UNIT IV

Key Management and Distribution

◆ Key distribution is the function that delivers a key to two parties who wish to exchange secure encrypted data. Some sort of mechanism or protocol is needed to provide for the secure distribution of keys.

◆ Key distribution often involves the use of master keys, which are infrequently used and are long lasting, and session keys, which are generated

and distributed for temporary use between two parties.

◆ Public-key encryption schemes are secure only if the authenticity of the public key is assured. A public-key certificate scheme provides the necessary security.

◆ X.509 defines the format for public-key certificates. This format is widely used in a variety of applications.

◆ A public-key infrastructure (PKI) is defined as the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

1.1 SYMMETRIC KEY DISTRIBUTION

USING SYMMETRIC ENCRYPTION

For symmetric encryption to work, the two parties to an exchange must share the same key, and that key must be protected from access by others. Furthermore, frequent key changes are usually desirable to limit the amount of data compromised if an attacker learns the key. Therefore, the strength of any cryptographic system rests with the *key distribution technique.*

For two parties A and B, key distribution can be achieved in a number of ways, as follows:

1. A can select a key and physically deliver it to B.

2. A third party can select the key and physically deliver it to A and B.

3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.

4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B.

Options 1 and 2 call for manual delivery of a key. For link encryption, this is a

reasonable requirement, because each link encryption device is going to be exchanging data only with its partner on the other end of the link. However, for end-to-end encryption over a network, manual delivery is awkward.

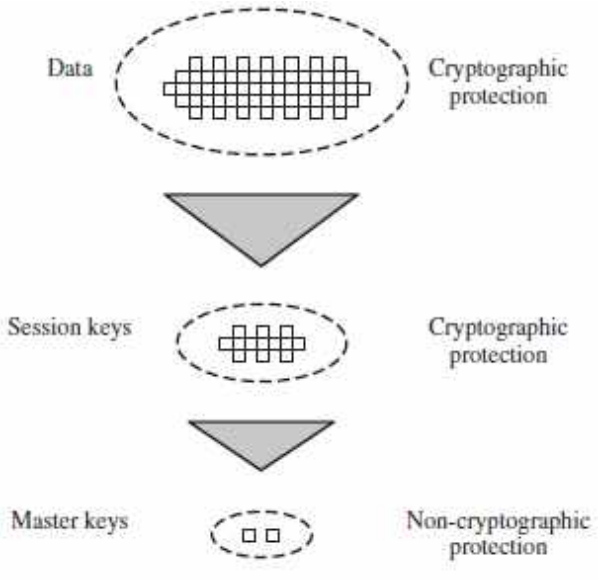
In a distributed system, any given host or terminal may need to engage in exchanges with many other hosts and terminals over time. Thus, each device needs a number of keys supplied dynamically. The problem is especially difficult in a wide-area distributed system.

option 3 is a possibility for either link encryption or end-to-end encryption, but if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed.

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

The use of a key distribution center is based on the use of a hierarchy of keys.

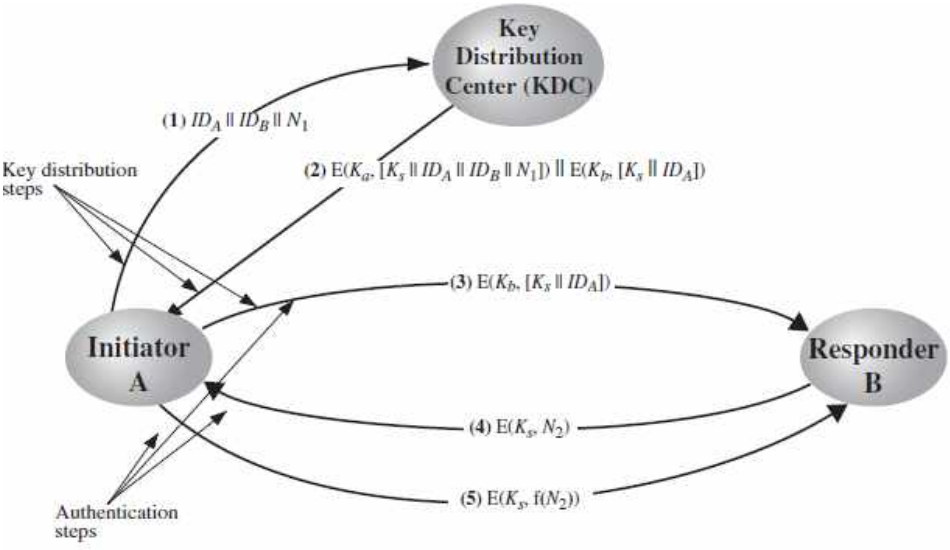
At a minimum, two levels of keys are used as shown below:



Communication between end systems is encrypted using a temporary key, often referred to as a session key. Typically, the session key is used for the duration of a logical connection, such as a frame relay connection or transport connection, and then discarded. Each session key is obtained from the key distribution center.

Session keys are transmitted in encrypted form, using a master key that is shared by the key distribution center and an end system or user. For each end system or user, there is a unique master key that it shares with the key distribution center.

A Key Distribution Scenario: A typical scenario is illustrated in Figure below:



The scenario assumes that each user shares a unique master key with the key distribution center (KDC).

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

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.**

Note: 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.

2. The KDC responds with a message encrypted using K_a. The message includes two items intended for A:
    - The one-time session key, K_s , to be used for the session.
    - The original request message, including the nonce, to enable A to match this response with the appropriate request.

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

In addition, the message includes two items intended for B:

- The one-time session key, K_s , to be used for the session
- An identifier of A (e.g., its network address), ID_A

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

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

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

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

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

Hierarchical Key Control: It is not necessary to limit the key distribution function to a single KDC, for very large networks, it may not be practical to do so.As an alternative, a hierarchy of KDCs can be established.

For example, there can be local KDCs, each responsible for a small domain of the overall internetwork, such as a single LAN or a single building. For communication among entities within the same local domain, the local KDC is responsible for key distribution. If two entities in different domains desire a shared key, then the corresponding local KDCs can communicate through a global KDC. In this case, any one of the three KDCs involved can actually select the key.

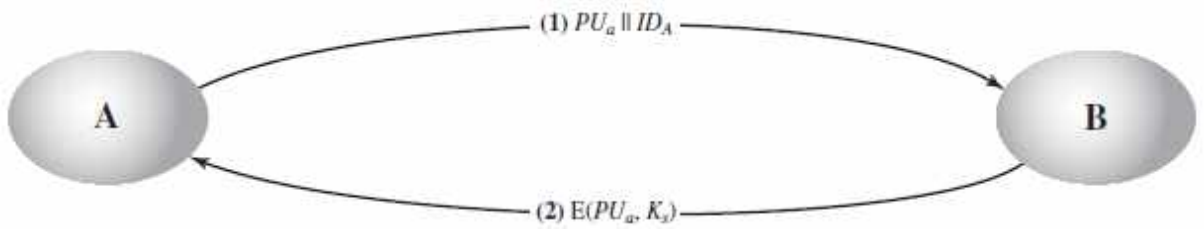
Session Key Lifetime: A **session key** is a single-use **key** used for encrypting all messages in one communication **session**.

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

For a connectionless protocol, such as a transaction-oriented protocol, there is no explicit connection initiation or termination. The most secure approach is to use a new session key for each exchange.

SYMMETRIC KEY DISTRIBUTION USING ASYMMETRIC ENCRYPTION: One of the most important uses of a public-key cryptosystem is to encrypt secret keys for distribution.

Simple Secret Key Distribution: Simple Secret Key Distribution illustrated in Figure below:



If A wishes to communicate with B, the following procedure is employed:

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

A and B can now securely communicate using conventional encryption and the session key K_s . At the completion of the exchange, both A and B discard K_s.

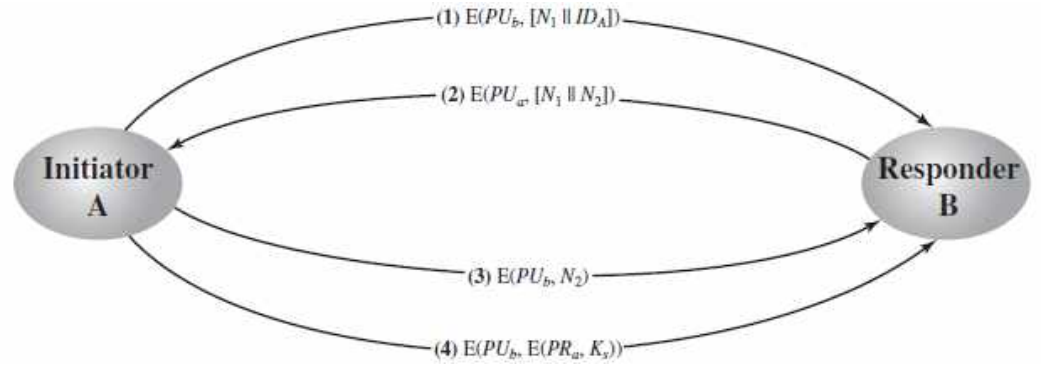
The protocol is insecure depicted in Figure above,

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

The result is that both A and B know K_s and are unaware that K_s has also been revealed to E.

Secret Key Distribution with Confidentiality

and Authentication: Protocol shown in Figure below, provides protection against both active and passive attacks.



Assume that A and B have exchanged public keys by one of the schemes. Then the following steps occur.

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

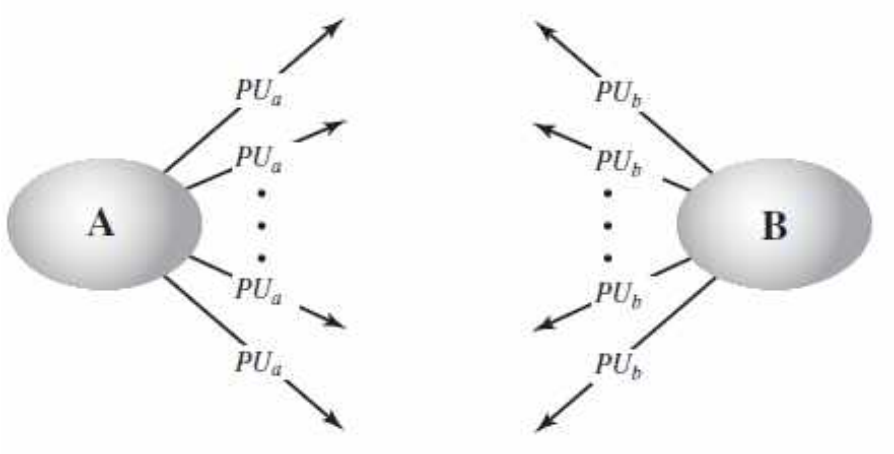
The result is that this scheme ensures both confidentiality and authentication in the exchange of a secret key.

A Hybrid Scheme: Another way to use public-key encryption to distribute secret keys is a hybrid approach in use on IBM mainframes. 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.

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

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

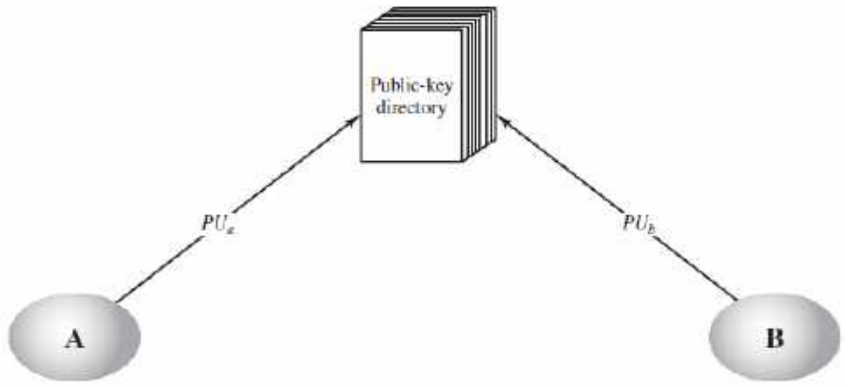
Public Announcement of Public Keys: Any participant can send his or her public key to any other participant or broadcast the key to the community at large as shown below:



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

Publicly Available Directory: A greater degree of security can be achieved by maintaining a publicly available dynamic directory of public keys.

Maintenance and distribution of the public directory would have to be the responsibility of some trusted entity or organization



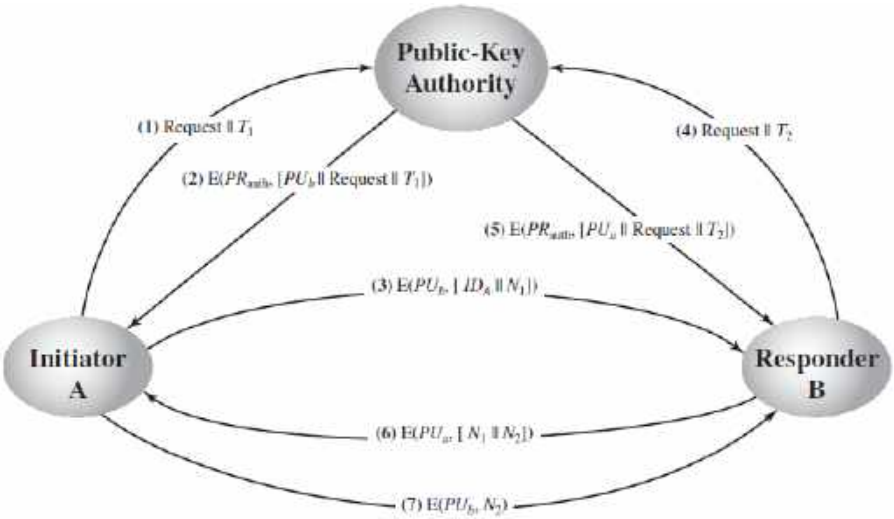
Publicly Available Directory scheme would include the following elements:

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

This scheme is clearly more secure than individual public announcements but still has vulnerabilities. If an adversary succeeds in obtaining or computing the

private key of the directory authority, the adversary could authoritatively pass out counterfeit public keys and subsequently impersonate any participant and eavesdrop on messages sent to any participant.

Public-Key Authority: Stronger security for public-key distribution can be achieved by providing tighter control over the distribution of public keys from the directory. A typical scenario is illustrated in Figure



The following steps occurs:

1. A sends a time-stamped 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:
    a) B's public key, PU_b which A can use to encrypt messages destined for B.
    b) The original request used to enable A to match this response with the corresponding earlier request and to verify that the original request was not altered before reception by the authority.
    c) The original timestamp given so A can determine that this is not an old message from the authority containing a key other than B's current public key.
3. A stores B's public key and also uses it to encrypt a message to B containing an identifier of A(ID_A) and a nonce (N_1), which is used to identify this transaction uniquely.
4. 4, 5. 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
7. 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.
8. A returns N_2, which is encrypted using B's public key, to assure B that its correspondent is A.

Public-Key Certificates: Public-Key Distribution Scenario is attractive , yet it has some drawbacks. The public-key authority could be somewhat of a bottleneck in the system, for a user must appeal to the authority for a public key for every other user that it wishes to contact.

The directory of names and public keys maintained by the authority is vulnerable to tampering.

An alternative approach is to use **certificates** that can be used by participants to exchange keys without contacting a public-key authority.

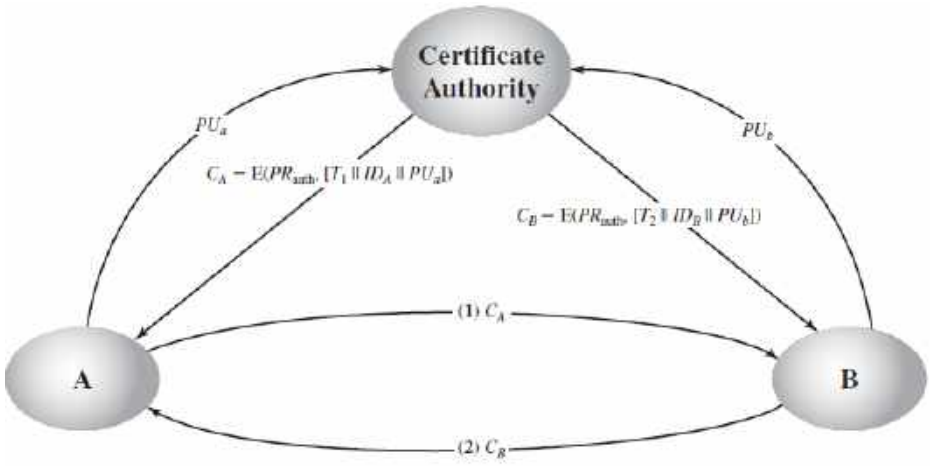
In essence, a certificate consists of a public key, an identifier of the key owner, and the whole block signed by a trusted third party. Typically, the third party is a certificate authority, such as a government agency or a financial institution.

A user can present his or her public key to the authority in a secure manner and obtain a certificate.

The following are the 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.

A certificate scheme is illustrated in Figure below:



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

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

Where PR_{auth} is the private key used by the authority and T is a time stamp.

A may then pass this certificate on to any other participant, who reads and verifies the certificate as follows:

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

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

Because the certificate is readable only using the authority's public key, this

verifies that the certificate came from the certificate authority.

The elements ID_A and PU_a provide the recipient with the name and public key of the certificate's holder. The timestamp T validates the currency of the certificate.

X.509 CERTIFICATES

ITU-T recommendation X.509 is part of the X.500 series of recommendations that define a directory service.

The directory is, in effect, a server or distributed set of servers that maintains a database of information about users.

The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users.

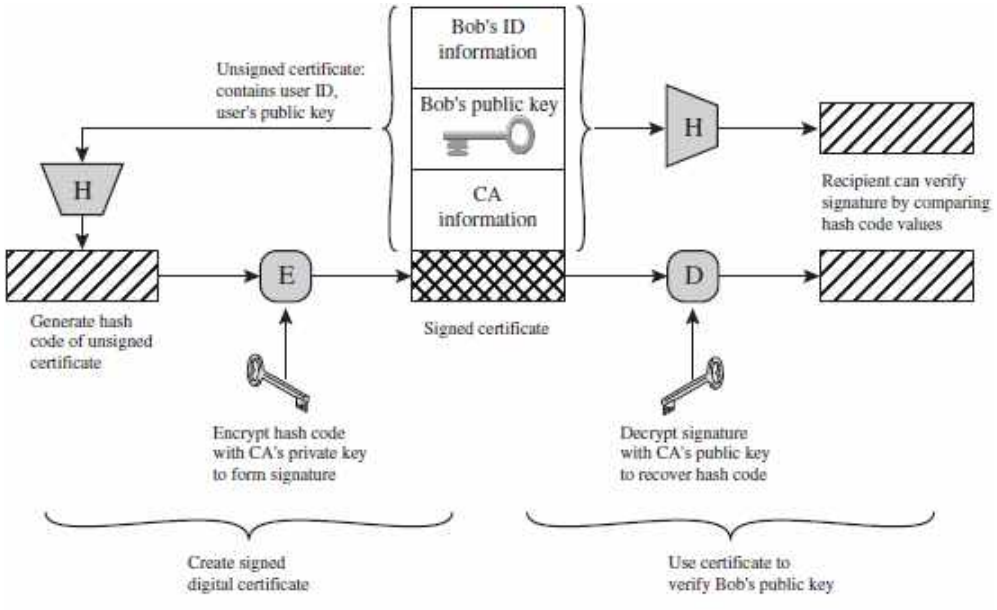
The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority.

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts.

For example, X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS

X.509 was initially issued in 1988. The standard was subsequently revised to address some of the security concerns and a revised recommendation was issued in 1993. A third version was issued in 1995 and revised in 2000.

X.509 is based on the use of public-key cryptography and digital signatures. Following Figure illustrates Public-Key Certificate Use



Certificates: The heart of the X.509 scheme is the public-key certificate associated with each user. These user certificates are assumed to be created by some trusted certification authority (CA) and placed in the directory by the CA or by the user.

The directory server itself is not responsible for the creation of public keys or for the certification function; it merely provides an easily accessible location for users to obtain certificates.

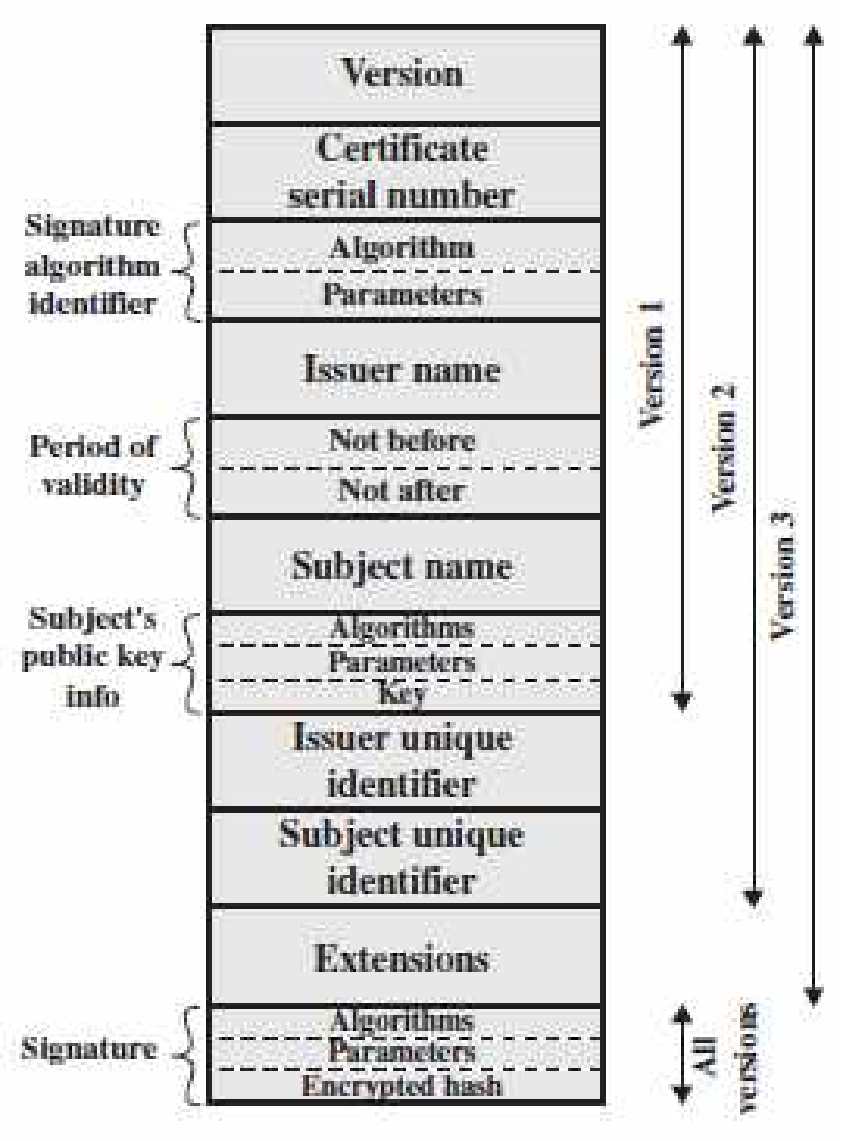
Figure below shows the general format of a certificate:

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.



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.

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 standard uses the following notation to define a certificate:

CA<<A>> = 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

The CA signs the certificate with its private key. If the corresponding public

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

OBTAINING A USER'S CERTIFICATE: User certificates generated by a CA have the following characteristics:

• Any user with access to the public key of the CA can verify the user public key that was certified.

• No party other than the certification authority can modify the certificate without this being detected.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. Thus, with many users, it may be more practical for there to be a number of CAs.

Now suppose that A has obtained a certificate from certification authority X_1 and B has obtained a certificate from CA X_2

A has used a chain of certificates to obtain B's public key. In the notation of

X.509, this chain is expressed as

X1<<X2>>X2<<B>>

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

X2<<X1>>X1<<A>>

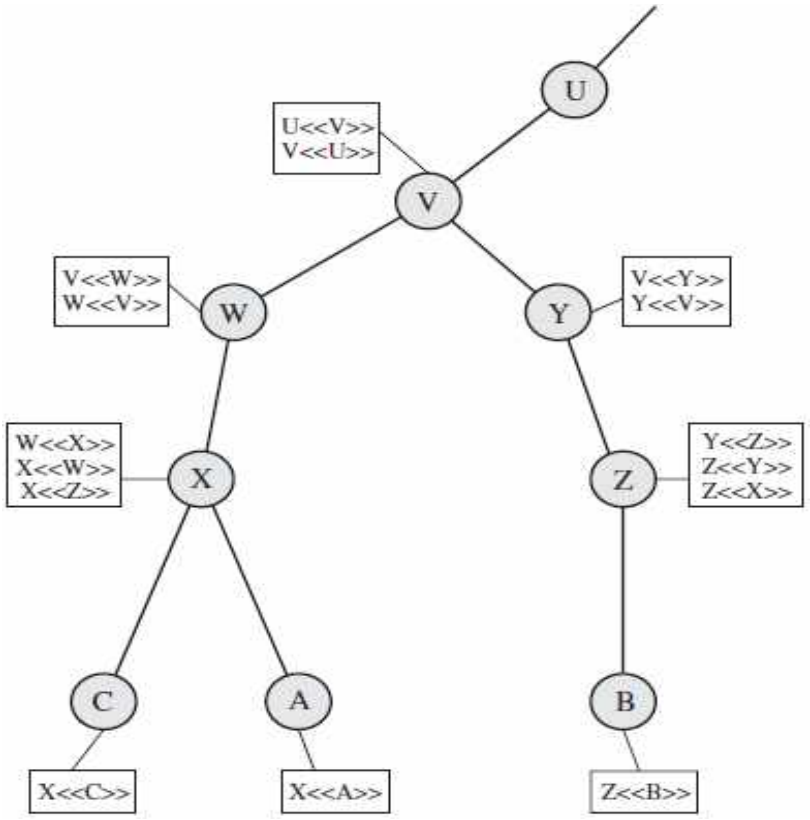
This scheme need not be limited to a chain of two certificates. An arbitrarily long path of CAs can be followed to produce a chain. A chain with N elements would be expressed as:

X1<<X2>>X2<<X3>>…X_N<<B>>

In this case, each pair of CAs in the chain must have created certificates for each other.

Figure below, taken from X.509, is an example of such a hierarchy. In this example, user A can acquire the following certificates from the directory to establish a certification path to B:

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

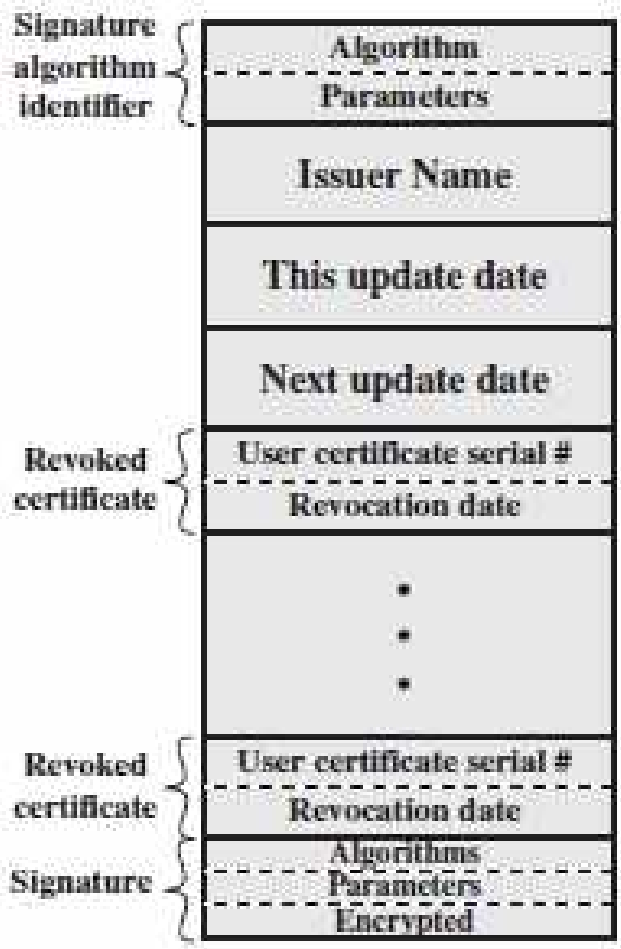


REVOCATION OF CERTIFICATES: *E*ach certificate includes a period of validity, much like a credit card. Typically, a new certificate is issued just before the expiration of the old one. T revoke a certificate before it expires, for one of the following reasons.

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

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

The following Figure shows certificate revocation list:



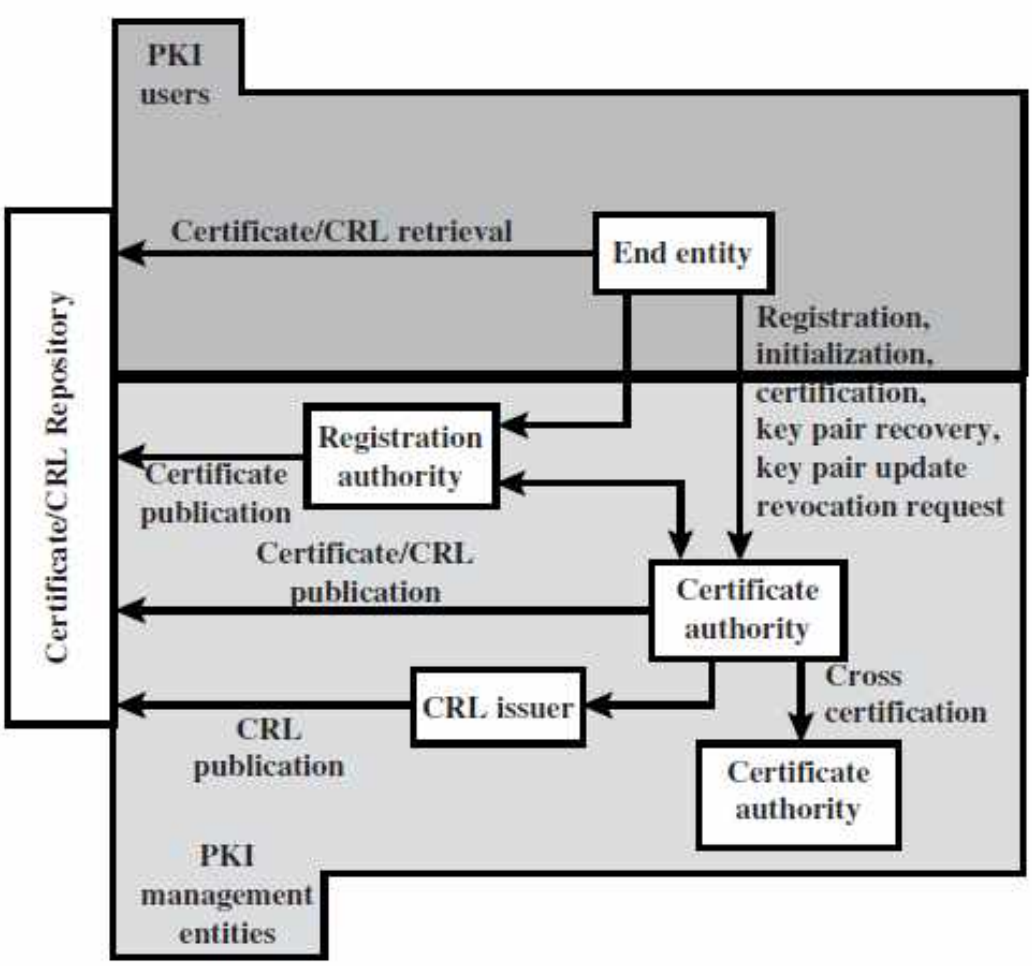
PUBLIC-KEY INFRASTRUCTURE

RFC 2822 (*Internet Security Glossary) defines public-key infrastructure (PKI) as* the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography.

The principal objective for developing a PKI is to enable secure, convenient, and efficient acquisition of public keys.

The Internet Engineering Task Force (IETF) Public Key Infrastructure X.509 (PKIX) working group has been the driving force behind setting up a formal (and generic) model based on X.509 that is suitable for deploying a certificate-based architecture on the Internet.

The following Figure shows PKIX Architectural Model and the interrelationship among the key elements of the PKIX model



The elements are:

End entity: A generic term used to denote end users, devices (e.g., servers, routers), or any other entity that can be identified in the subject field of a public key certificate.

Certification authority (CA): The issuer of certificates and (usually) certificate revocation lists (CRLs).

Registration authority (RA): An optional component that can verifies the identity of entities requesting their digital certificates to be stored at the CA.

CRL issuer: An optional component that a CA can delegate to publish CRLs.

Repository: A generic term used to denote any method for storing certificates and CRLs so that they can be retrieved by end entities.

PKIX Management Functions: PKIX identifies a number of management functions that potentially need to be supported by management protocols.

1) Registration: Registration begins the process of enrolling in a PKI. Registration usually involves some offline or online procedure for mutual authentication. Typically, the end entity is issued one or more shared secret keys used for subsequent authentication.

2) Initialization: Before a client system can operate securely, it is necessary to install key materials that have the appropriate relationship with keys stored elsewhere in the infrastructure.

3) Certification: This is the process in which a CA issues a certificate for a user's public key, returns that certificate to the user's client system, and/or posts that certificate in a repository.

4) Key pair recovery: Key pair recovery allows end entities to restore their encryption/decryption key pair from an authorized key backup facility (typically, the CA that issued the end entity's certificate).

5) Key pair update: All key pairs need to be updated regularly (i.e., replaced with a new key pair) and new certificates issued. Update is required when the certificate lifetime expires.

6) Revocation request: An authorized person advises a CA of an abnormal situation requiring certificate revocation. Reasons for revocation include private key compromise, change in affiliation, and name change.

7) Cross certification: Two CAs exchange information used in establishing a cross-certificate. A cross-certificate is a certificate issued by one CA to another CA that contains a CA signature key used for issuing certificates.

PKIX Management Protocols: The PKIX working group has defines two alternative management protocols between PKIX entities.

RFC 2510 defines the certificate management protocols (CMP), each of the management functions is explicitly identified by specific protocol exchanges.

RFC 2797 defines certificate management messages over CMS (CMC), where

CMS refers to RFC 2630, cryptographic message syntax.

USER AUTHENTICATION

User authentication enable communicating parties to satisfy themselves about identity.

2.1 REMOTE USER-AUTHENTICATION PRINCIPLES

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

Identification step: Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)

Verification step: Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

There are four general means of authenticating a user's identity, which can be

used alone or in combination:

Something the individual knows: Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.

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

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

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

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

Mutual Authentication: Protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form.

Replay attacks:

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

Repetition that can be logged: An opponent can replay a time stamped message within the 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 the replay 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 of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

Timestamps: Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment. This approach requires that clocks among the various participants be synchronized.

Challenge/response: Party A, expecting a fresh message from B, first sends B a nonce (challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

One-Way Authentication: One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail

message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that

the message can be handled by the store-and-forward e-mail protocol.

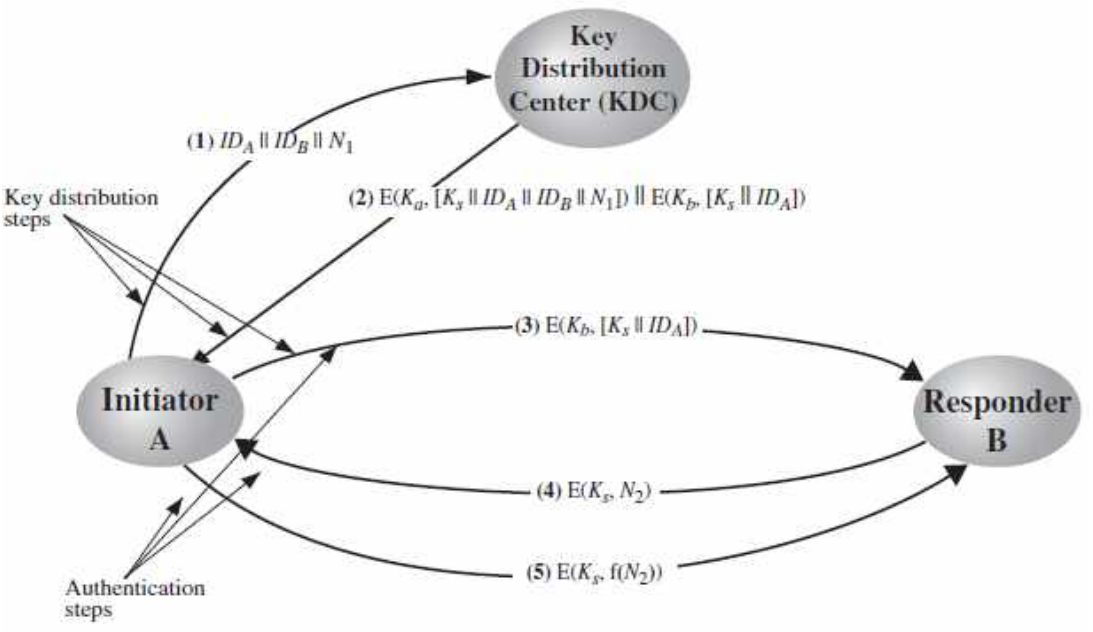
However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

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

REMOTE USER-AUTHENTICATION USING SYMMETRIC ENCRYPTION: Secret keys Ka and Kb 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 Ks to A and B.

Mutual Authentication: A two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment.

Figure below illustrates secret key distribution using a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC.



The protocol can be summarized as follows.

1. $A \rightarrow KDC: \quad ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s \| ID_A])$
4. $B \rightarrow A: \quad E(K_s, N_2)$
5. $A \rightarrow B: \quad E(K_s, f(N_2))$

The purpose of the protocol is to distribute securely a session key Ks to A and B. A securely acquires a new session key in step 2.The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of Ks and assures B that this is a fresh message because of the use of the nonce N_2. The purpose of steps 4 and 5 is to prevent a certain type of replay attack.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3.

X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay.

To overcome this weakness by a modification to the protocol that includes the addition of a timestamp to steps 2 and 3 , and it consists of the following steps.

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

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

$$|Clock - T| < \Delta t_1 + \Delta t_2$$

Where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source.

The above protocol still has risk. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.

The problem occurs when a sender's clock is ahead of the intended recipient's

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

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for clock synchronization, is to rely on handshaking protocols using nonces. The protocol is:

1. $A \rightarrow B: \quad ID_A \| N_a$
2. $B \rightarrow KDC: \quad ID_B \| N_b \| E(K_b, [ID_A \| N_a \| T_b])$
3. $KDC \rightarrow A: \quad E(K_a, [ID_B \| N_a \| K_s \| T_b]) \| E(K_b, [ID_A \| K_s \| T_b]) \| N_b$
4. $A \rightarrow B: \quad E(K_b, [ID_A \| K_s \| T_b]) \| E(K_s, N_b)$

The following protocol ensues subsequent authentication to B, avoiding the need to contact the authentication server repeatedly.

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

One-Way Authentication: In case email we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content **M** , the sequence is as follows:

1. $A \rightarrow KDC: \quad ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: \quad E(K_a, [K_s \| ID_B \| N_1 \| E(K_b, [K_s \| ID_A])])$
3. $A \rightarrow B: \quad E(K_b, [K_s \| ID_A]) \| E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be

able to read it. It also provides a level of authentication that the sender is A.

KERBEROS: Kerberos4 is an authentication service developed as part of Project Athena at MIT.

Assume an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. In this environment, a workstation cannot be trusted to identify its users correctly to network services. In particular, the following three threats exist:

1. A user may gain access to a particular workstation and pretend to be another user operating from that workstation.
2. A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.
3. A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

In any of these cases, an unauthorized user may be able to gain access to services and data that he or she is not authorized to access.

Kerberos is a network authentication protocol. It is designed to provide strong authentication for client/server applications by using cryptography. Kerberos relies exclusively on symmetric encryption, making no use of public-key encryption.

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

The first published report on Kerberos listed the following requirements.

Secure: Kerberos should be strong enough so that a network eavesdropper should not be able to obtain the necessary information to impersonate a user.

Reliable: Kerberos should be highly reliable and should employ a distributed server architecture with one system able to back up another for availability of the Kerberos service.

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.

Kerberos Version 4: Version 4 of Kerberos makes use of DES to provide the authentication service.

A SIMPLE AUTHENTICATION DIALOGUE: In an unprotected network environment, any client can apply to any server for service. The obvious security risk is that of impersonation. An opponent can pretend to be another client and obtain unauthorized privileges on server machines.

Each server can be required to undertake this task for each client/server interaction, but in an open environment, this places a substantial burden on each server.

An alternative is to use an authentication server (AS) that knows the passwords of all users and stores these in a centralized database and AS shares a unique secret key with each server.

Consider the following hypothetical dialogue:

where

C =client;

AS = authentication server;

V =server;

ID_C = *identifier of user on C;*

ID_V= Identifier of V; P_C = Password of user C

AD_C = network address of C

K_V= secret encryption key shared by AS and V

In this scenario, the user logs on to a workstation and requests access to server V.

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

A more secure authentication dialogue

There are two major problems associated with the previous approach:

1. Plaintext transmission of the password.

2. Each time a user has to enter the password.

To solve these problems, we introduce a scheme for avoiding plaintext passwords, and anew server, known as ticket granting server (TGS). The hypothetical scenario is as follows:

Once per user logon session:

(1) $C \rightarrow AS: \quad ID_C \| ID_{tgs}$

(2) $AS \rightarrow C: \quad E(K_c, Ticket_{tgs})$

Once per type of service:

(3) $C \rightarrow TGS: \quad ID_C \| ID_V \| Ticket_{tgs}$

(4) $TGS \rightarrow C: \quad Ticket_v$

Once per service session:

(5) $C \rightarrow V: \quad ID_C \| Ticket_v$

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

1. The client requests a ticket-granting ticket on behalf of the user by sending its user's ID and password to the AS, together with the TGS ID, indicating a request to use the TGS service.
2. The AS responds with a ticket that is encrypted with a key that is derived from the user's password.

When this response arrives at the client, the client prompts the user for his or her password, generates the key, and attempts to decrypt the incoming message. If the correct password is supplied, the ticket is successfully recovered.

Now that the client has a ticket-granting ticket, access to any server can be

obtained with steps 3 and 4:

1. The client requests a service-granting ticket on behalf of the user. For this purpose, the client transmits a message to the TGS containing the user's ID, the ID of the desired service, and the ticket-granting ticket.
2. The TGS decrypts the incoming ticket and verifies the success of the decryption by the presence of its ID. It checks to make sure that the lifetime has not expired. Then it compares the user ID and network address with the incoming information to authenticate the user. If the user is permitted access to the server V, the TGS issues a ticket to grant access to the requested service.

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

1. The client requests access to a service on behalf of the user. For this purpose, the client transmits a message to the server containing the user's ID and the service-granting ticket. The server authenticates by using the contents of the ticket.

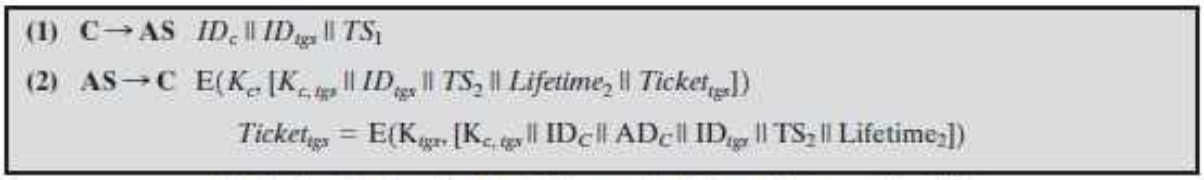
This new scenario satisfies the two requirements of only one password query per user session and protection of the user password.

THE VERSION 4 AUTHENTICATION DIALOGUE: Two additional problems remain in the more secure authentication dialogue:

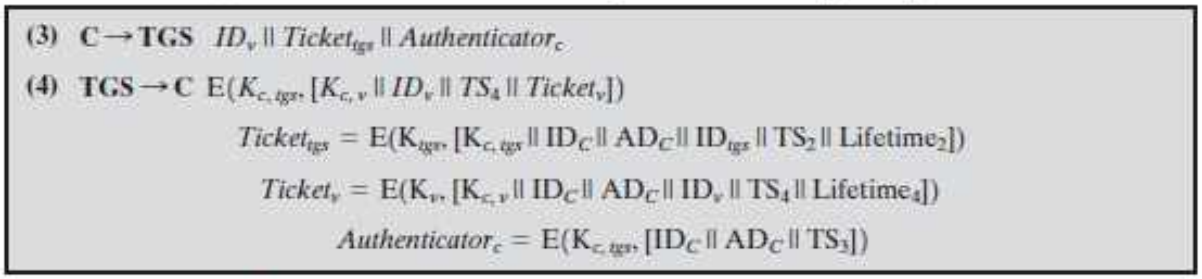
- Lifetime associated with the ticket granting ticket. If the lifetime is very short, then the user will be repeatedly asked for a password. If the lifetime is long, then the opponent has the greater opportunity for replay.

• Requirement for the servers to authenticate themselves to users.

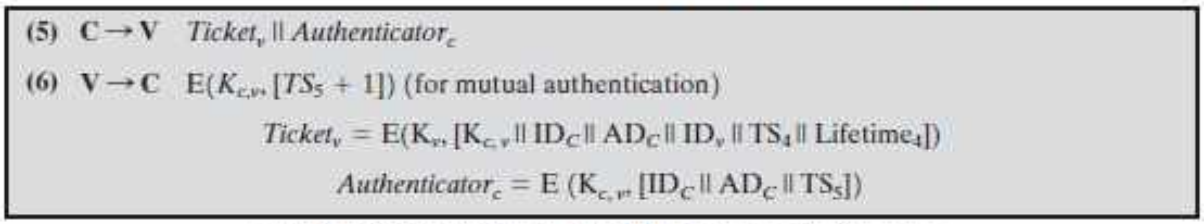
Table below Summary of Kerberos Version 4 Message Exchanges:



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

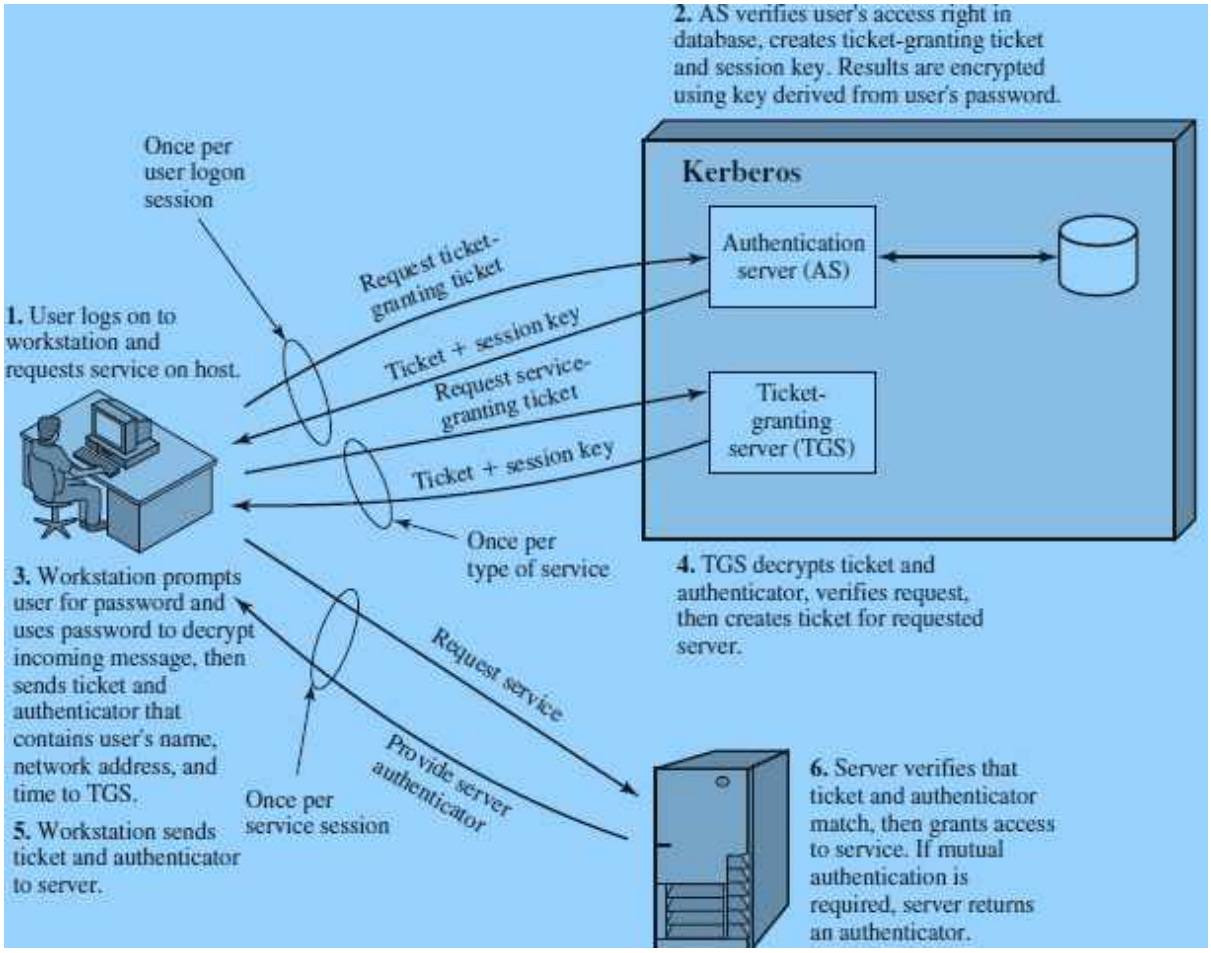


(b) Ticket-Granting Service Exchange to obtain service-granting ticket



(c) Client/Server Authentication Exchange to obtain service

Figure below shows Overview of Kerberos

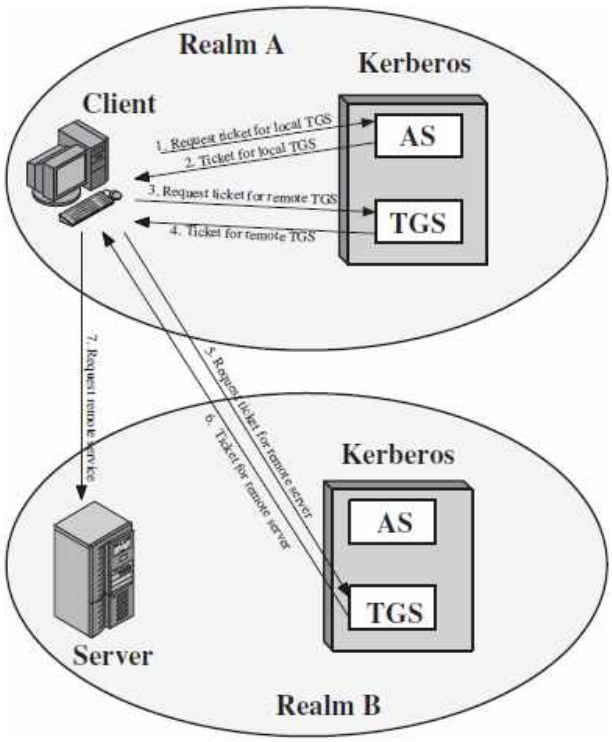


Kerberos Realms and Multiple Kerberi : A full-service Kerberos environment consisting of a Kerberos server, a number of clients, and a number of application servers requires the following:

1. The Kerberos server must have the user ID and hashed passwords of all participating users in its database. All users are registered with the Kerberos server.
2. The Kerberos server must share a secret key with each server. All servers are registered with the Kerberos server.

Such an environment is referred to as a **Kerberos realm.**

The concept of *realm can be explained as follows.*



The details of the exchanges illustrated in the Figure above 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 ticket presented to the remote server (V_{rem}) indicates the realm in which the user was originally authenticated.

Kerberos version 5

Version 5 of Kerberos provides a number of improvements over version 4.

• developed in mid 1990's

• provides improvements over v4

– addresses environmental shortcomings

– and technical deficiencies

• specified as Internet standard RFC 1510

Differences between version 4 and 5

Version 5 is intended to address the limitations of version 4 in two areas:

Environmental shortcomings

o encryption system dependence

o internet protocol dependence o

message byte ordering

o ticket lifetime

o authentication forwarding

o inter-realm authenticaiton

Technical deficiencies

o double encryption

o PCBC (Plaintext Cipher Block Chaining) encryption

o Session keys

o Password attacks

In **PCBC** mode, each block of plaintext is XORed with both the previous plaintext block and the previous ciphertext block before being **encrypted**.

REMOTE USER AUTHENTICATION USING

ASYMMETRIC ENCRYPTION

Mutual Authentication: This protocol assumes that each of the two parties is in possession of the current public key of the other.

A protocol using timestamps is provided in:

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

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

This protocol is compact but, as before, requires the synchronization of clocks.

Another approach, proposed by Woo and Lam, makes use of nonces.

The protocol consists of the following steps.

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. $A \rightarrow B: \quad E(K_s, N_b)$

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

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

One-Way Authentication

Public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality, authentication, or both.

If confidentiality is the primary concern, then the following may be more efficient:

$$A \rightarrow B: \quad E(PU_b, K_s) \| E(K_s, M)$$

If authentication is the primary concern, then a digital signature

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

both the message and signature can be encrypted with the recipient's public key:

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

An effective way to provide this assurance is the digital certificate

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

Electronic mail security

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.

With the explosively growing reliance on e-mail, there grows a demand for authentication and confidentiality services.

Two schemes stand out as approaches that enjoy widespread use: Pretty Good Privacy (PGP) and S/MIME.

PRETTY GOOD PRIVACY(PGP): PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications.

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.
4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

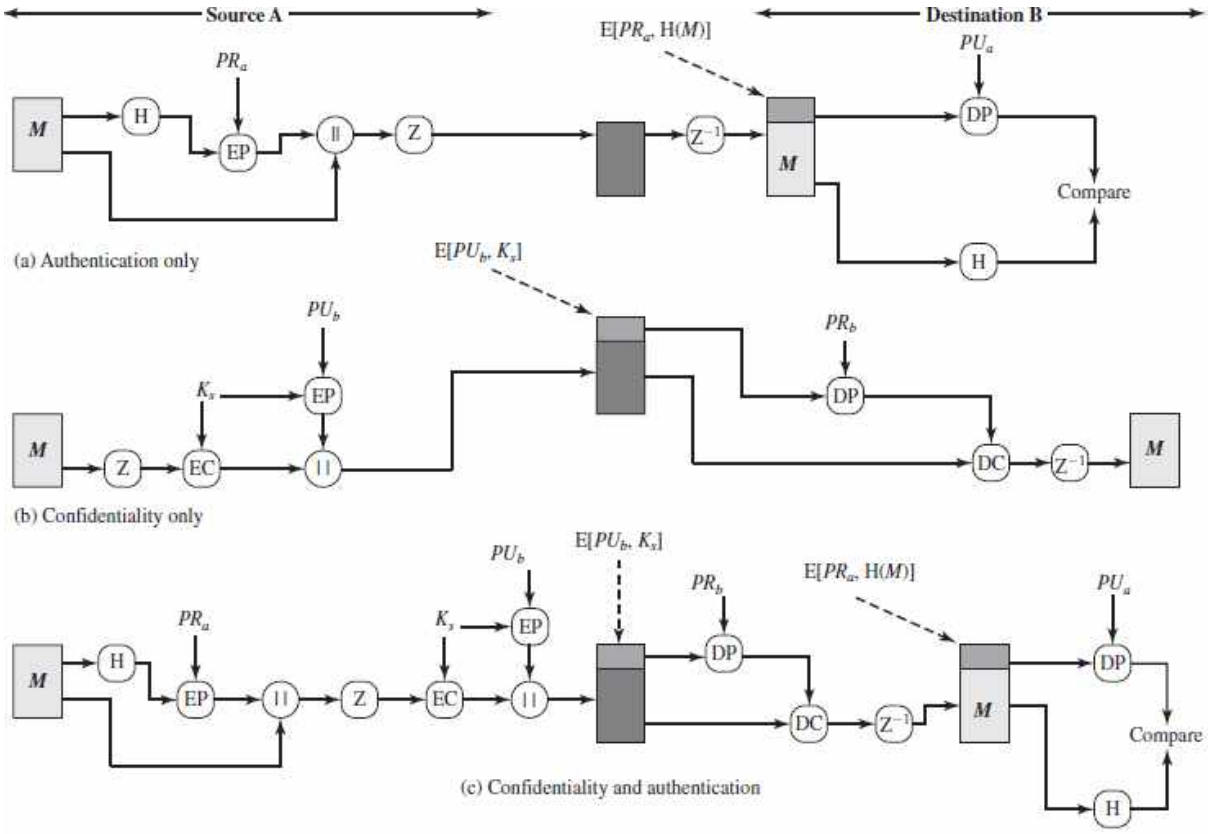
Notation: The following symbols are used in PGP.

H = hash function
|| = concatenation
Z = compression using ZIP algorithm
R64 = conversion to radix 64 ASCII format
K_s = session key used in symmetric encryption scheme
PR_a = private key of user A, used in public-key encryption scheme
PU_a = public key of user A, used in public-key encryption scheme
EP = public-key encryption
DP = public-key decryption
EC = symmetric encryption
DC = symmetric decryption

Operational Description: The actual operation of PGP consists of four services: authentication, confidentiality, compression, and e-mail compatibility. The following table summarizes PGP services:

Function	Algorithms Used	Description
Digital signature	DSS/SHA or RSA/SHA	A hash code of a message is created using SHA-1. This message digest is encrypted using DSS or RSA with the sender's private key and included with the message.
Message encryption	CAST or IDEA or Three-key Triple DES with Diffie-Hellman or RSA	A message is encrypted using CAST-128 or IDEA or 3DES with a one-time session key generated by the sender. The session key is encrypted using Diffie-Hellman or RSA with the recipient's public key and included with the message.
Compression	ZIP	A message may be compressed for storage or transmission using ZIP.
E-mail compatibility	Radix-64 conversion	To provide transparency for e-mail applications, an encrypted message may be converted to an ASCII string using radix-64 conversion.

The following figure shows PGP Cryptographic Functions:



AUTHENTICATION: Figure (a) Authenticaton Only illustrates the digital signature service provided by PGP. 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.

CONFIDENTIALITY: Another basic service provided by PGP is confidentiality, encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used. The 64-bit cipher feedback (CFB) mode is used.

Figure (b)Confidentiality illustrates the sequence as follows:

1. 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 AND AUTHENTICATION: As Figure (c)Confidentiality and Authentication illustrates, both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. 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).

COMPRESSION: As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression

and Z–1 for decompression

E-MAIL COMPATIBILITY:

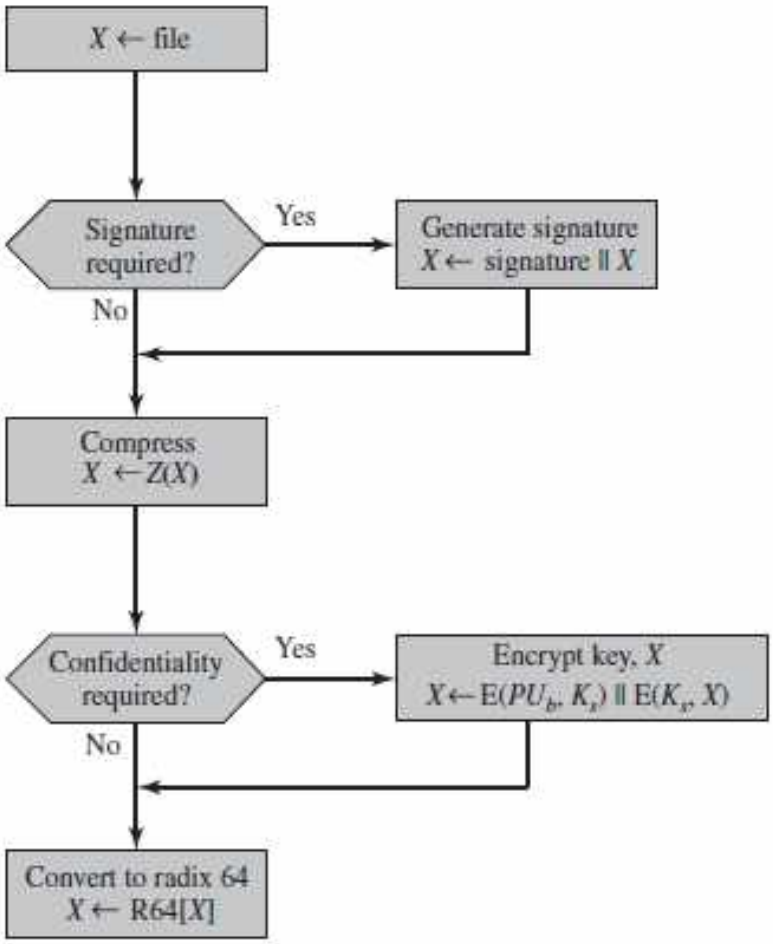
When PGP is used, at least part of the block to be transmitted is encrypted. If only the signature service is used, then the message digest is encrypted (with the sender's private key). If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets. However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.

The scheme used for this purpose is radix-64 conversion. Each group of three

octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors.

The use of radix 64 expands a message by 33%. Fortunately, the session key and signature portions of the message are relatively compact, and the plaintext message has been compressed.

Following Figure shows generic transmission of PGP Messages:



Following Figure shows generic Reception of PGP Messages:

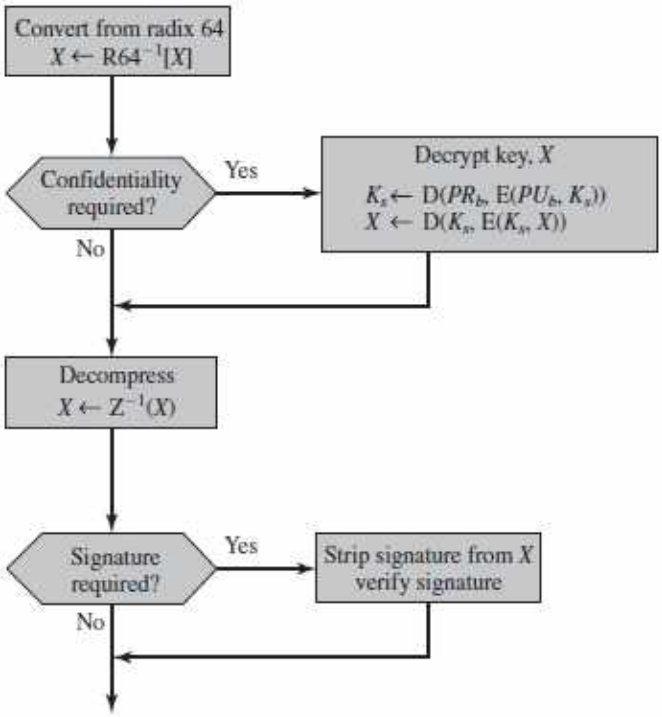
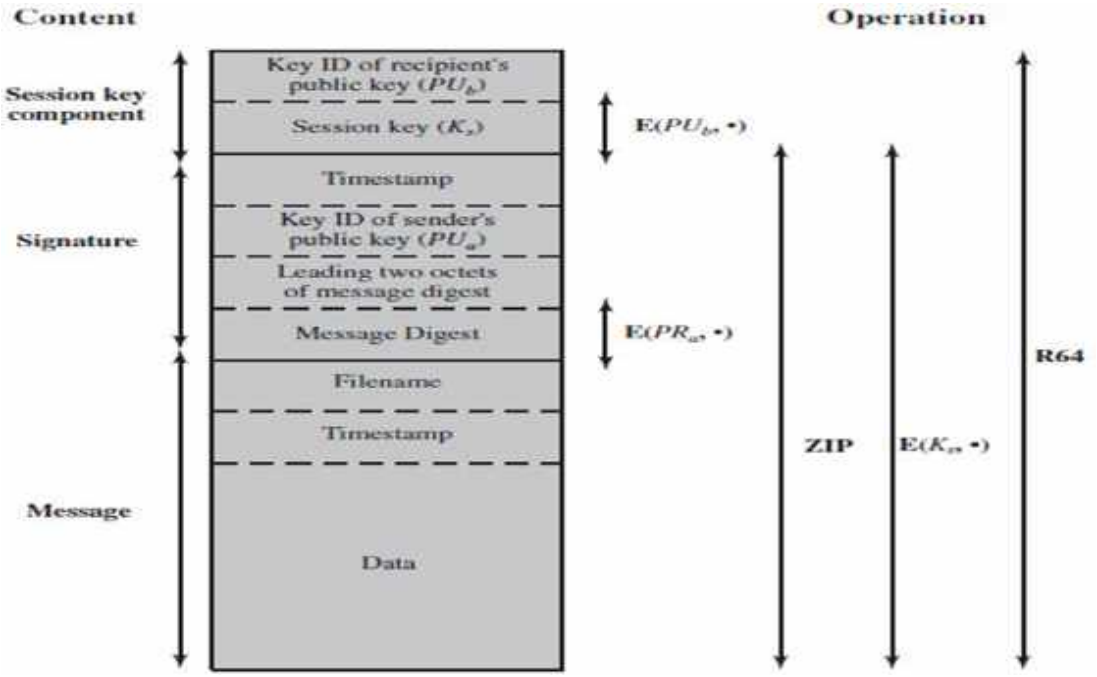


Figure below shows General Format PGP Message (from A to B)



Notation:

$E(PU_b, \bullet)$ = encryption with user b's public key
$E(PR_a, \bullet)$ = encryption with user a's private key
$E(K_s, \bullet)$ = encryption with session key
ZIP = Zip compression function
R64 = Radix-64 conversion function

The message component includes the actual data to be stored or transmitted, as well as a filename and a timestamp that specifies the time of creation.

The signature component includes the following.

• Timestamp: The time at which the signature was made.

• Message digest: The 160-bit SHA-1 digest encrypted with the sender's private signature key.

• Leading two octets of message digest: Enables the recipient to determine if

the correct public key was used to decrypt the message digest for authentication by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.

• Key ID of sender's public key: Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

S/MIME: Secure/Multipurpose Internet Mail Extension (S/MIME) is a security enhancement to the MIME Internet e-mail format standard based on technology from RSA Data Security.

RFC 5322: RFC 5322 defines a format for text messages that are sent using electronic mail.

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 usually 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:

Date: October 8, 2009 2:15:49 PM EDT

From: William Stallings <ws@shore.net>

Subject: The Syntax in RFC 5322

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): MIME is an extension to the RFC 5322 framework that is intended to address some of the problems and limitations of the use of Simple Mail Transfer Protocol (SMTP).

The following limitations of the SMTP/5322 scheme.

1. SMTP cannot transmit executable files or other binary objects.

2. SMTP cannot transmit text data that includes national language characters, because these are represented by 8-bit codes.
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. Etc…

OVERVIEW: The MIME specification includes the following elements.

1. Five new message header fields are defined, which may be included in an RFC 5322 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.

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).

MIME CONTENT TYPES

The following table shows 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.

Figure shows Example for MIME Message Structure

```
--unique-boundary-1
Content-type: text/plain; charset=US-ASCII

This could have been part of the previous part, but illustrates explicit versus implicit typing of body parts.

--unique-boundary-1
Content-Type: multipart/parallel;   boundary=unique-boundary-2

--unique-boundary-2
Content-Type: audio/basic
Content-Transfer-Encoding: base64

    ... base64-encoded 8000 Hz single-channel mu-law-format audio data goes here....

--unique-boundary-2
Content-Type: image/jpeg
Content-Transfer-Encoding: base64

    ... base64-encoded image data goes here....

--unique-boundary-2--

--unique-boundary-1
Content-type: text/enriched

This is <bold><italic>richtext.</italic></bold> <smaller>as defined in RFC 1896</smaller>

Isn't it <bigger><bigger>cool?</bigger></bigger>

--unique-boundary-1
Content-Type: message/rfc822

From: (mailbox in US-ASCII)
To: (address in US-ASCII)
Subject: (subject in US-ASCII)
Content-Type: Text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: Quoted-printable

    ... Additional text in ISO-8859-1 goes here ...

--unique-boundary-1--
```

S/MIME Functions

enveloped data

encrypted content and associated keys

signed data

encoded message + signed digest

clear-signed data

cleartext message + encoded signed digest

signed & enveloped data

nesting of signed & encrypted entities

S/MIME Cryptographic Algorithms

digital signatures: DSS & RSA

hash functions: SHA-1 & MD5

session key encryption: ElGamal & RSA

message encryption: AES, Triple-DES, RC2/40 and others

MAC: HMAC with SHA-1

have process to decide which algorithm to use

S/MIME Messages

S/MIME secures a MIME entity with a signature, encryption, or both

forming a MIME wrapped PKCS object

have a range of content-types:

enveloped data

signed data

clear-signed data

registration request

certificate only message

S/MIME Certificate Processing

S/MIME uses X.509 v3 certificates

managed using a hybrid of a strict X.509 CA hierarchy & PGP's web of trust

each client has a list of trusted CA's certificates

and own public/private key pairs & certificates

certificates must be signed by trusted CA's

Certificate Authorities

have several well-known CA's

Verisign one of most widely used

Verisign issues several types of Digital IDs

increasing levels of checks & hence trust

Class	**Identity**	**Checks**	**Usage**
1	name/email check	web browsing/email	
2	+ enroll/addr check	email, subs, s/w validate	

3	+ ID documents	e-banking/service access

S/MIME Enhanced Security Services

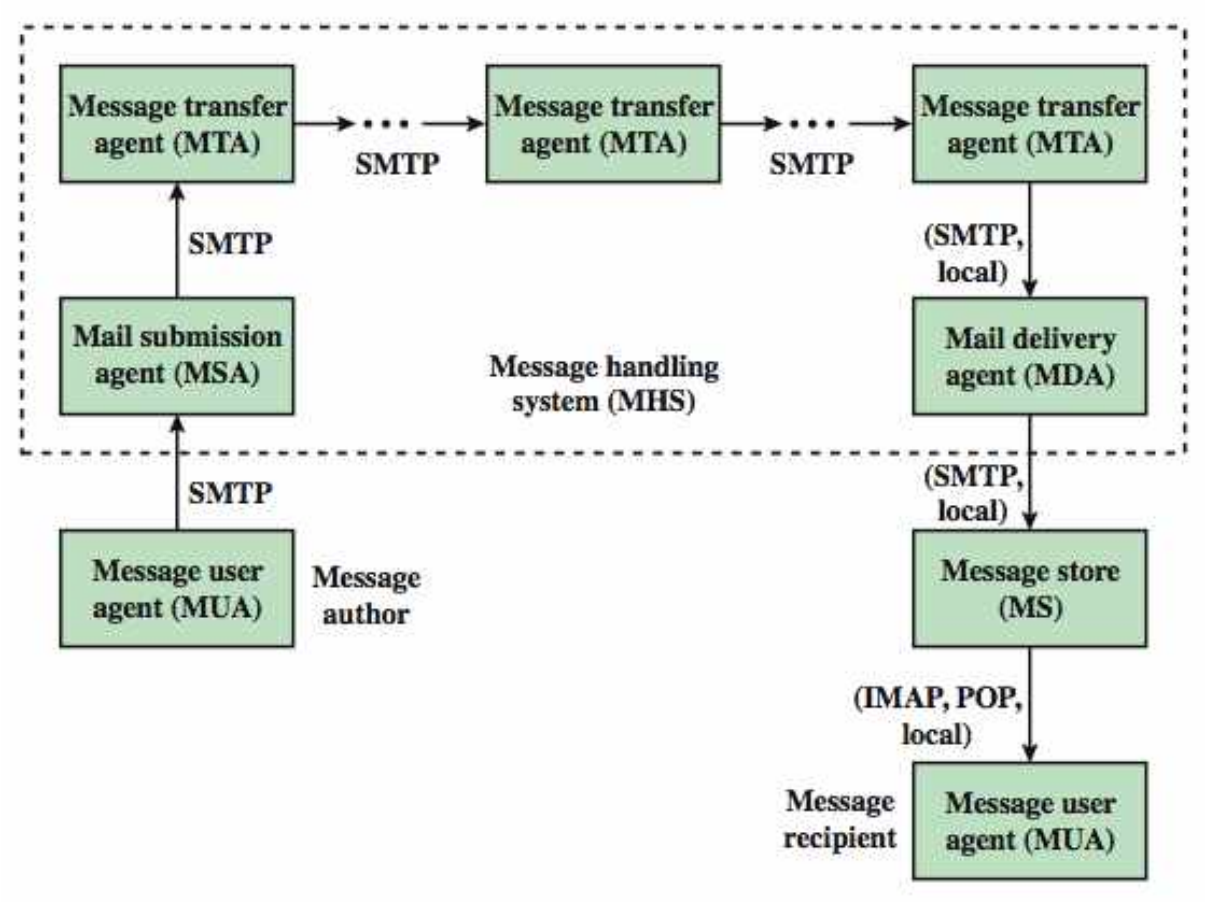
3 proposed enhanced security services:

signed receipts

security labels

secure mailing lists

Internet Mail Architecture



Email Threats

Malware. ...

Spam and phishing. ...

Social engineering. ...

Entities with malicious intent. ...

Unintentional acts by authorized users.