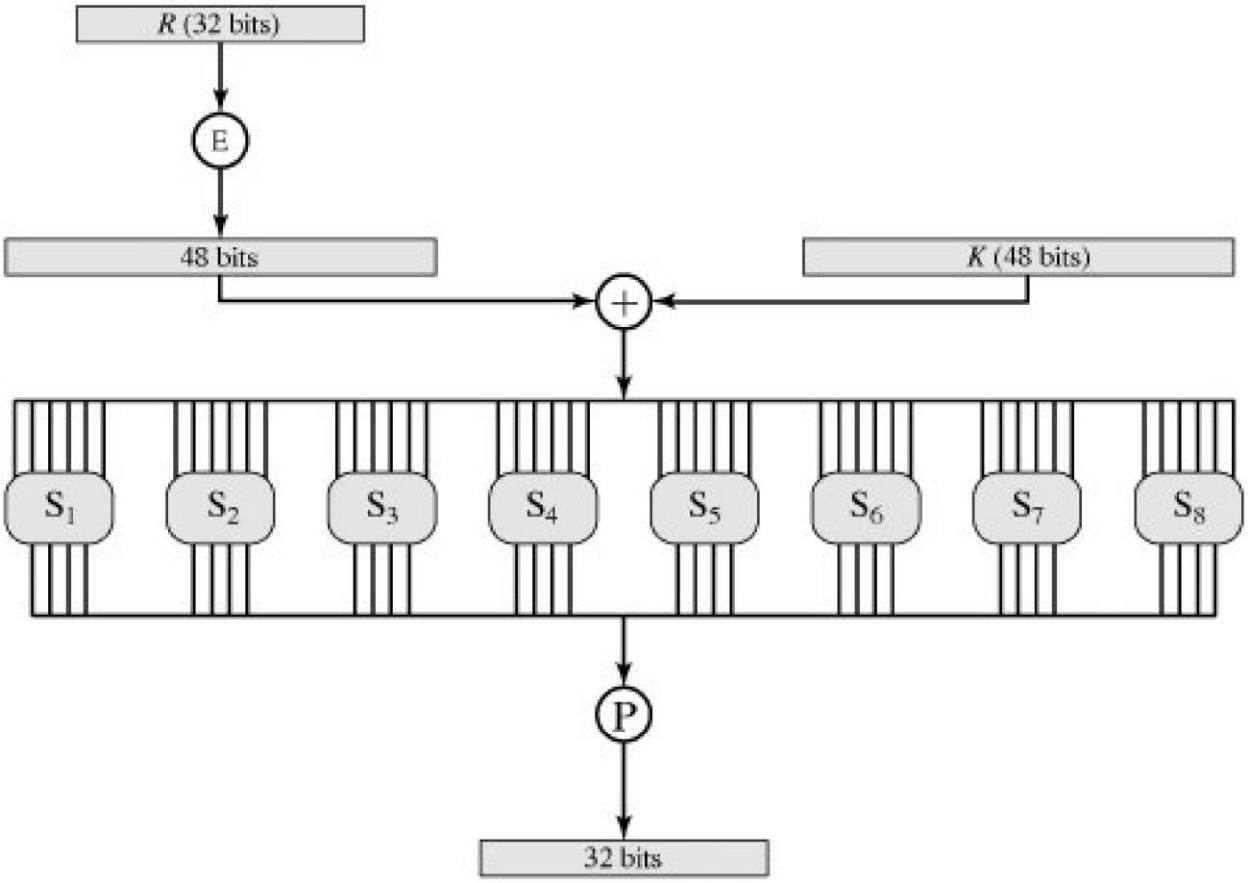


Figure 3.6. Calculation of F(R, K)

S_1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Table 3.3. Definition of DES S-Boxes

Key Generation

we see that a 64-bit key is used as input to the algorithm. The bits of the key are numbered from 1 through 64; every eighth bit is ignored.The key is first subjected to a permutation governed by a table labeled Permuted Choice One. The resulting 56-bit key is then treated as two 28-bit quantities, labeled*C* 0 and *D*0. At each round, *Ci*-1 and *Di*-1 are separately subjected to a circular left shift, or rotation, of 1 or 2 bits. These shifted values serve as input to the next round. They also serve as input to Permuted Choice Two, which produces a 48-bit output that serves as input to the function F(*Ri*-1, *Ki*).

The Strength of DES

Since its adoption as a federal standard, there have been lingering concerns about the level of security provided by DES. These concerns, by and large, fall into two areas: key size and the nature of the algorithm.

The Use of 56-Bit Keys

With a key length of 56 bits, there are 2 56 possible keys, which is approximately 7.2 x 10 16Thus, on the face of it, a brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years (see Table 2.2) to break the cipher. however, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond [DIFF77]. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about $20 million in 1977 dollars. It is important to note that there is more to a key-search attack than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext. If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated. If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate. Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed. The EFF approach addresses this issue as well and introduces some

automated techniques that would be effective in many contexts. Fortunately, there are a number of alternatives to DES, the most important of which are AES and triple DES, discussed in Chapters 5 and 6, respectively.

The Nature of the DES Algorithm

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviors of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

Timing Attacks

We discuss timing attacks in more detail in Part Two, as they relate to public-key algorithms. However, the issue may also be relevant for symmetric ciphers. In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various ciphertexts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs on an approach that yields the Hamming weight (number of bits equal to one) of the secret key. This is a long way from knowing the actual key, but it is an intriguing first step. The authors conclude that DES appears to be fairly resistant to a successful timing attack but suggest some avenues to explore. Although this is an interesting line of attack, it so far appears unlikely that this technique will ever be successful against DES or more powerful symmetric ciphers such as triple DES and AES.

DES Design Criteria

The criteria used in the design of DES,focused on the design of the S-boxes and on the P function that takes

the output of the S boxes (Figure 3.6). The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which

this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.

2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.

3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.

4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.

5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.

6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.

7. This is a criterion similar to the previous one, but for the case of three S-boxes. Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES.

If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties. The criteria for the permutation P are as follows:

1. The four output bits from each S-box at round i are distributed so that two of them affect (provide input for) "middle bits" of round $(i + 1)$ and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes. The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.

2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.

3. For two S-boxes j, k, if an output bit from Sj affects a middle bit of Sk on the next round, then an output bit from Sk cannot affect a middle bit of Sj. This implies that for $j = k$, an output bit from Sj must not affect a middle bit of Sj.

These criteria are intended to increase the diffusion of the algorithm.

Number of Rounds

The cryptographic strength of a Feistel cipher derives from three aspects of the design: the number of rounds, the function F, and the key schedule algorithm. Let us look first at the choice of the number of rounds. The greater the number of rounds, the more difficult it is to perform cryptanalysis, even for a relatively weak F. In general, the criterion should be that the number of rounds is chosen so that known cryptanalytic efforts require greater effort than a simple brute-force key search attack. This criterion was certainly used in the design of DES. Schneier observes that for 16-round DES, a differential cryptanalysis attack is slightly less efficient than brute force: the differential cryptanalysis attack requires perations,whereasbrute force requires 2^ 55. If DES had 15 or fewer rounds, differential cryptanalysis would require less effort than brute-force key search.

Design of Function F

The heart of a Feistel block cipher is the function F. As we have seen, in DES, this function relies on the use of S-boxes. This is also the case for most other symmetric block ciphers, as we shall see in Chapter 4. However, we can make some general comments about the criteria for designing F. After that, we look specifically at S-box design.

Design Criteria for F

The function F provides the element of confusion in a Feistel cipher. Thus, it must be difficult to "unscramble" the substitution performed by F. One obvious criterion is that F be nonlinear, as we discussed previously. The more nonlinear F, the more difficult any type of cryptanalysis will be. There are several measures of nonlinearity, which are beyond the scope of this book. In rough terms, the more difficult it is to approximate F by a set of linear equations, the more nonlinear F is.

S-Box Design

One of the most intense areas of research in the field of symmetric block ciphers is that of S-box design. The papers are almost too numerous to count. Here we mention some general principles. In essence, we would like any change to the input vector to an S-box to result in random-looking changes to the output. The relationship should be nonlinear and difficult to approximate with linear functions. Obvious characteristic of the S-box is its size. An n x m S-box has n input bits and m output bits. DES has 6 x 4 S-boxes. Blowfish, has 8 x 32 S-boxes. Larger S-boxes, by and

large, are more resistant to differential and linear cryptanalysis. On the other hand, the larger the dimension n , the (exponentially) larger the lookup table. Thus, for practical reasons, a limit on f Equal to about 8 to 10 is usually imposed. Another practical consideration is that the larger the S-box, the more difficult it is to design it properly. S-boxes are typically organized in a different manner than used in DES. An n x m S-box typically consists of 2n rows of m bits each. The n bits of input select one of the rows of the S-box, and the m bits in that row are the output. For example, in an 8 x 32 S-box, if the input is 00001001, the output consists of the 32 bits in row 9 (the first row is labeled row 0).

Differential and Linear Cryptanalysis

For most of its life, the prime concern with DES has been its vulnerability to brute-force attack because of its relatively short (56 bits) key length. However, there has also been interest in finding cryptanalytic attacks on DES. With the increasing popularity of block ciphers with longer key lengths, including triple DES, brute-force attacks have become increasingly impractical. Thus, there has been increased emphasis on cryptanalytic attacks on DES and other symmetric block ciphers. In this section, we provide a brief overview of the two most powerful and promising approaches: differential cryptanalysis and linear cryptanalysis.

Differential Cryptanalysis

One of the most significant advances in cryptanalysis in recent years is differential cryptanalysis. In this section, we discuss the technique and its applicability to DES.

History

Although differential cryptanalysis is a powerful tool, it does not do very well against DES. The reason, according to a member of the IBM team that designed DES [COPP94], is that differential cryptanalysis was known to the team as early as 1974. The need to strengthen DES against attacks using differential cryptanalysis played a large part in the design of the S-boxes and the permutation P. As evidence of the impact of these changes, consider these comparable results reported in [BIHA93]. Differential cryptanalysis of an eight-round LUCIFER algorithm requires

only 256 chosen plaintexts, whereas an attack on an eight-round version of DES requires 2 14 chosen plaintexts.

Differential Cryptanalysis Attack

The differential cryptanalysis attack is complex; [BIHA93] provides a complete description. The rationale behind differential cryptanalysis is to observe the behavior of pairs of text blocks evolving along each round of the cipher, instead of observing the evolution of a single text block. Here, we provide a brief overview so that you can get the flavor of the attack.

We begin with a change in notation for DES. Consider the original plaintext block *m* to consist of two halves *m*0,*m*1. Each round of DES maps the right-hand input into the left-hand output and sets the right-hand output to be a function of the left-hand input and the subkey for this round. So, at each round, only one new 32-bit block is created. If we label each new block *m*1 17), then the intermediate message halves are related as follows:

mi+1 = *mi*-1 Xor f(*mi*, *Ki*), *i* = 1, 2, ..., 16

In differential cryptanalysis, we start with two messages, *m* and *m*', with a known XOR difference D*m* = *m* Xor *m*', and consider the difference between the intermediate message halves: *mi* = *mi* Xor *mi*' Then we have:

$$\begin{aligned}\Delta m_{i+1} &= m_{i+1} \oplus m'_{i+1} \\ &= [m_{i-1} \oplus \mathrm{f}(m_i, K_i)] \oplus [m'_{i-1} \oplus \mathrm{f}(m'_i, K_i)] \\ &= \Delta m_{i-1} \oplus [\mathrm{f}(m_i, K_i) \oplus \mathrm{f}(m'_i, K_i)]\end{aligned}$$

Now, suppose that many pairs of inputs to f with the same difference yield the same output difference if the same subkey is used. To put this more precisely, let us say that *X may cause Y with probability p*, if for a fraction *p* of the pairs in which the input XOR is *X*, the output XOR equals *Y*. We want to suppose that there are a number of values of*X* that have high probability of causing a particular output difference. Therefore, if we know D*mi*-1 and D*mi* with high

probability, then we know D*mi*+1 with high probability. Furthermore, if a number of such differences are determined, it is feasible to determine the subkey used in the function f.

The overall strategy of differential cryptanalysis is based on these considerations for a single round. The procedure is to begin with two plaintext messages *m* and *m*' with a given difference and trace through a probable pattern of differences after each round to yield a probable difference for the ciphertext. Actually, there are two probable patterns of differences for the two 32-bit halves: (D*m*17||*m*16). Next, we submit *m* and *m*' for encryption to determine the actual difference under the unknown key and compare the result to the probable difference. If there is a match,

E(*K*, *m*)Xor E(*K*, *m*') = (D*m*17||*m*16)

then we suspect that all the probable patterns at all the intermediate rounds are correct. With that assumption, we can make some deductions about the key bits. This procedure must be repeated many times to determine all the key bits.

Linear Cryptanalysis

A more recent development is linear cryptanalysis, described in [MATS93]. This attack is based on finding linear approximations to describe the transformations performed in DES. This method can find a DES key given 2 43 known plaintexts, as compared to 2 47 chosen plaintexts for differential cryptanalysis. Although this is a minor improvement, because it may be easier to acquire known plaintext rather than chosen plaintext, it still leaves linear cryptanalysis infeasible as an attack on DES. So far, little work has been done by other groups to validate the linear cryptanalytic approach.

We now give a brief summary of the principle on which linear cryptanalysis is based. For a cipher with n-bit plaintext and ciphertext blocks

and an *m*-bit key, let the plaintext block be labeled P[1], ... P[*n*], the cipher text block C[1], ... C[*n*], and the key K[1], ... K[*m*]. Then define

A[*i, j, ..., k*] = *A*[*i*] xor *A*[*j*]xor ... *A*[*k*]

The objective of linear cryptanalysis is to find an effective *linear* equation of the form:

P[a1, a2, ..., aa] XOR C[b1, b2, ..., bb] = K[g1, g2, ..., gc]

(where *x* = 0 or 1; 1 ≤*a, b*≤ *n*, 1 ≤*c* ≤*m*, and where the a, b and g terms represent fixed, unique bit locations) that holds with probability *p* 0.5. The further *p* is from 0.5, the more effective the

equation. Once a proposed relation is determined, the procedure is to compute the results of the left-hand side of the preceding equation for a large number of plaintext-ciphertext pairs. If the result is 0 more than half the time, assume K[g1, g2, ..., gc] = 0. If it is 1 most of the time, assume Kg[1, g2, ..., gc] = 1. This gives us a linear equation on the key bits. Try to get more such relations so that we can solve for the key bits. Because we are dealing with linear equations, the problem can be approached one round of the cipher at a time, with the results combined.

Block Cipher Design Principles

Although much progress has been made in designing block ciphers that are cryptographically strong, the basic principles have not changed all that much since the work of Feistel and the DES design team in the early 1970s. It is useful to begin this discussion by looking at the published design criteria used in the DES effort. Then we look at three critical aspects of block cipher design: the number of rounds, design of the function F, and key scheduling.

DES Design Criteria

The criteria used in the design of DES, as reported in [COPP94], focused on the design of the S-boxes and on the P function that takes the output of the S boxes (Figure 3.6). The criteria for the S-boxes are as follows:

1. No output bit of any S-box should be too close a linear function of the input bits. Specifically, if we select any output bit and any subset of the six input bits, the fraction of inputs for which this output bit equals the XOR of these input bits should not be close to 0 or 1, but rather should be near 1/2.

2. Each row of an S-box (determined by a fixed value of the leftmost and rightmost input bits) should include all 16 possible output bit combinations.

3. If two inputs to an S-box differ in exactly one bit, the outputs must differ in at least two bits.

4. If two inputs to an S-box differ in the two middle bits exactly, the outputs must differ in at least two bits.

5. If two inputs to an S-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.

6. For any nonzero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibiting that difference may result in the same output difference.

7. This is a criterion similar to the previous one, but for the case of three S-boxes.

Coppersmith pointed out that the first criterion in the preceding list was needed because the S-boxes are the only nonlinear part of DES.

If the S-boxes were linear (i.e., each output bit is a linear combination of the input bits), the entire algorithm would be linear and easily broken. We have seen this phenomenon with the Hill cipher, which is linear. The remaining criteria were primarily aimed at thwarting differential cryptanalysis and at providing good confusion properties.

The criteria for the permutation P are as follows:

1. The four output bits from each S-box at round i are distributed so that two of them affect (provide input for) "middle bits" of round $(i + 1)$ and the other two affect end bits. The two middle bits of input to an S-box are not shared with adjacent S-boxes.

The end bits are the two left-hand bits and the two right-hand bits, which are shared with adjacent S-boxes.

2. The four output bits from each S-box affect six different S-boxes on the next round, and no two affect the same S-box.

3. For two S-boxes j, k, if an output bit from Sj affects a middle bit of Sk on the next round, then an output bit from Sk cannot affect a middle bit of Sj. This implies that for $j = k$, an output bit from Sj must not affect a middle bit of Sj.

These criteria are intended to increase the diffusion of the algorithm.

MODULE 2- Public Key Cryptography and RSA

Introduction to Public key Cryptography:

- Public key cryptography also called as **asymmetric cryptography**.
- It was invented by whitfield **Diffie** and Martin **Hellman** in 1976. Sometimes this cryptography also called as **Diffie-Helman Encryption**.
- Public key algorithms are based on mathematical problems which admit no efficient solution that are inherent in certain integer factorization, discrete logarithm and Elliptic curve relations.

Public key Cryptosystem Principles:

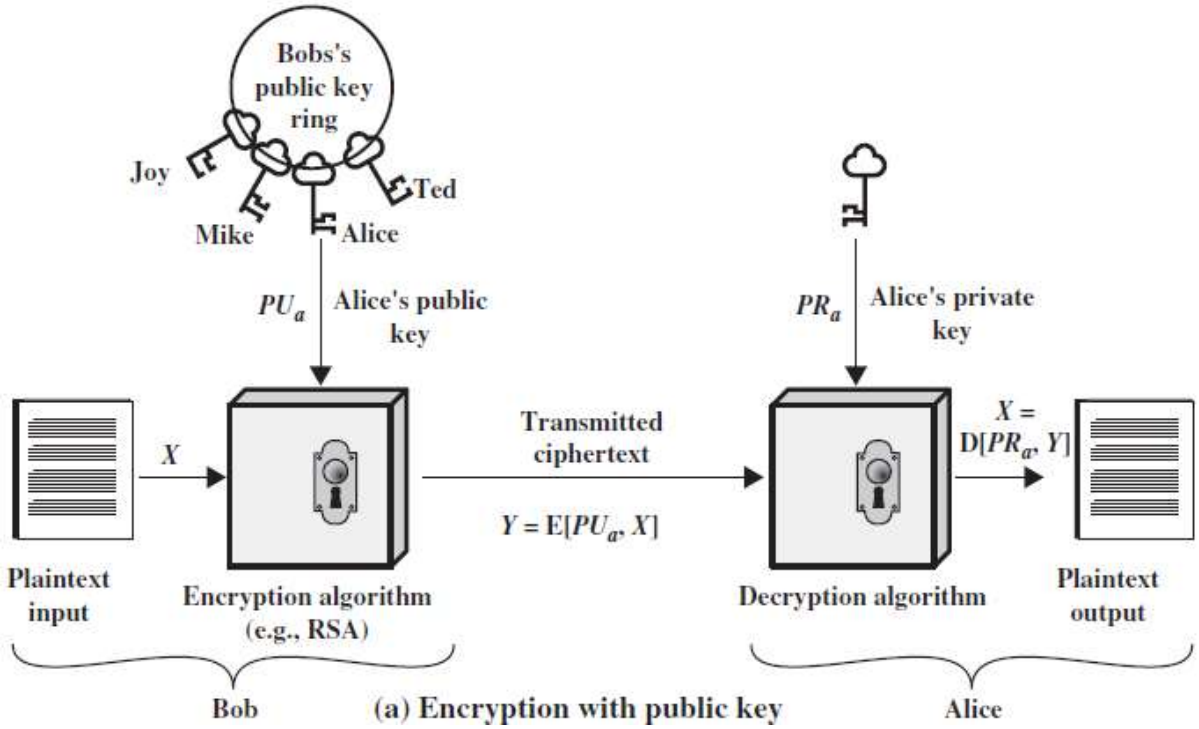
- The concept of public key cryptography in invented for two most difficult problems of Symmetric key encryption.
    - The Key Exchange Problem
    - The Trust Problem

    The Key Exchange Problem: The key exchange problem arises from the fact that communicating parties must somehow share a secret key before any secure communication can be initiated, and both parties must then ensure that the key remains secret. Of course, direct key exchange is not always feasible due to risk, inconvenience, and cost factors.

    The Trust Problem: Ensuring the integrity of received data and verifying the identity of the source of that data can be very important. Means in the symmetric key cryptography system, receiver doesn't know whether the message is coming for particular sender.
- This public key cryptosystem uses two keys as pair for encryption of plain text and Decryption of cipher text.
- These two keys are names as "**Public key**" and "**Private key**". The private key is kept secret where as public key is distributed widely.
- A message or text data which is encrypted with the public key can be decrypted only with the corresponding private-key
- This two key system very useful in the areas of confidentiality (secure) and authentication

A **public-key encryption** scheme has six ingredients		
1	**Plaintext**	This is the readable message or data that is fed into the algorithm as input.
2	**Encryption algorithm**	The encryption algorithm performs various transformations on the plaintext.
3	**Public key**	This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input
4	**Private key**	
5	**Ciphertext**	This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
6	**Decryption algorithm**	This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

Public key cryptography for providing confidentiality (secrecy)



The essential steps are the following.

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key.The companion key is kept private.As Figure 9.1a suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

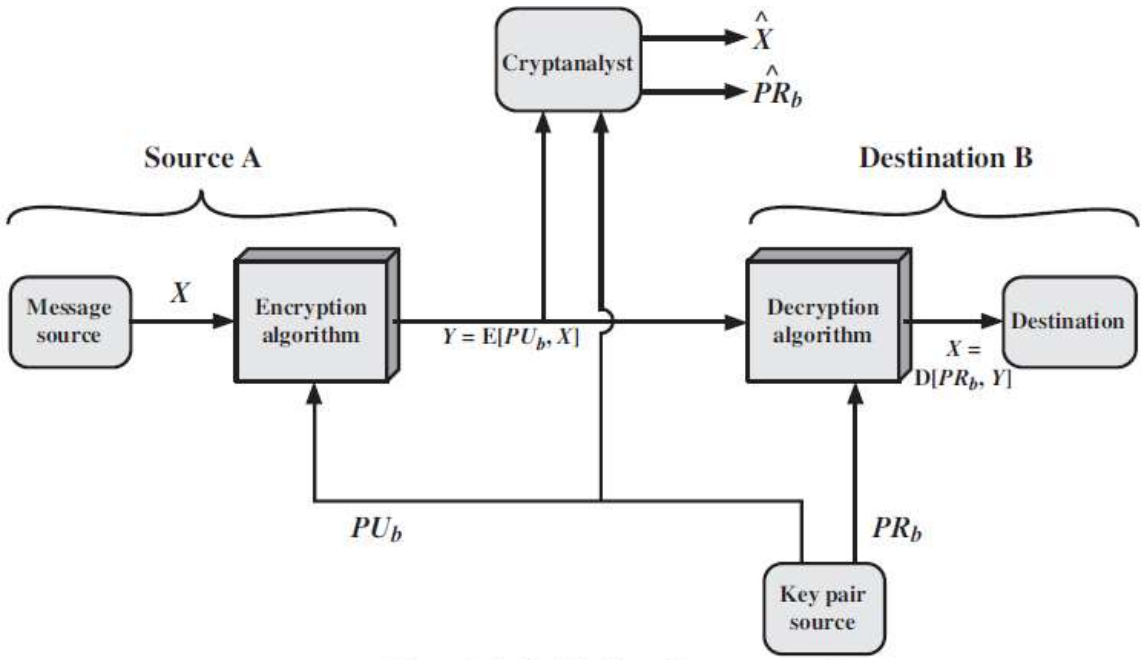


Figure 9.2 Public-Key Cryptosystem: Secrecy

There is some source A that produces a message in plaintext ***X* = [*X*1, *X*2, . . . ,*XM*].**

The ***M*** elements of ***X*** are letters in some finite alphabet. The message is intended for destination **B**.
B generates a related pair of keys: a public key, PU_b, and a private key, PR_b.

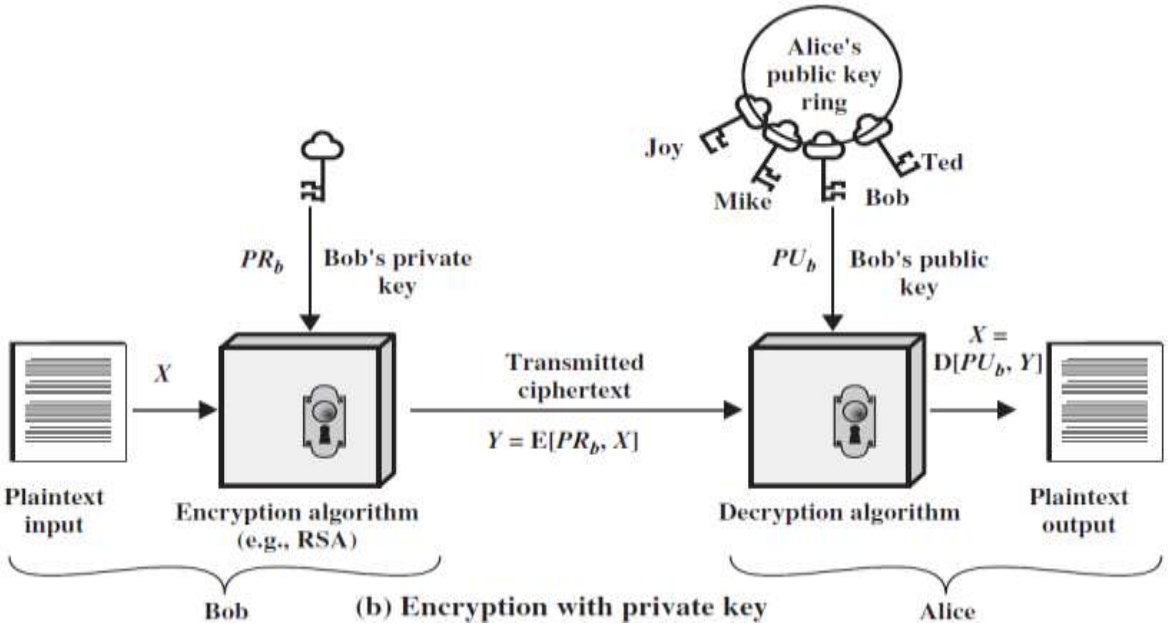
PR_b is known only to B, whereas PU_b is publicly available and therefore accessible by A.
With the message X and the encryption key PU_b as input, A forms the ciphertext $Y = [Y1, Y2, \ldots, Y_N]$:

$$Y = \mathrm{E}(PU_b, X)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$X = \mathrm{D}(PR_b, Y)$$

Public key cryptography for proving Authentication:



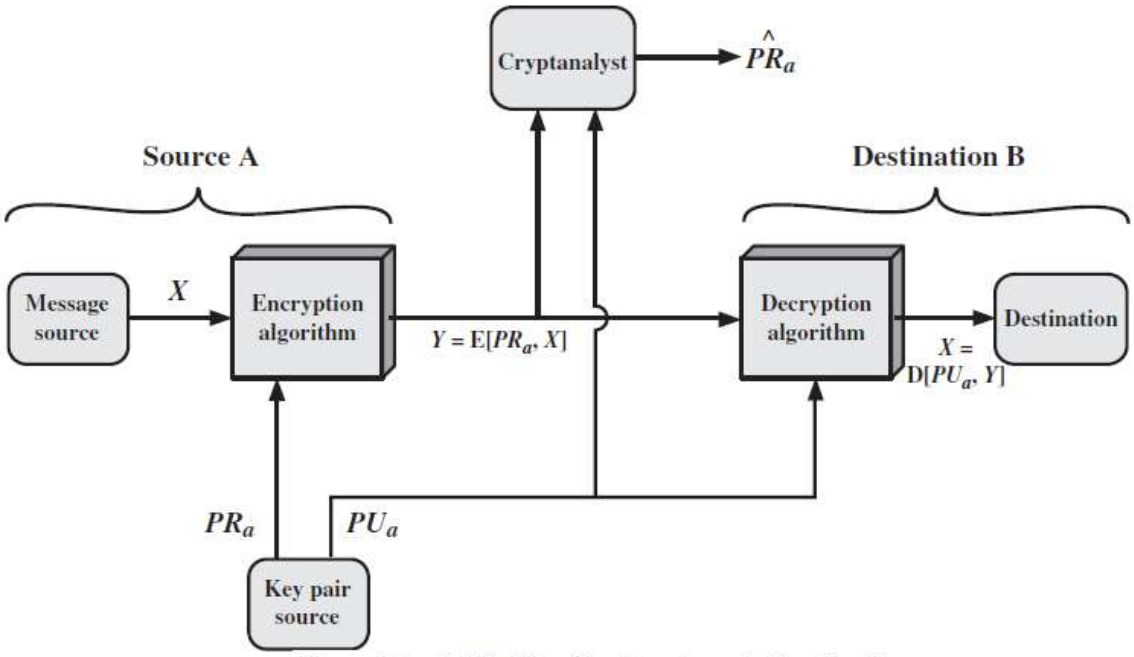


Figure 9.3 Public-Key Cryptosystem: Authentication

The above diagrams show the use of public-key encryption to provide authentication:

$$Y = \mathrm{E}(PR_a, X)$$

$$X = \mathrm{D}(PU_a, Y)$$

- In this case,A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a **digital signature.**

- It is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.

Public key cryptography for both authentication and confidentiality (Secrecy)

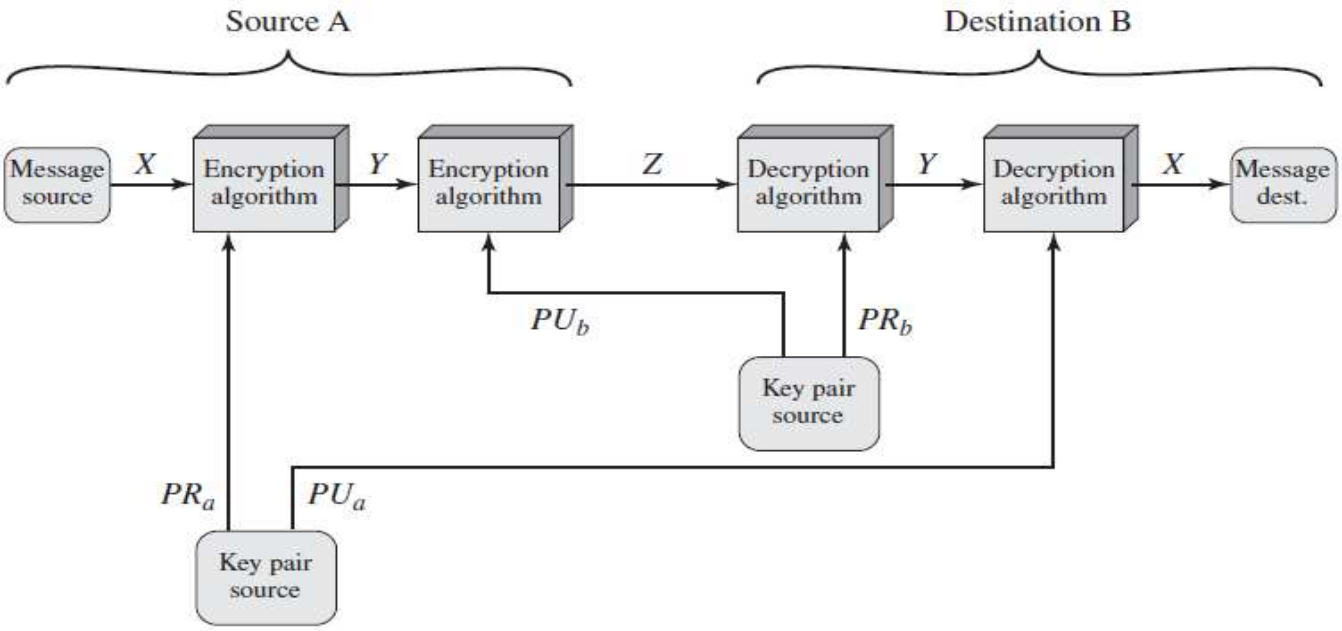


Figure 9.4 Public-Key Cryptosystem: Authentication and Secrecy

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (above figure):

$$Z = \mathrm{E}(PU_b, \mathrm{E}(PR_a, X))$$

$$X = \mathrm{D}(PU_a, \mathrm{D}(PR_b, Z))$$

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided.

Applications for Public-Key Cryptosystems

Public-key systems are characterized by
the use of a cryptographic algorithm with two keys, one held private and one available publicly. Depending on the application, the sender uses either the sender's private key or the receiver's public key, or both, to perform some type of cryptographic function.
the use of **public-key cryptosystems**

into three categories

• Encryption /decryption: The sender encrypts a message with the recipient's public key.

• Digital signature: The sender "signs" a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.

• Key exchange: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

Applications for Public-Key Cryptosystems

Algorithm	**Encryption/Decryption**	**Digital Signature**	**Key Exchange**
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Public-Key Cryptanalysis

As with symmetric encryption, a public-key encryption scheme is vulnerable to a brute-force attack. The countermeasure is the same: Use large keys. However, there is a tradeoff to be considered. Public-key systems depend on the use of some sort of invertible mathematical function. The complexity of calculating these functions may not scale linearly with the number of bits in the key but grow more rapidly than that. Thus, the key size must be large enough to make brute-force attack impractical but small enough for practical encryption and decryption. In practice, the key sizes that have been proposed do make brute-force attack impractical but result in encryption/decryption speeds that are too slow for general-purpose use. Instead, as was mentioned earlier, public-key encryption is currently confined to key management and signature applications.

RSA

- It is the most common public key algorithm.
- This RSA name is get from its inventors first letter (Rivest (R), Shamir (S) and Adleman (A)) in the year 1977.
- The RSA scheme is a block cipher in which the plaintext & ciphertext are integers between 0 and n-1 for some 'n'.
- A typical size for 'n' is 1024 bits or 309 decimal digits. That is, n is less than 2^{1024}

Description of the Algorithm:

- RSA algorithm uses an expression with exponentials.
- In RSA plaintext is encrypted in blocks, with each block having a binary value less than some number n. that is, the block size must be less than or equal to $\mathbf{\log_2(n)}$
- **RSA** uses two exponents 'e' and 'd' where e→public and d→private.
- Encryption and decryption are of following form, for some PlainText 'M' and CipherText block 'C'

$$C = M^e \bmod n$$

$$M = C^d \bmod n$$

$$M = C^d \bmod = (M^e \bmod n)^d \text{ mon } n = (M^e)^d \bmod n = M^{ed} \bmod n$$

Both sender and receiver must know the value of n.
The sender knows the value of 'e' & only the reviver knows the value of 'd' thus this is a public key encryption algorithm with a
Public key PU={e, n}
Private key PR={d, n}

Requirements:
The RSA algorithm to be satisfactory for public key encryption, the following requirements must be met:

1. It is possible to find values of e, d n such that " $\mathbf{M^{ed} \bmod n = M}$ " for all M<n
2. It is relatively easy to calculate " $\mathbf{M^e \bmod n}$ " and " $\mathbf{C^d \bmod n}$ "for M<n
3. It is infeasible to determine "d" given 'e' & 'n'. The " $\mathbf{M^{ed} \bmod n = M}$ " relationship holds if 'e' & 'd' are multiplicative inverses modulo Ø(n).
   Ø(n)→ Euler Totient function
   For p,q primes where p*q and p≠q.
   Ø(n)= Ø(pq)=(p-1)(q-1)

   Then the relation between 'e' & 'd' can be expressed as " $\boxed{\mathbf{ed \bmod Ø(n)=1}}$ "
   this is equivalent to saying

$$\boxed{\mathbf{ed \equiv 1 \bmod Ø(n)}}$$
$$\boxed{\mathbf{d \equiv e^{-1} \bmod Ø(n)}}$$

   That is 'e' and 'd' are multiplicative inverses mod Ø(n).
   Note: according to the rules of modular arithmetic, this is true only if 'd' (and 'e') is relatively prime to Ø(n).
   Equivalently **gcd(Ø(n), d)=1.**

Steps of RSA algorithm:
Step 1→Select 2 prime numbers p & q
Step 2→Calculate n=pq
Step 3→Calculate Ø(n)=(p-1)(q-1)
Step 4→ Select or find integer e (public key) which is relatively prime to Ø(n).
ie., e with gcd (Ø(n), e)=1 where 1<e< Ø(n).
Step 5→ Calculate "d" (private key) by using following condition. $\boxed{\mathbf{ed = 1 \bmod Ø(n)}}$
d< Ø(n).
Step 6→ Perform encryption by using $\boxed{C = M^e \bmod n}$
Step 7→ perform Decryption by using $\boxed{M = C^d \bmod n}$

Example:
1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$.
3. Calculate $Ø(n) = (p - 1)(q - 1) = 16 \times 10 = 160$.
4. Select e such that e is relatively prime to $Ø(n) = 160$ and less than $Ø(n)$; we choose $e = 7$.
5. Determine d such that $\boldsymbol{de \equiv 1 \pmod{160}}$ and $d < 160$.The correct value is $d = 23$, because $23 * 7 = 161 = (1 \times 160) + 1$; d can be calculated using the extended Euclid's algorithm

The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.
The example shows the use of these keys for a plaintext input of $M = 88$. For encryption,

we need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \bmod 187 = [(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)] \bmod 187$$

$$88^1 \bmod 187 = 88$$

$$88^2 \bmod 187 = 7744 \bmod 187 = 77$$

$$88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$$

$$88^7 \bmod 187 = (88 \times 77 \times 132) \bmod 187 = 894{,}432 \bmod 187 = 11$$

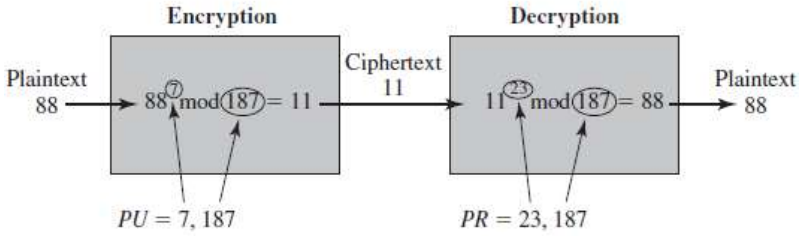


Figure 9.6 Example of RSA Algorithm

For decryption, we calculate $M = 11^{23} \bmod 187$:

$$11^{23} \bmod 187 = [(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^8 \bmod 187) \times (11^8 \bmod 187)] \bmod 187$$

$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 \times 121 \times 55 \times 33 \times 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$$

The Security of RSA

Four possible approaches to attacking the RSA algorithm are

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

Diffie-Hellman Key Exchange:

- Diffie-Hellman key exchange is the first published public key algorithm
- This Diffie-Hellman key exchange protocol is also known as exponential key agreement. And it is based on mathematical principles.
- The purpose of the algorithm is to enable two users to exchange a key securely that can then be used for subsequent encryption of messages.
- This algorithm itself is limited to exchange of the keys.
- This algorithm depends for its effectiveness on the difficulty of computing discrete logarithms.
- The discrete logarithms are defined in this algorithm in the way of define a primitive root of a prime number.
  - Primitive root: we define a primitive root of a prime number P as one whose power generate all the integers form 1 to P-1 that is if 'a' is a primitive root of the prime number P, then the numbers

    $a \bmod P, a^2 \bmod P, a^3 \bmod P, \ldots\ldots a^{P-1} \bmod P$ are distinct and consist of the integers form 1 through P-1 in some permutation.

    For any integer 'b' and 'a', here 'a' is a primitive root of prime number P, then

    $b \equiv a^i \bmod P \qquad 0 \le i \le (P-1)$

  The exponent i → is refer as discrete logarithm or index of b for the base a, mod P.

  The value denoted as **$\text{ind}_{a,p}(b)$**

Algorithm for Diffie-Hellman Key Exchange:

Step 1→ two public known numbers q, α

q→ Prime number

α→ primitive root of q and α< q.

Step 2 → if A & B users wish to exchange a key

a) User A select a random integer X_A<q and computes $Y_A = \alpha^{X_A} \bmod q$
b) User B independently select a random integer X_B <q and computes $Y_B = \alpha^{X_B} \bmod q$
c) Each side keeps the X value private and Makes the Y value available publicly to the outer side.

Step 3→ User A Computes the key as $K = (Y_B)^{X_A} \bmod q$

User B Computes the key as $K = (Y_A)^{X_B} \bmod q$

Step 4→ two calculation produce identical results

$K = (Y_B)^{X_A} \bmod q$

$K = (\alpha^{X_B} \bmod q)^{X_A} \bmod q$ (We know that $Y_B = \alpha^{X_B} \bmod q$)

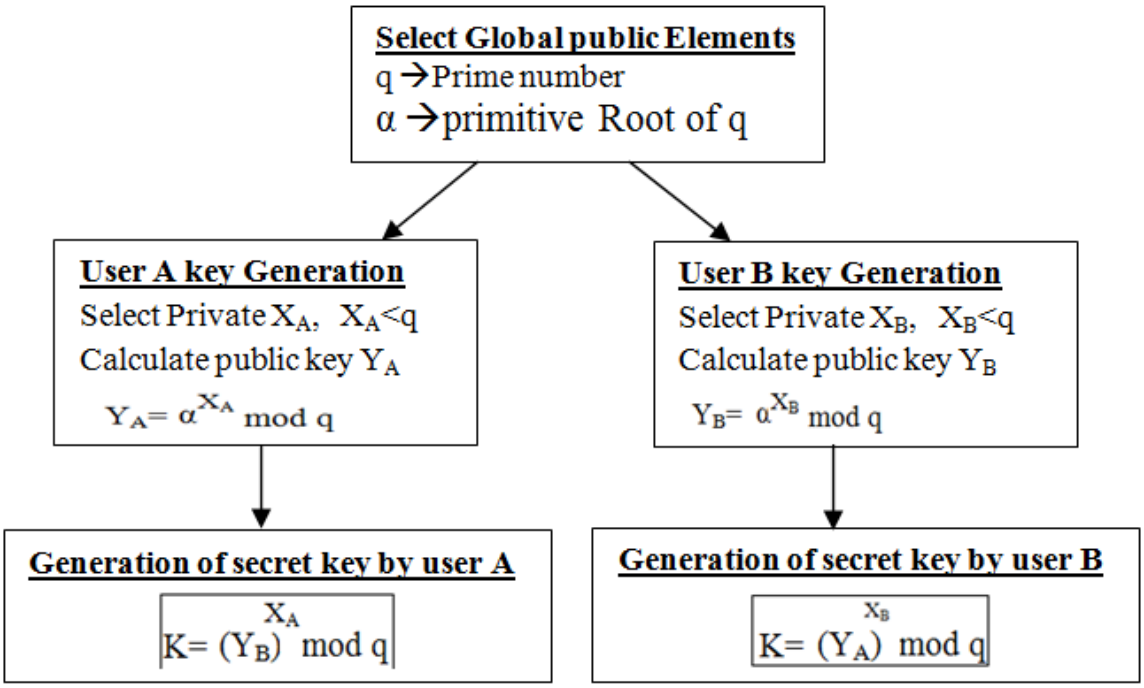
$= (\alpha^{X_B})^{X_A} \bmod q$

$= (\alpha^{X_A})^{X_B} \bmod q$

$= (\alpha^{X_A} \bmod q)^{X_B} \bmod q$

$= (Y_A)^{X_B} \bmod q$ (We know that $Y_A = \alpha^{X_A} \bmod q$)

The result is that the two sides have exchanged a secret key.



Example:

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case $\alpha = 3$. A and B select secret keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

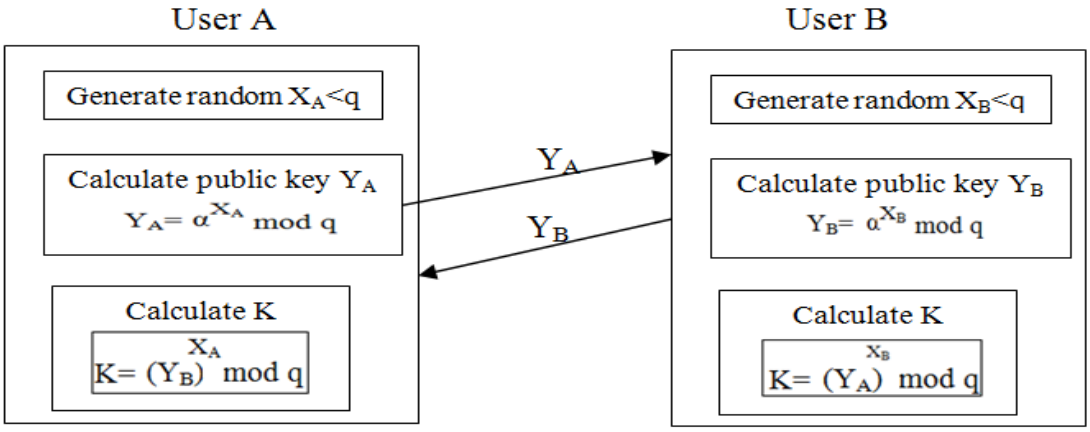
After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume an attacker would have available the following information:

$$q = 353; \alpha = 3; Y_A = 40; Y_B = 248$$



MAN-in the Middle Attack (MITM)

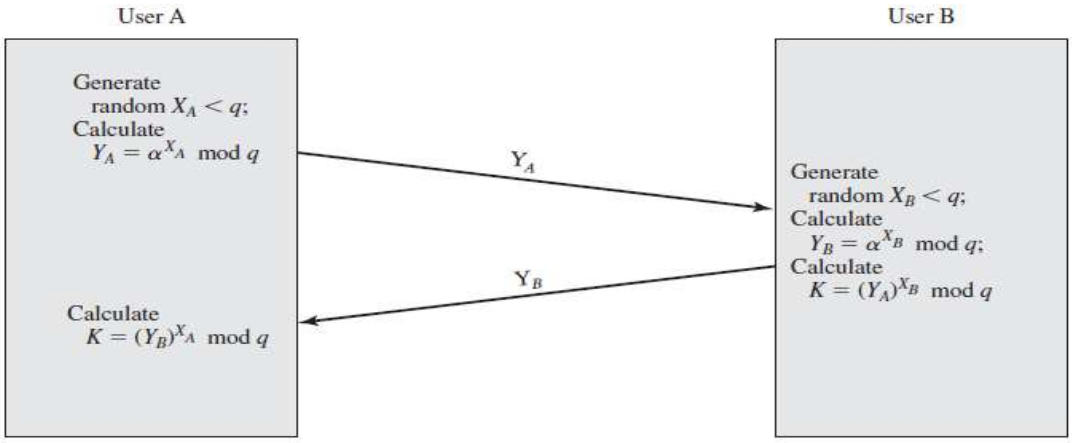


Figure 10.2 Diffie-Hellman Key Exchange

Definition: A man in the middle attack is a form of eavesdropping where communication between two users is monitored and modified by an unauthorized party.

Generally the attacker actively eavesdrops by intercepting (stoping) a public key message exchange.
The Diffie- Hellman key exchange is insecure against a "Man in the middle attack".

Suppose user 'A' & 'B' wish to exchange keys, and D is the adversary (opponent). The attack proceeds as follows.

1. 'D' prepares for the attack by generating two random private keys X_{D1} & X_{D2} and then computing the corresponding public keys Y_{D1} and Y_{D2}.
2. 'A' transmits 'Y_A' to 'B'
3. 'D' intercepts Y_A and transmits Y_{D1} to 'B'. and D also calculates $K2 = (Y_A)^{X_{D2}} \bmod q$.
4. 'B' receives Y_{D1} & calculate $K1 = (Y_{D1})^{X_B} \bmod q$.
5. 'B' transmits 'Y_B' to 'A"
6. 'D' intercepts 'Y_B' and transmits Y_{D2} to 'A' and 'D' calculate K1 $K1 = (Y_B)^{X_{D1}} \bmod q$.
7. A receives Y_{D2} and calculates $K2 = (Y_{D2})^{X_A} \bmod q$

At this point, Bob and Alice think that they share a secret key, but instead Bob and Darth share secret key *K*1 and Alice and Darth share secret key *K*2. All future communication between Bob and Alice is compromised in the following way.

1. A sends an encrypted message *M*: E(*K*2, *M*).
2. D intercepts the encrypted message and decrypts it to recover *M*.
3. D sends B $E(K1, M)$ or E(*K*1, *M*′), where *M*′ is any message. In the first case, D simply wants to eavesdrop on the communication without altering it. In the second case, D wants to modify the message going to B

The key exchange protocol is vulnerable to such an attack because it does not authenticate the participants. This vulnerability can be overcome with the use of digital signatures and public-key certificates.

Elliptic Curve Cryptography

- **Definition: Elliptic curve cryptography (ECC)** is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. These are analogy of existing public key cryptosystem in which modular arithmetic is replaced by operations defined over elliptic curve.
- The use of elliptic curves in cryptography was suggested independently by **Neal Koblitz** and **Victor S. Miller** in **1985.**
- Elliptic curve cryptography (ECC) is one of the most powerful but least understood types of cryptography in wide use today. An increasing number of websites make extensive use of ECC to secure everything from customers' HTTPS connections to how they pass data between data centers.

An elliptic curve is defined by an equation in two variables with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group.

Elliptic Curves over Real Numbers

Elliptic curves are not ellipses. They are so named because they are described by cubic equations,

$y^2 + axy + by = x^3 + cx^2 + dx + e$ is similar to equation of calculating circumference of an ellipse.

Where

a,b,c,d and e → real numbers.

X and Y are→taken on values in the real numbers.

For utilization of this in cryptography

$y^2 = x^3 + ax + b$ → EQ1, is sufficient.

Such equations are said to be cubic, or of degree 3, because the highest exponent they contain is a 3. Also included in the definition of an elliptic curve is a single element denoted *O* and called the *point at infinity* or the *zero point. To plot such a curve, we need to compute*

$y = \sqrt{x^3 + ax + b}$ For given values of and, the plot consists of positive and negative values of for each value of . Thus, each curve is symmetric about $y = 0$.

Two families of elliptic curves are used in cryptographic applications:

- Prime curves over $\mathbf{Z_p}$ ***[it is** Best for software application]*
- Binary curves over $\mathbf{GF(2^m)}$ ***[it is** Best for software application]*

Prime curves over Z_p

In Prime curves over $\mathbf{Z_p}$ **,** p → referred to as a modulus.

we use a cubic equation in which the variables and coefficients all take on values in the set of integers from **0** through p **- 1** and in which calculations are performed modulo p.

from EQ1, in this case coefficients and variables limited to $\mathbf{Z_p}$.

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$ → eq2

Now consider the set E*p*(*a*, *b*) consisting of all pairs of integers (*x*, *y*) that satisfy Equation eq2 together with a point at infinity .The coefficients a and b and the variables x and y are all elements of Zp.

For example, Equation eq2 is satisfied for $a = 1, b = 1, x = 9, y = 7, \alpha = 1$ $p = 23$:

$$7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$$
$$49 \bmod 23 = 739 \bmod 23$$
$$3 = 3$$

For example, let *p* = 23 and consider the elliptic curve *y*2 = *x*3 + *x* + 1 In this case, *a* = *b* = 1 For the set $E_{23}(1, 1)$, we are only interested in the nonnegative integers in the quadrant from (0, 0) through (*p* - 1, *p* - 1) that satisfy the equation mod *p*.

Elliptic Curves over GF(2^m):
A finite field GF(2^m) consists of 2m elements, together with addition & multiplication operations that can be defined over polynomials.

For elliptic Curves over GF(2^m), we use a cubic equation in which the variables and coefficients all take on values in GF(2^m), for some number m.
By this, the form of cubic equation appropriate for cryptographic application.
The form is $y^2 + xy = x^3 + ax^2 + b$ → EQ3.

To form a cryptographic system using elliptic curves, we need to find a "hard problem" corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation $Q = kP$ where $Q, P \in E_p(a, b)$ and $k < p$.
It is relatively easy to calculate *Q* given *k and P*
But it is relatively hard to determine given Q and P.
This is called the discrete logarithm problem for elliptic curves.

ECC Diffie-Hellman Key Exchange:

ECC can do key exchange, that is analogous to Diffie Hellman.
Key exchange using elliptic curves can be done in the following manner.
First pick a large integer q , which is either a prime number P or an integer of the form 2^m and elliptic curve parameters a & b for equation $y^2 \bmod p = (x^3 + ax + b) \bmod p$ or $y^2 + xy = x^3 + ax^2 + b$.
This define elliptic group of point $E_q(a,b)$.
Pick a base point G=(x1,y1) in $E_p(a,b)$ whose order is a very large value n.
The order n of a point G on an elliptic curve is the smallest +ve integer n such that $nG=0.E_q(a,b)$

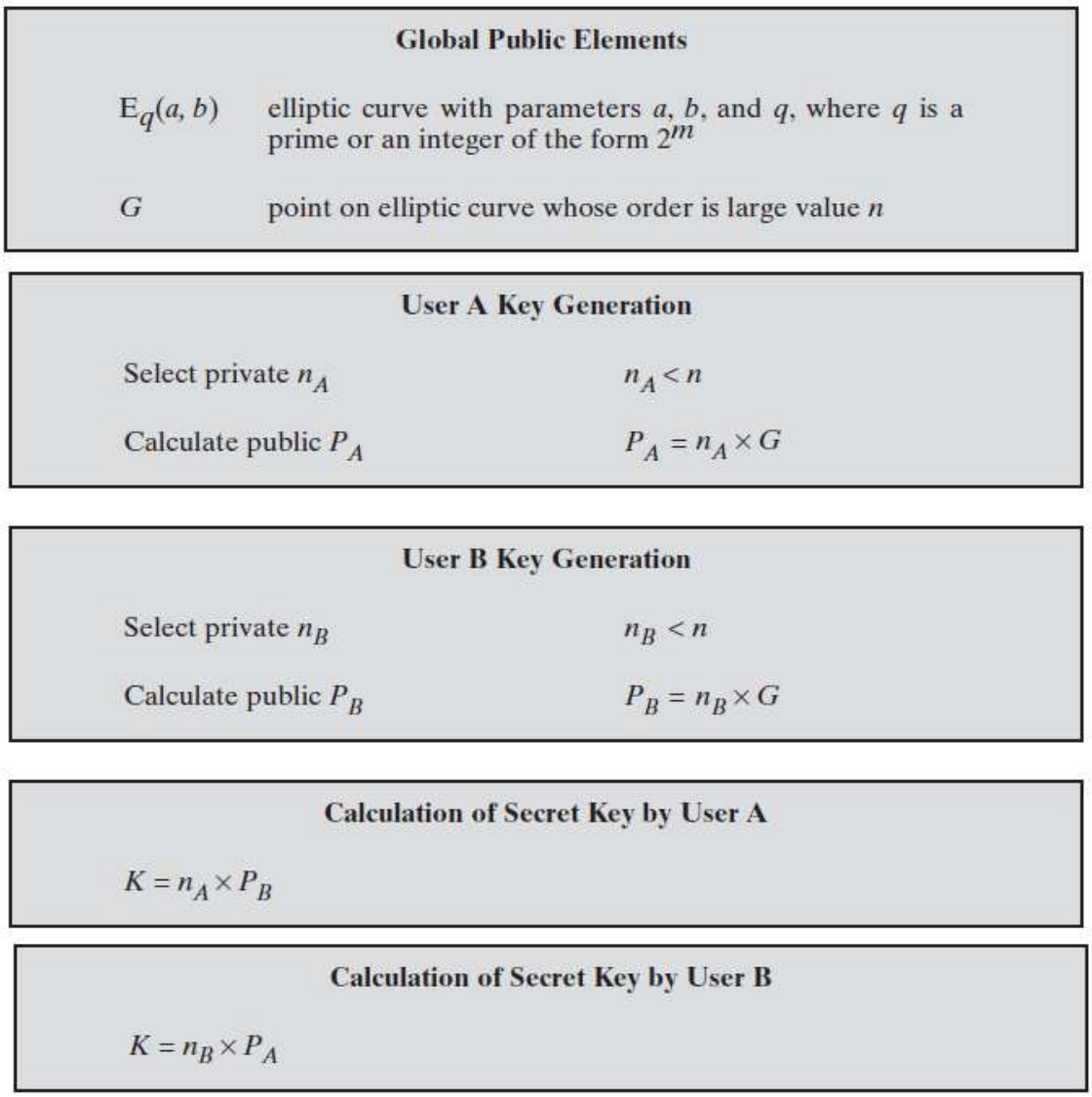


Figure 10.7 ECC Diffie-Hellman Key Exchange

$$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$$

Elliptic Curve Encryption/Decryption:

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. In this subsection, we look at perhaps the simplest. The first task in this system is to encode the plaintext message m to be sent as an x–y point P_m. It is the point P_m that will be encrypted as a ciphertext and subsequently decrypted. Note that we cannot simply encode the message as the x or y coordinate of a point, because not all such coordinates are in $E_q(a, b)$; for example, see Table 10.1. Again, there are several approaches to this encoding, which we will not address here, but suffice it to say that there are relatively straightforward techniques that can be used.

As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group $E_q(a, b)$ as parameters. Each user A selects a private key n_A and generates a public key $P_A = n_A \times G$.

To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m consisting of the pair of points:

$$C_m = \{kG, P_m + kP_B\}$$

Note that A has used B's public key P_B. To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$$P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$$

A has masked the message P_m by adding kP_B to it. Nobody but A knows the value of k, so even though P_b is a public key, nobody can remove the mask kP_B. However, A also includes a "clue," which is enough to remove the mask if one knows the private key n_B. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed to be hard.

As an example of the encryption process (taken from [KOBL94]), take $p = 751$; $E_p(-1, 188)$, which is equivalent to the curve $y^2 = x^3 - x + 188$; and $G = (0, 376)$. Suppose that A wishes to send a message to B that is encoded in the elliptic point $P_m = (562, 201)$ and that A selects the random number $k = 386$. B's public key is $P_{B = (201, 5)}$. We have $386(0, 376) = (676, 558)$, and $(562, 201) + 386(201, 5) = (385, 328)$. Thus, A sends the cipher text $\{(676, 558), (385, 328)\}$.

Module 3: Key management and Distribution

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. Therefore, the 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.

Physical delivery (1 & 2) is simplest - but only applicable when there is personal contact between recipient and key issuer. This is fine for link encryption where devices & keys occur in pairs, but does not scale as number of parties who wish to communicate grows. 3 is mostly based on 1 or 2 occurring first.

A third party, whom all parties trust, can be used as a **trusted intermediary** to mediate the establishment of secure communications between them (4). Must trust intermediary not to abuse the knowledge of all session keys. As number of parties grow, some variant of 4 is only practical solution to the huge growth in number of keys potentially needed.

Key distribution centre:

- 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.
- 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 and then discarded
- **Master key** is shared by the key distribution center and an end system or user and used to encrypt the session key.

Key Distribution Scenario:

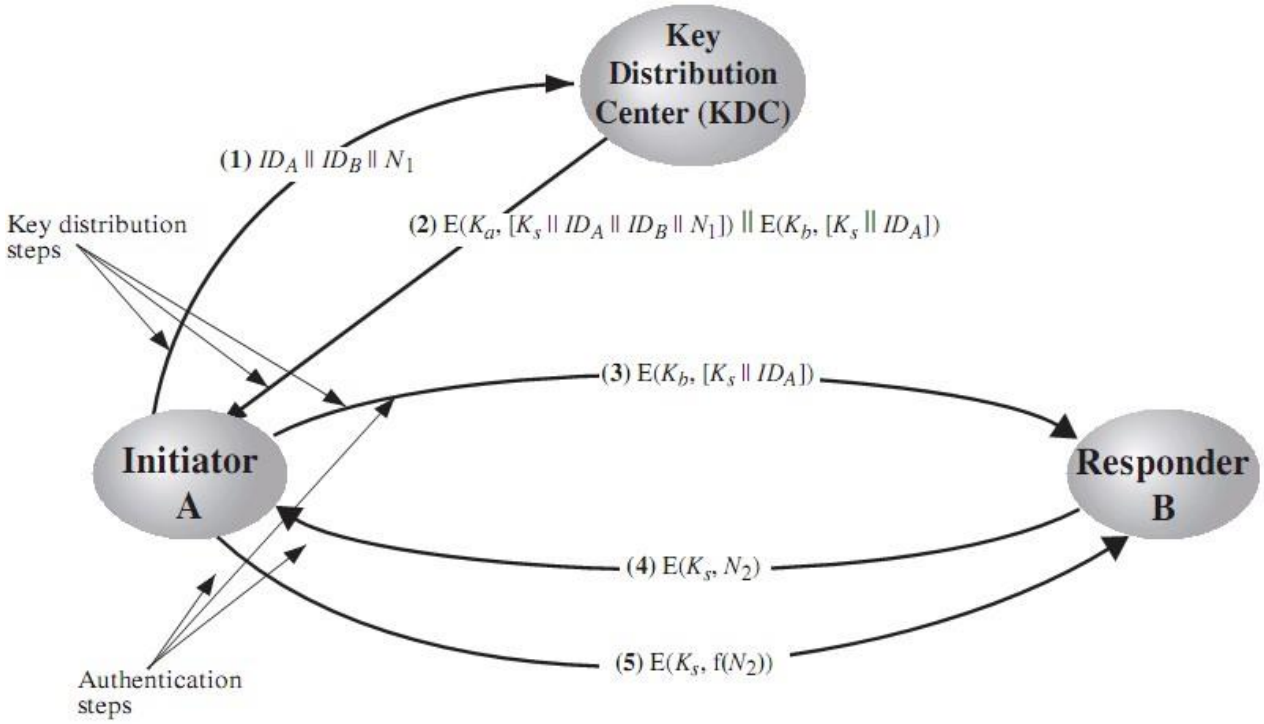


Figure 14.3 Key Distribution Scenario

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, 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 Ka 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, Ks,** 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, Ks to be used for the session
- An identifier of A (e.g., its network address), IDA

These last two items are encrypted with Kb (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 3, perform an authentication function.

Major Issues with KDC:

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.
- If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC.
- 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.

Session Key Lifetime

- 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**.
- 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 14.3is 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.

The steps involved in establishing a connection are shown in Figure 14.4.

1. When one host wishes to set up a connection to another host, it transmits a connection-request packet.
2. The SSM saves that packet and applies to the KDC for permission to establish the connection.
3. 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.

4. The requesting SSM can now release the connection request packet, and a connection is set up between the two end systems.
5. All user data exchanged between the two end systems are encrypted by their respective SSMs using the onetime session key.

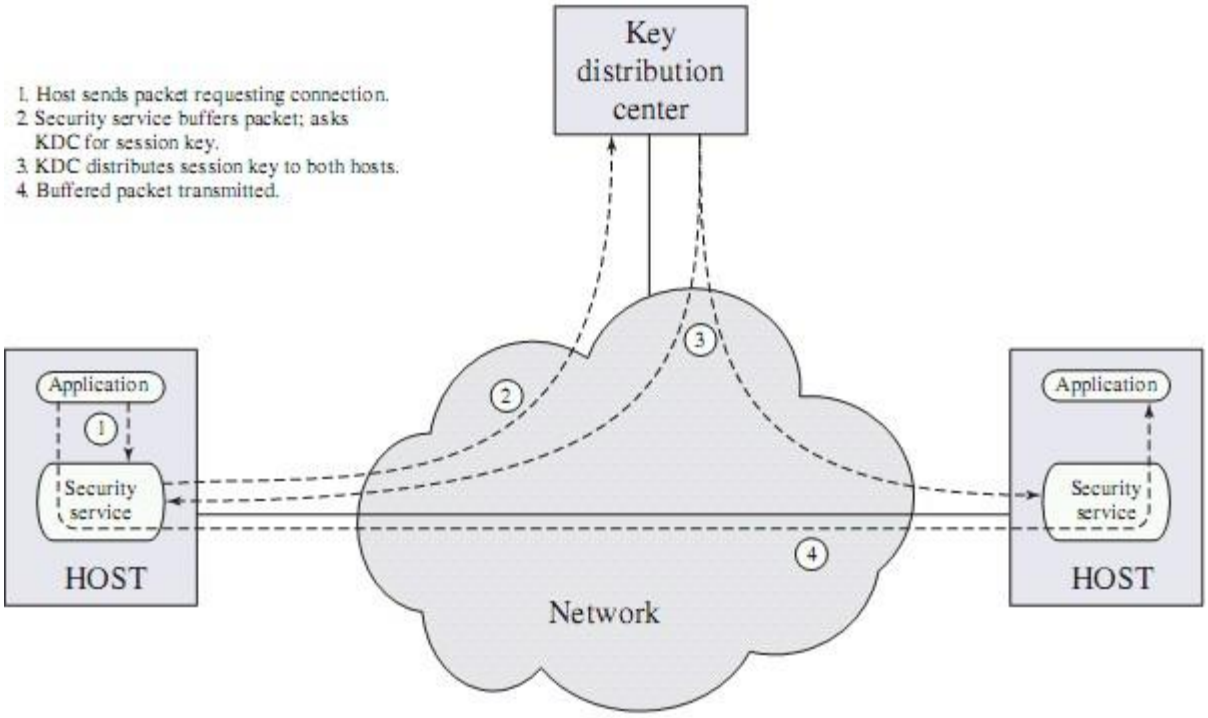


Figure 14.4 Automatic Key Distribution for Connection-Oriented Protocol

- 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, .

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.

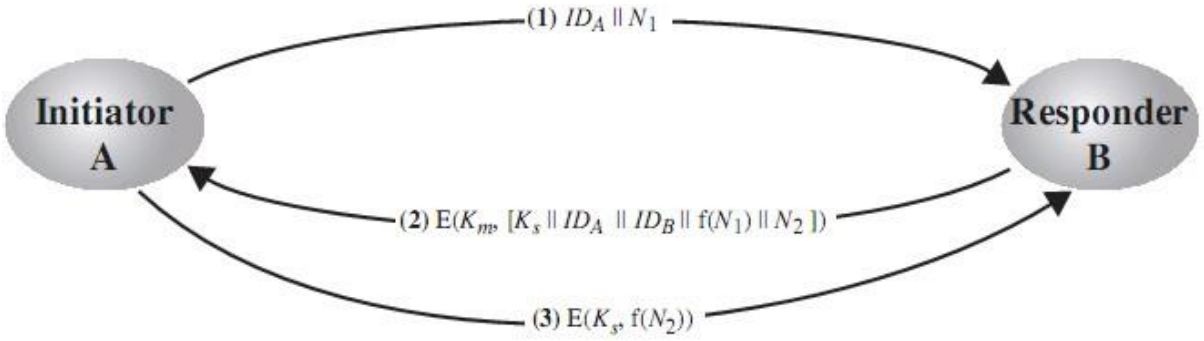


Figure 14.5 Decentralized Key Distribution

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.

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 here. 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.

The control vector is cryptographically coupled with the key at the time of key generation at the KDC.

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. 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 = Km $\oplus$H
Ciphertext = E([Km $\oplus$H], Ks)

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

D([Km$\oplus$H], E([Km $\oplus$H], Ks))

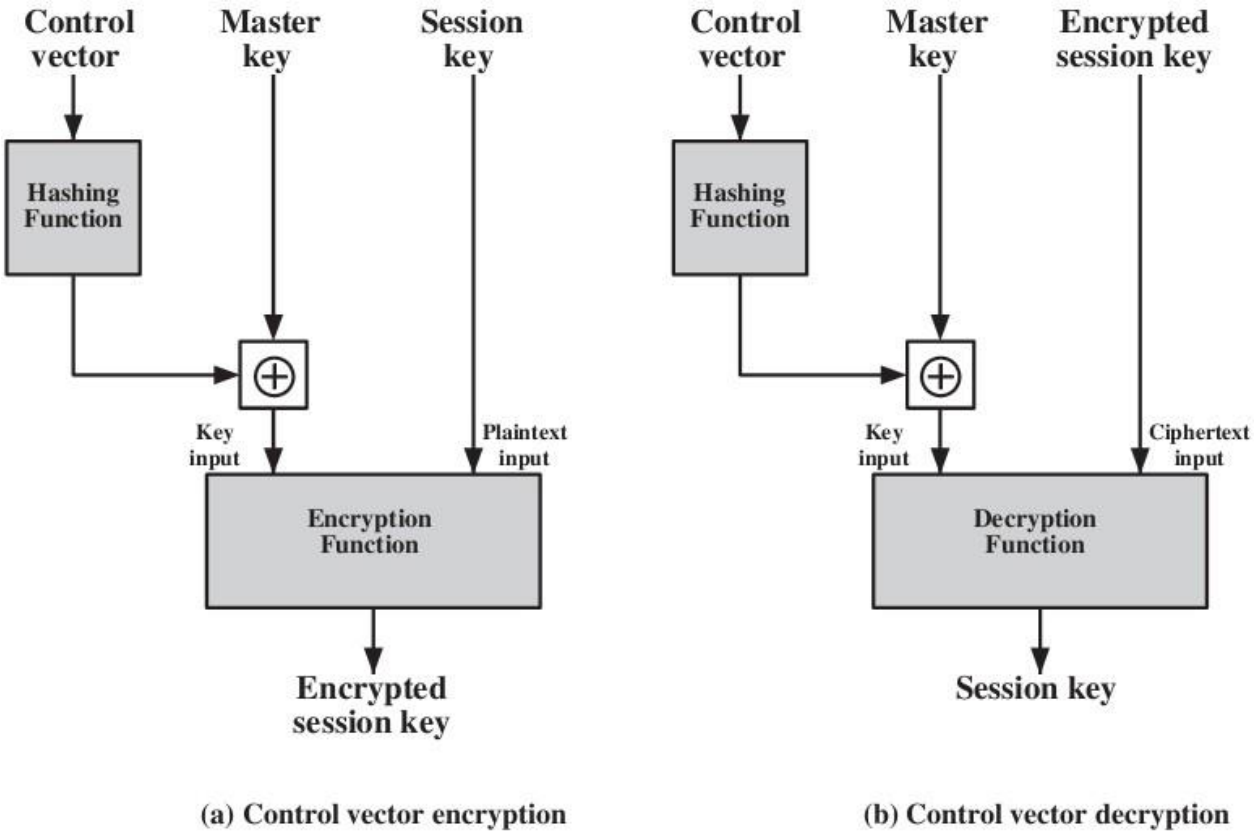


Figure 14.6 Control Vector Encryption and Decryption

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.

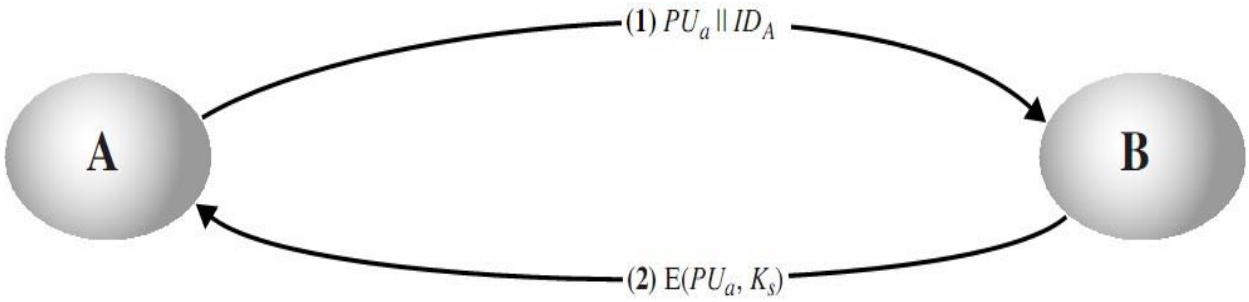
Use of the control vector has two **advantages over use of an 8-bit tag.** First, there is no restriction on length of the control vector, which enables arbitrarily complex controls to be imposed on key use. Second, the control vector is available in clear form at all stages of operation. Thus, control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

14.2 SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION

- Once public keys have been distributed or have become accessible, secure communication that thwarts eavesdropping, tampering, or both, is possible.
- Public-key encryption provides for the distribution of secret keys to be used for conventional encryption.

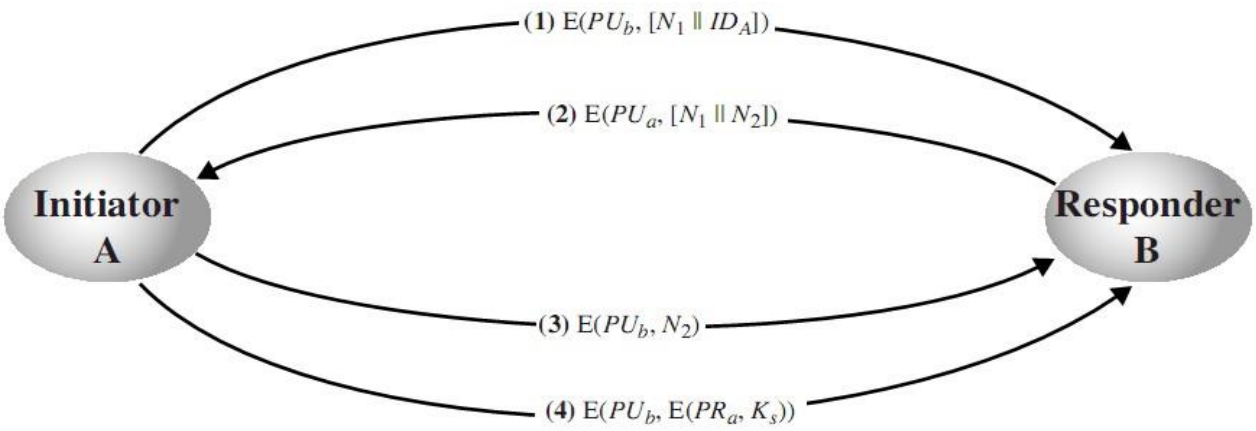
Simple Secret Key Distribution

- A generates a public/private key pair {PUa, PRa} and transmits a message to B consisting of PUa and an identifier of A, IDA
- B generates a secret key, Ks, and transmits it to A, encrypted with A's public key.
- A computes D(PRa, E(PUa, Ks)) to recover the secret key. Because only A can decrypt the message, only A and B will know the identity of Ks.
- A discards PUa and PRa and B discards PUa.



Here third party can intercept messages and then either relay the intercepted message or substitute another message Such an attack is known as a **man-in-the-middle attack.**

Secret Key Distribution with Confidentiality and Authentication:



- 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 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
- A returns N_2 encrypted using B's public key, to assure B that its correspondent 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.
- B computes $D(PU_a, D(PR_b, M))$ to recover the secret key.

A Hybrid Scheme:

Yet another way to use public-key encryption to distribute secret keys is a hybrid approach.

- 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 addition of a public-key layer provides a secure, efficient means of distributing master keys.

Distribution of Public Keys:

Several techniques have been proposed for the distribution of public keys, which can mostly be grouped into the categories shown.

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

Public Announcement of Public Keys

The point of public-key encryption is that the public key is public, hence any participant can send his or her public key to any other participant, or broadcast the key to the community at large. eg. append PGP keys to email messages or post to news groups or email list

Figure 10.1 Uncontrolled Public Key Distribution

Its major weakness is forgery, anyone could pretend to be user A and send a public key to another participant or broadcast such a public key. Until the forgery is discovered they can masquerade as the claimed user.

Publicly Available Directory

- can obtain greater security by registering keys with a public directory
- directory must be trusted with properties:

  - The authority maintains a directory with a {name, public key} entry for each participant.
  - Each participant registers a public key with the directory authority.
  - A participant may replace the existing key with a new one at any time because the corresponding private key has been compromised in some way.
  - Participants could also access the directory electronically. For this purpose, secure, authenticated communication from the authority to the participant is mandatory.

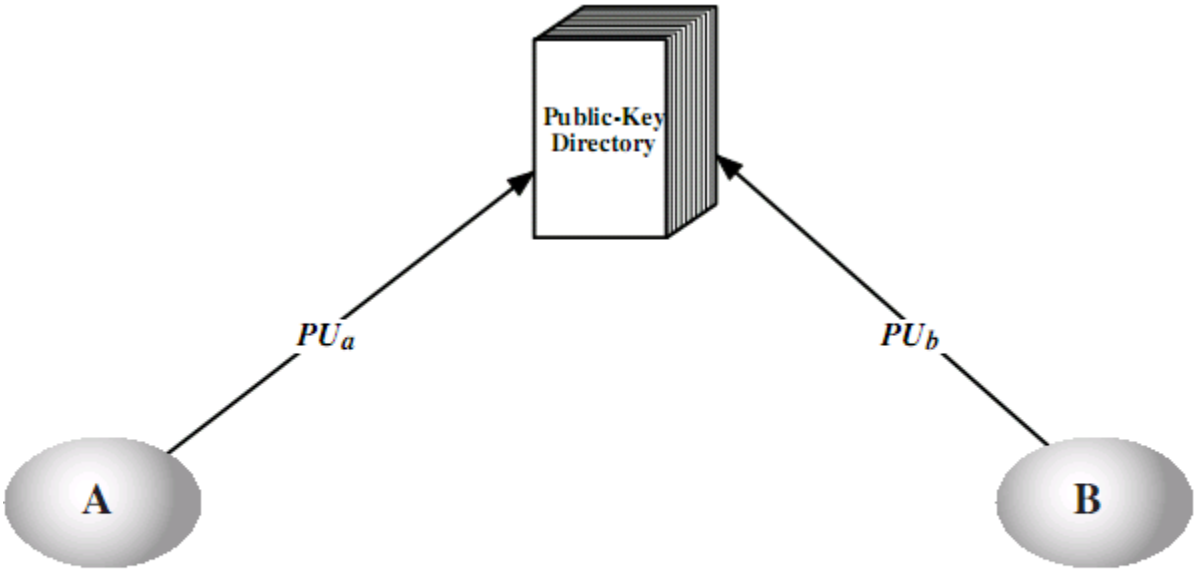


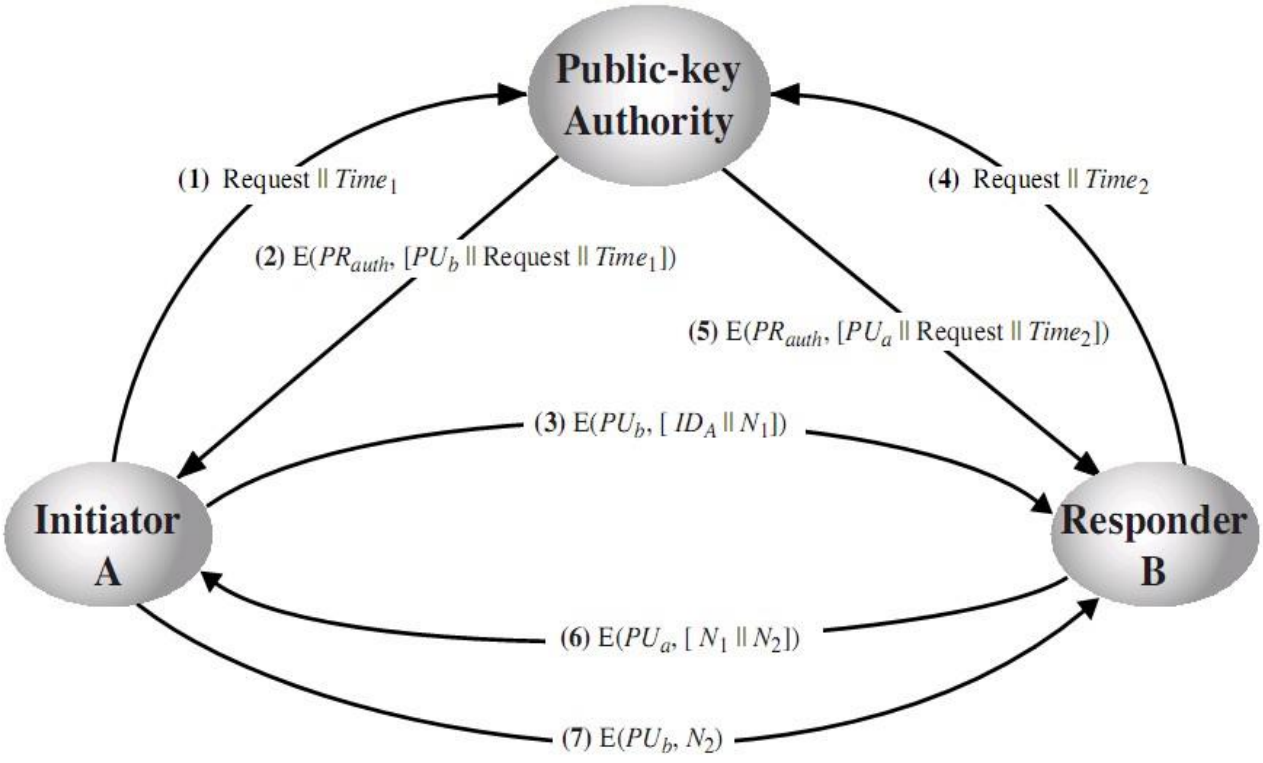
Figure 10.2 Public Key Publication

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.
- It requires users to know the public key for the directory, and that they interact with directory in real-time to obtain any desired public key securely.
- Totally seven messages are required.



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, 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, 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. B retrieves A's public key from the authority in the same manner as A retrieved B's public key.
5. 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, encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Certificates

- A user must appeal to the authority for a public key for every other user that it wishes to contact and it is vulnerable to tampering too.
- Public key certificates can be used to exchange keys without contacting a public-key authority.
- A certificate binds an **identity** to **public key**, with all contents **signed** by a trusted Public-Key or Certificate Authority (CA).
- This can be verified by anyone who knows the public-key authorities public-key.

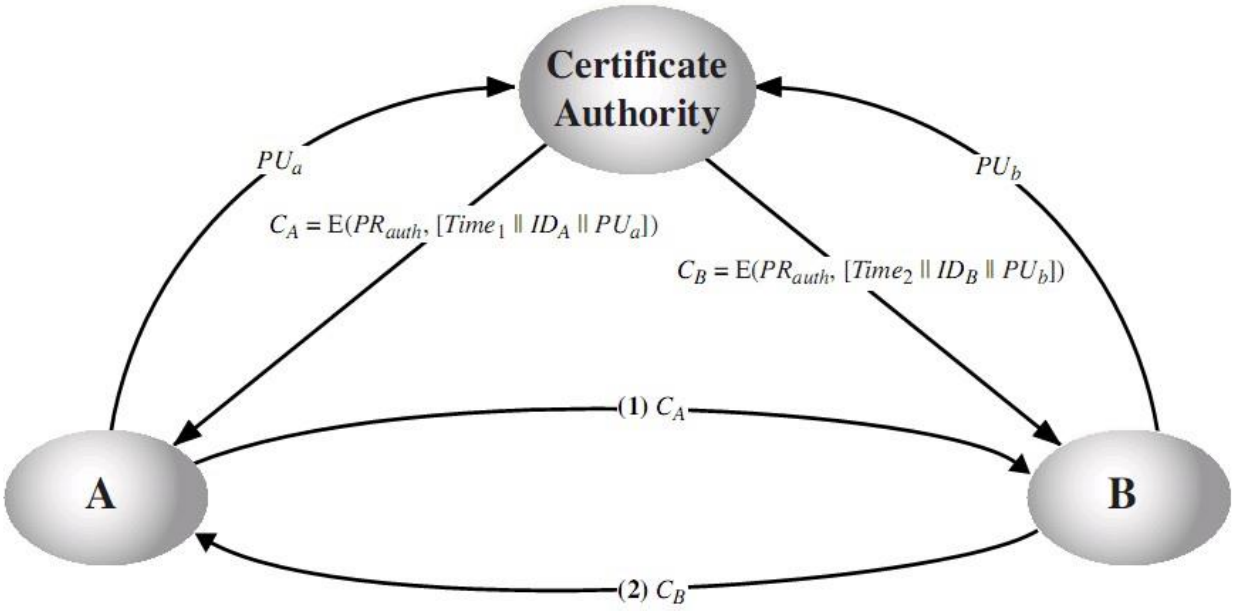
A participant can also convey its key information to another by transmitting its certificate.

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.
4. Any participant can verify the currency of the certificate.

One scheme has become universally accepted for formatting public-key certificates: the X.509 standard.

X.509 certificates are used in most network security applications, including IP security, secure sockets layer (SSL), secure electronic transactions (SET), and S/MIME.



14.4 X.509 CERTIFICATES

X.509 is part of the X.500 series of recommendations that define a directory service, being a server or distributed set of servers that maintains a database of information about users.

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

The X.509 certificate format is widely used, in for example S/MIME, IP Security and SSL/TLS and SET. X.509 was initially issued in 1988. The standard was ubsequently revised to address some of the security concerns; a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

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.

The standard uses the notation for a certificate of:

CA<<A>> where the CA signs the certificate for user A with its private key. In more detail CA<<A>> = CA {V, SN, AI, CA, UCA, A, UA, Ap, TA}.

If the corresponding public key is known to a user, then that user can verify that a certificate signed by the CA is valid.

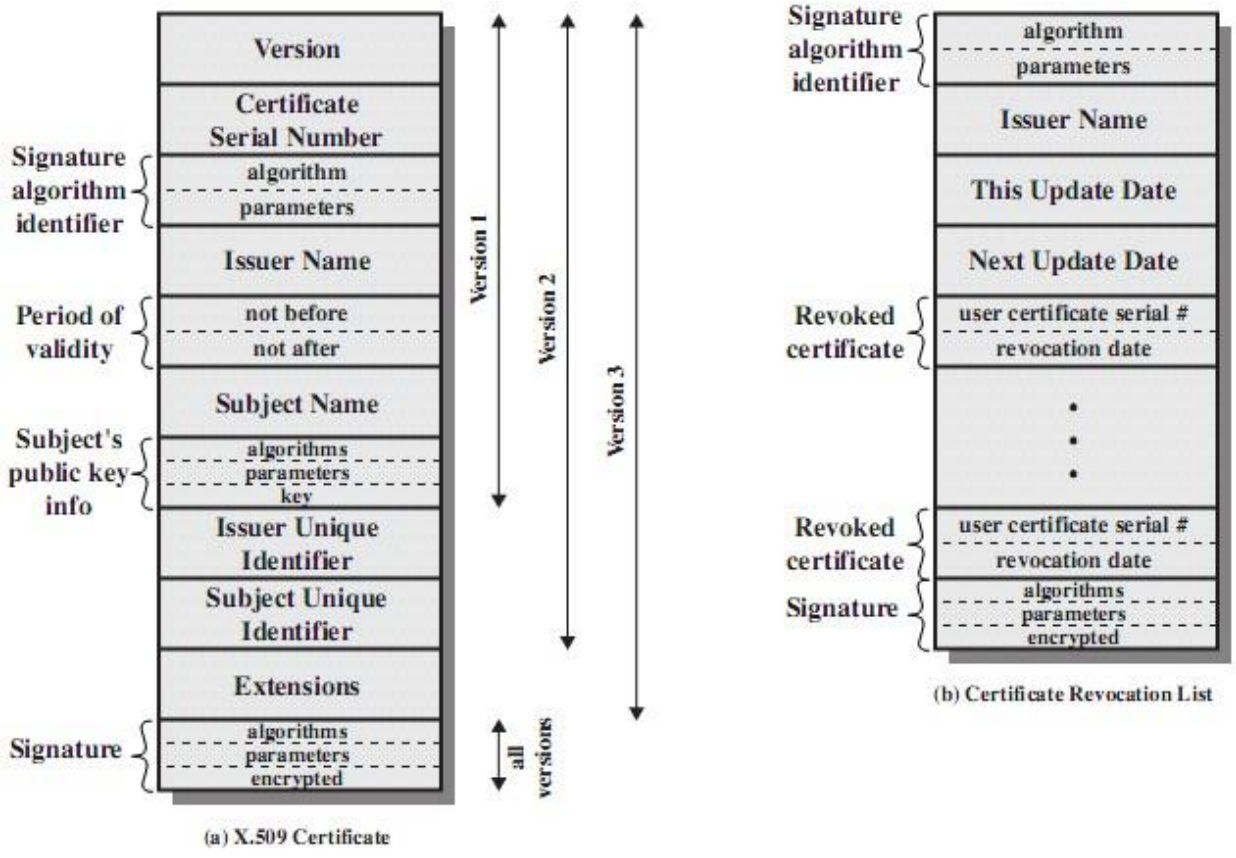


Figure 14.4 X.509 Formats

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

The unique identifier fields were added in version 2 to handle the possible reuse of subject and/or issuer names over time. These fields are rarely used. The standard uses the following notation to define a certificate:

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

where

Y<< X >> = the certificate of user X issued by certification authority Y

Y {I} = the signing of I by Y. It consists of I with an encrypted hash code appended

V = version of the certificate

SN = serial number of the certificate

AI = identifier of the algorithm used to sign the certificate

CA = name of certificate authority

UCA = optional unique identifier of the CA

A = name of user A

UA = optional unique identifier of the user A

Ap = public key of user A

T^A = period of validity of the certificate

Obtaining a 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 with out 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.

CA Hierarchy:

If both parties use the same CA, they know its public key and can verify others certificates. If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Hence there has to be some means to form a chain of certifications between the CA's used by the two parties, by the use of client and parent certificates. 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. It is assumed that each client trusts its parent's certificates.

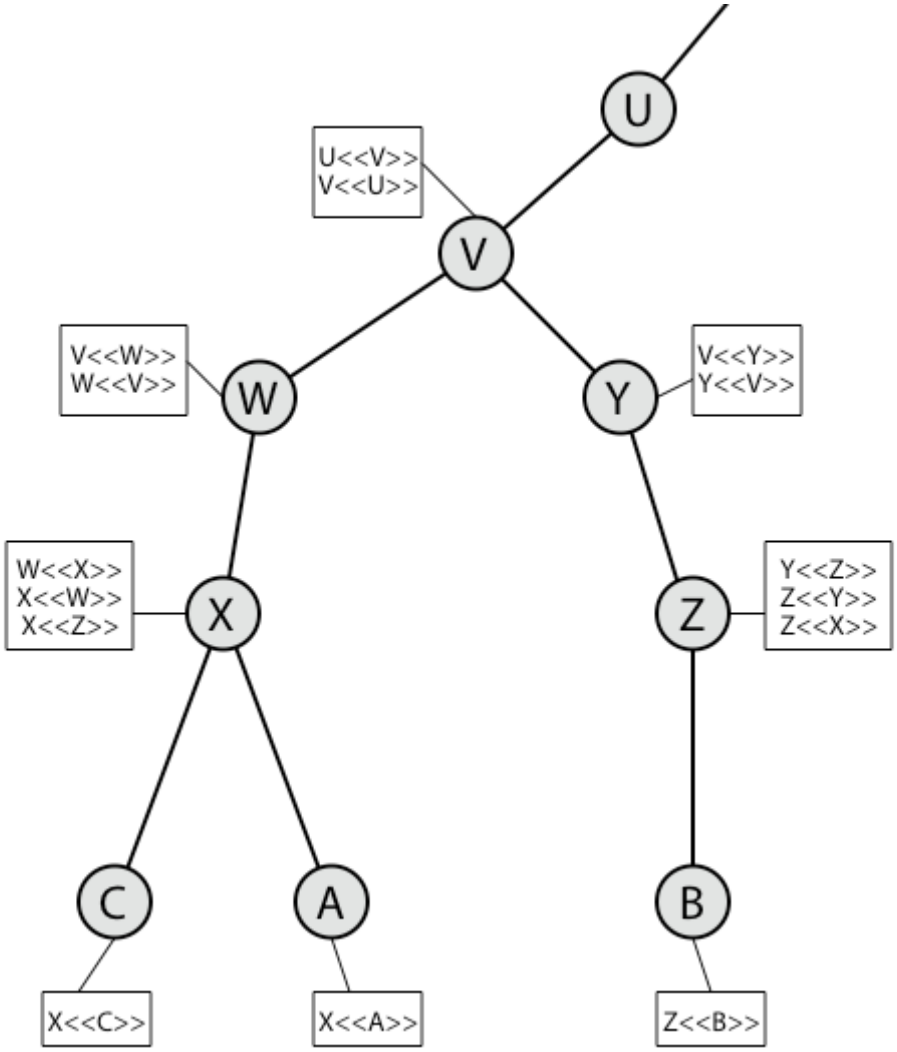


Figure 14.15 illustrates the use of an X.509 hierarchy to mutually verify clients certificates. The connected circles indicate the hierarchical relationship among the CAs; the

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, we can track chains of certificates as follows:

A acquires B certificate using chain

X<<W>>W<<V>>V<<Y>>Y<<Z>>Z<<B>>

B acquires A certificate using chain:

Z<<Y>>Y<<V>>V<<W>>W<<X>>X<<A>>

Certificate Revocation:

A certificate includes a period of validity. 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 a range of 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.

To support this, each CA must maintain a list consisting of all revoked but not expired certificates issued by that CA, known as the certificate revocation list (CRL). Each certificate revocation list (CRL) posted to the directory is signed by the issuer and includes (as shown in Figure 14.14b previously) 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, by checking the directory CRL each time a certificate is received, this often does not happen in practice.

X.509 Version 3

The X.509 version 2 format does not convey all of the information. Rather than continue to add fields to a fixed format, standards developers felt that a more flexible approach was needed. X.509 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 or not.

Certificate Extensions

The certificate extensions fall into three main categories:

- **Key and policy information** - 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.
- **Subject and issuer attributes** - support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject; eg. postal address, email address, or picture image
- **Certification path constraints** - allow constraint specifications to be included in certificates issued for CA's by other CA's that may restrict the types of certificates that can be issued by the subject CA or that may occur subsequently in a certification chain.

****