# UNIT-4

## AUTHENTICATION SERVICES KERBEROS

Kerberos provides a centralized authentication server whose function is to authenticate users to servers and servers to users. Kerberos relies exclusively on conventional encryption, making no use of public-key encryption.

The following are the requirements for Kerberos:

· **Secure:** A network eavesdropper should not be able to obtain the necessary information to impersonate a user. More generally, Kerberos should be strong enough that a potential opponent does not find it to be the weak link.

· **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, the overall scheme of Kerberos is that of a trusted third- party authentication service that uses a protocol based on that proposed by Needham and Schroeder [NEED78] It is trusted in the sense that clients and servers trust Kerberos to mediate their mutual authentication. Assuming the Kerberos protocol is well designed, then the authentication service is secure if the Kerberos server itself is secure.

**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. To counter this threat, servers must be able to confirm the identities of clients who request service. 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. In addition, the AS shares a unique secret key with each server. The simple authentication dialogue is as follows:

**A more secure authentication dialogue**

There are two major problems associated with the previous

approach: Plaintext transmission of the password.

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:**

C >> AS: IDc||IDtgs

AS >> C: Ekc (Tickettgs)

**Once per type of service:**

C >> TGS: IDc||IDv||Tickettgs

TGS >> C: ticketv

**Once per service session:**

5. C >> V: IDc||ticketv

Tickettgs= Ektgs(IDc||ADc||IDtgs||TS1||Lifetime1) Ticketv= Ekv(IDc||ADc||IDv||TS2||Lifetime2)

C: Client, AS: Authentication Server, V: Server, IDc : ID of the client, Pc:Password of the client, ADc: Address of client, IDv : ID of the server, Kv: secret key shared by AS and V, ||: concatenation, IDtgs: ID of the TGS server, TS1, TS2: time stamps, lifetime: lifetime of the ticket.

The new service, TGS, issues tickets to users who have been authenticated to AS. Thus, the user first requests a ticket-granting ticket (Tickettgs) from the AS. The client module in the user workstation saves this ticket. Each time the user requires access to a new service, the client applies to the TGS, using the ticket to authenticate itself. The TGS then grants a ticket for the particular service. The client saves each service-granting ticket and uses it to authenticate its user to a server each time a particular service is requested. Let us look at the details of this scheme:

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.

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.Because only the correct user should know the password, only the correct user can recover the ticket. Thus, we have used the password to obtain credentials from Kerberos without having to transmit the password in plaintext.

Now that the client has a ticket-granting ticket, access to any server can be obtained with steps 3 and 4:

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.

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.

The service-granting ticket has the same structure as the ticket-granting ticket. Indeed, because the TGS is a server, we would expect that the same elements are needed to authenticate a client to the TGS and to authenticate a client to an application server. Again, the ticket contains a timestamp and lifetime. If the user wants access to the same service at a later time, the client can simply use the previously acquired service-granting ticket and need not bother the user for a password. Note that the ticket is encrypted with a secret key (Kv) known only to the TGS and the server, preventing alteration.

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

**5.** 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 nd 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.

### 4.1.1 Kerbero V4 Authentication Dialogue Message Exchange

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. The actual Kerberos protocol version 4 is as follows:

a basic third-party authentication scheme

have an Authentication Server (AS)

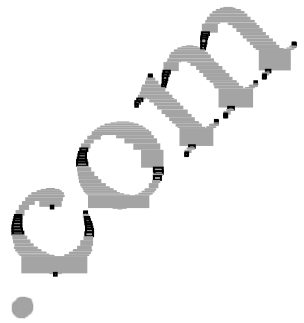
–users initially negotiate with AS to identify self

–AS provides a non-corruptible authentication credential (ticket granting ticket TGT)

have a Ticket Granting server (TGS)

–users subsequently request access to other services from TGS on basis of users TGT

| **Message (1)** | **Client requests ticket-granting ticket** |
|---|---|
| IDC | Tells AS identity of user from this client |
| IDtgs | Tells AS that user requests access to TGS |
| TS1 | Allows AS to verify that client's clock is synchronized with that of AS |
| **Message (2)** | **AS returns ticket-granting ticket** |
| Kc | Encryption is based on user's password, enabling AS and client to verify password, and protecting contents of message (2) |
| Kc,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 |
| IDtgs | Confirms that this ticket is for the TGS |



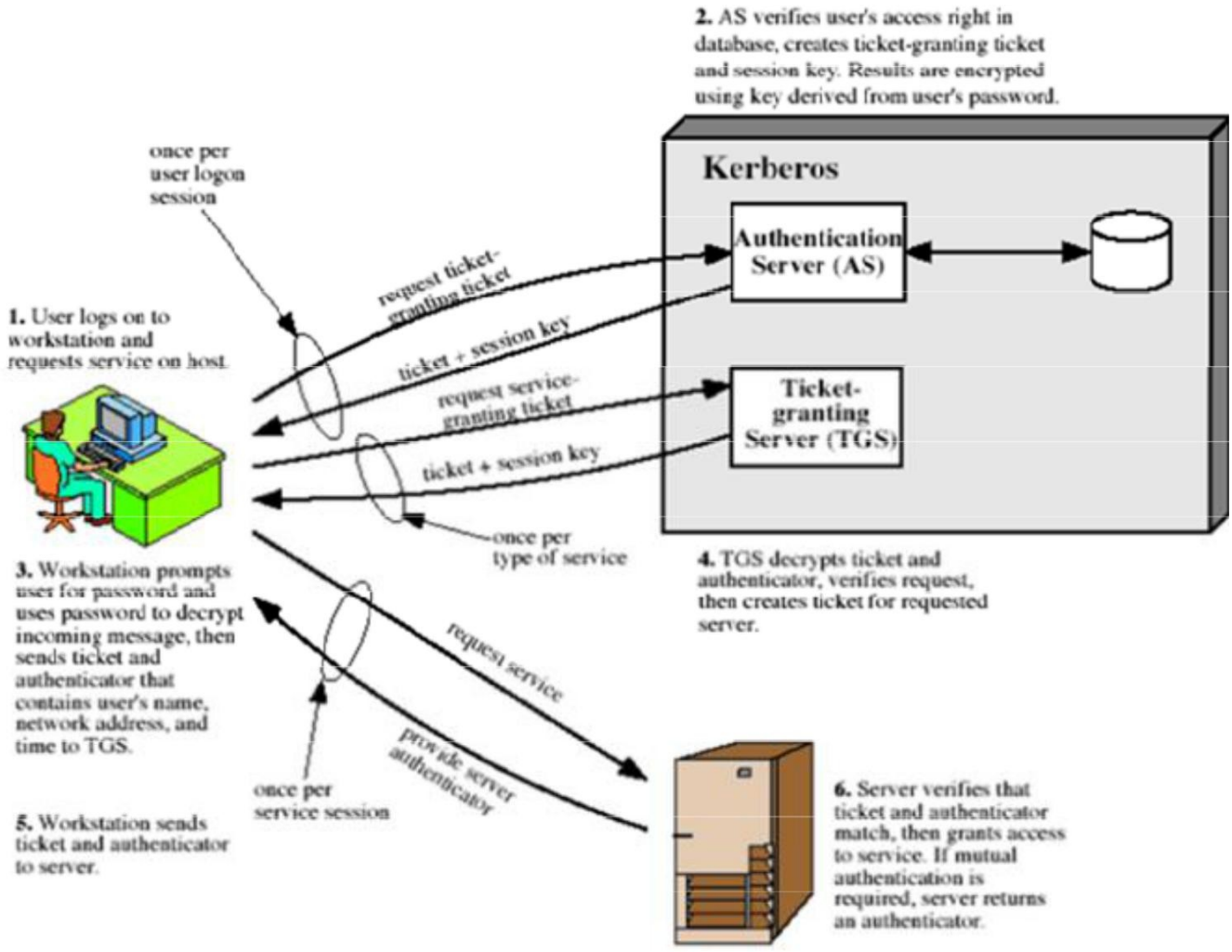
**The table given below illustrates the mode of dialogue in V4**

| **(a) Authentication Service Exchange: to obtain ticket-granting ticket** |
|---|
| **(1) C → AS:** $ID_c \parallel ID_{tgs} \parallel TS_1$ |
| **(2) AS → C:** $\mathrm{E}_{K_c}\left[K_{c,tgs} \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2 \parallel Ticket_{tgs}\right]$ |
| $Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$ |
| **(b) Ticket-Granting Service Exchange: to obtain service-granting ticket** |
| **(3) C → TGS:** $ID_v \parallel Ticket_{tgs} \parallel Authenticator_c$ |
| **(4) TGS → C:** $\mathrm{E}_{K_{c,tgs}}\left[K_{c,v} \parallel ID_v \parallel TS_4 \parallel Ticket_v\right]$ |
| $Ticket_{tgs} = \mathrm{E}_{K_{tgs}}\left[K_{c,tgs} \parallel ID_C \parallel AD_C \parallel ID_{tgs} \parallel TS_2 \parallel Lifetime_2\right]$ |
| $Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$ |
| $Authenticator_c = \mathrm{E}_{K_{tgs}}\left[ID_C \parallel AD_C \parallel TS_3\right]$ |
| **(c) Client/Server Authentication Exchange: to obtain service** |
| **(5) C → V:** $Ticket_v \parallel Authenticator_c$ |
| **(6) V → C:** $\mathrm{E}_{K_{c,v}}\left[TS_5 + 1\right]$ (for mutual authentication) |
| $Ticket_v = \mathrm{E}_{K_v}\left[K_{c,v} \parallel ID_C \parallel AD_C \parallel ID_v \parallel TS_4 \parallel Lifetime_4\right]$ |
| $Authenticator_c = \mathrm{E}_{K_{c,v}}\left[ID_C \parallel AD_C \parallel TS_5\right]$ |

| | |
|---|---|
| TS2 | Informs client of time this ticket was issued |
| Lifetime2 | Informs client of the lifetime of this ticket |
| Tickettgs | Ticket to be used by client to access TGS |
| | (a) Authentication Service Exchange |
| **Message (3)** | **Client requests service-granting ticket** |
| IDV | Tells TGS that user requests access to server V |
| Tickettgs | Assures TGS that this user has been authenticated by AS |
| Authenticatorc | Generated by client to validate ticket |
| **Message (4)** | **TGS returns service-granting ticket** |
| Kc,tgs | Key shared only by C and TGS protects contents of message (4) |
| Kc,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 |
| IDv | Confirms that this ticket is for server V |
| TS4 | Informs client of time this ticket was issued |
| Ticketv | Ticket to be used by client to access server V |
| Tickettgs | Reusable so that user does not have to reenter password |
| Ktgs | Ticket is encrypted with key known only to AS and TGS, to prevent tampering |
| Kc,tgs | Copy of session key accessible to TGS used to decrypt authenticator, thereby authenticating ticket |
| IDC | Indicates the rightful owner of this ticket |
| ADC | Prevents use of ticket from workstation other than one that initially requested the ticket |
| IDtgs | Assures server that it has decrypted ticket properly |
| TS2 | Informs TGS of time this ticket was issued |
| Lifetime2 | Prevents replay after ticket has expired |
| Authenticatorc | 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 |
| Kc,tgs | Authenticator is encrypted with key known only to client and TGS, to prevent tampering |

| | |
|---|---|
| IDc | Must match ID in ticket to authenticate ticket |
| ADc | Must match address in ticket to authenticate ticket |
| TS3 | Informs TGS of time this authenticator was generated |
| | (b) Ticket-Granting Service Exchange |
| **Message (5)** | **Client requests service** |
| Ticketv | Assures server that this user has been authenticated by AS |
| Authenticatorc | Generated by client to validate ticket |
| Message (6) | Optional authentication of server to client |
| Kc,v | Assures C that this message is from V |
| TS5 + 1 | Assures C that this is not a replay of an old reply |
| Ticketv | Reusable so that client does not need to request a new ticket from TGS for each access to the same server |
| Kv | Ticket is encrypted with key known only to TGS and server, to prevent tampering |
| Kc,v | Copy of session key accessible to client; used to decrypt authenticator, thereby authenticating ticket |
| IDC | Indicates the rightful owner of this ticket |
| ADc | Prevents use of ticket from workstation other than one that initially requested the ticket |
| IDv | Assures server that it has decrypted ticket properly |
| TS4 | Informs server of time this ticket was issued |
| Lifetime4 | Prevents replay after ticket has expired |
| Authenticatorc | 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 |
| Kc,v | Authenticator is encrypted with key known only to client and server, to prevent tampering |
| IDC | Must match ID in ticket to authenticate ticket |
| ADc | Must match address in ticket to authenticate ticket |
| TS5 | Informs server of time this authenticator was generated |
| | (c) Client/Server Authentication |

**Kerberos 4 Overview**



### 4.1.2 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. TheKerberosservermusthavetheuserID and hashedpasswordsofall participating usersinitsdatabase.Allusersareregisteredwith the Kerberos server.
2. TheKerberosservermustshareasecretkey with eachserver.Allserversare registered with the Kerberos server.Such an environment is referred to as a **Kerberos realm**.

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

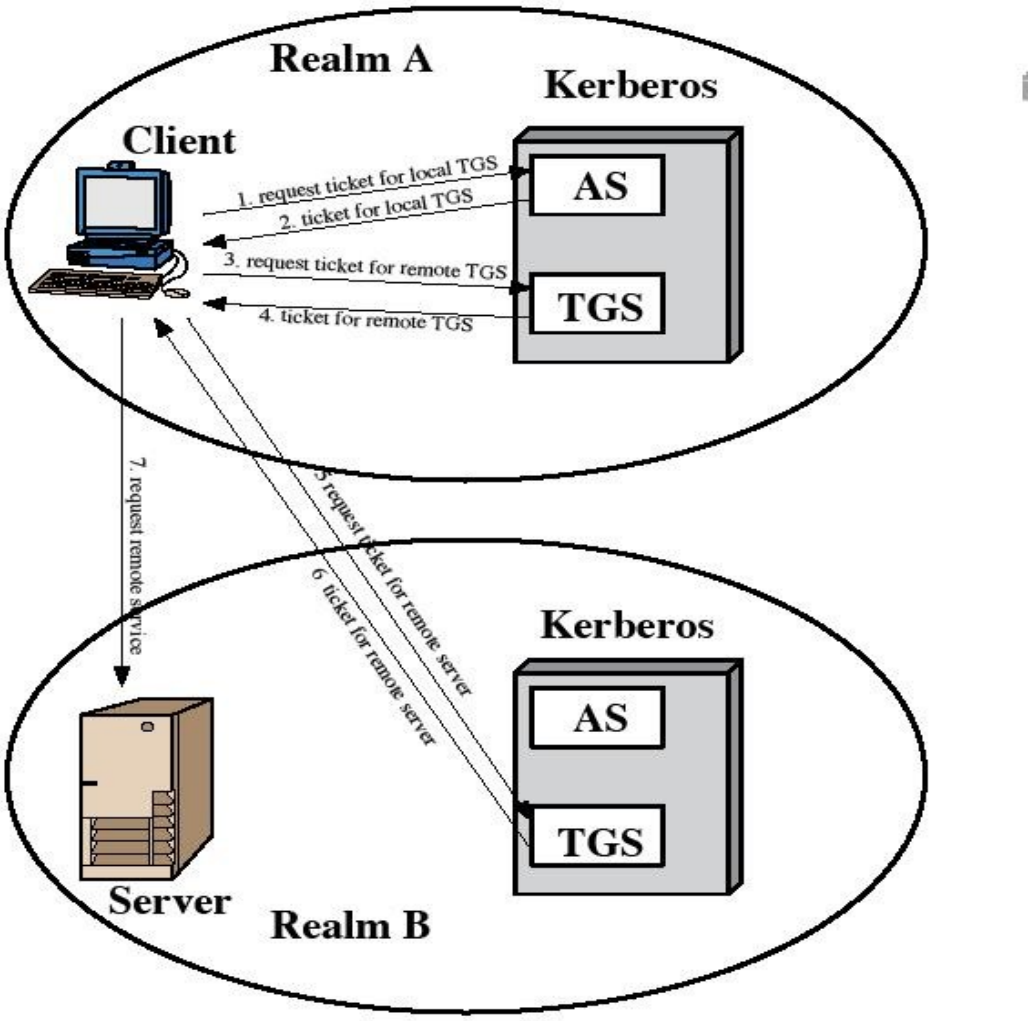



**Figure 14.2 Request for Service in Another Realm**

A Kerberos realm is a set of managed nodes that share the same Kerberos database. The Kerberos

database resides on the Kerberos master computer system, which should be kept in a physically secure room.

A read-only copy of the Kerberos database might also reside on other Kerberos computer systems. However, all changes to the database must be made on the master computer system. Changing or accessing the contents of a Kerberos database requires the Kerberos master password.

A related concept is that of a Kerberos principal, which is a service or user that is known to the Kerberos system.

Each Kerberos principal is identified by its principal name. Principal names consist of three parts: a service or user name, an instance name, and a realm name

Networks of clients and servers under different administrative organizations typically constitute different realms.

That is, it generally is not practical, or does not conform to administrative policy, to have users and servers in one administrative domain registered with a Kerberos server elsewhere.

However, users in one realm may need access to servers in other realms, and some servers may be willing to provide service to users from other realms, provided that those users are authenticated.

Kerberos provides a mechanism for supporting such interrealm authentication. For two realms to support interrealm authentication, a third requirement is added:

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.

The scheme requires that the Kerberos server in one realm trust the Kerberos server in the other realm to authenticate its users. Furthermore, the participating servers in the second realm must also be willing to trust the Kerberos server in the first realm.

**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 encryption o Session keys
- o Password attacks

**The version 5 authentication dialogue**

| **(a) Authentication Service Exchange: to obtain ticket-granting ticket** |
|---|
| **(1) C → AS:** Options \|\| $ID_c$ \|\| $Realm_c$ \|\| $ID_{tgs}$ \|\| $Times$ \|\| $Nonce_1$ |
| **(2) AS → C:** $Realm_c \| ID_C \| Ticket_{tgs} \| E_{K_c}\left[K_{c,tgs} \| Times \| Nonce_1 \| Realm_{tgs} \| ID_{tgs}\right]$ |
| $Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$ |
| **(b) Ticket-Granting Service Exchange: to obtain service-granting ticket** |
| **(3) C → TGS:** Options \|\| $ID_v$ \|\| $Times$ \|\| $Nonce_2$ \|\| $Ticket_{tgs}$ \|\| $Authenticator_c$ |
| **(4) TGS → C:** $Realm_c \| ID_C \| Ticket_v \| E_{K_{c,tgs}}\left[K_{c,v} \| Times \| Nonce_2 \| Realm_v \| ID_V\right]$ |
| $Ticket_{tgs} = E_{K_{tgs}}\left[Flags \| K_{c,tgs} \| Realm_c \| ID_C \| AD_C \| Times\right]$ |
| $Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$ |
| $Authenticator_c = E_{K_{c,tgs}}\left[ID_C \| Realm_c \| TS_1\right]$ |
| **(c) Client/Server Authentication Exchange: to obtain service** |
| **(5) C → V:** Options \|\| $Ticket_v$ \|\| $Authenticator_c$ |
| **(6) V → C:** $E_{K_{c,V}}$ [ $TS_2$ \|\| Subkey \|\| Seq# ] |
| $Ticket_v = E_{K_v}\left[Flags \| K_{c,v} \| Realm_c \| ID_C \| AD_C \| Times\right]$ |
| $Authenticator_c = E_{K_{c,v}}\left[ID_C \| Realm_c \| TS_2 \| Subkey \| Seq\#\right]$ |

irst, consider the authentication service exchange. Message (1) is a client request for a ticket-granting ticket. As before, it includes the ID of the user and the TGS. The following new

elements are added:

Realm: Indicates realm of user

Options: Used to request that certain flags be set in the returned ticket

Times: Used by the client to request the following time settings in the ticket:

from: the desired start time for the requested ticket

till: the requested expiration time for the requested ticket rtime: requested renew-till time

**Nonce**: A random value to be repeated in message (2) to assure that the response is fresh and has not been replayed by an opponent

Message (2) returns a ticket-granting ticket, identifying information for the client, and a block encrypted using the encryption key based on the user's password.
This block includes the session key to be used between the client and the TGS, times specified in message (1), the nonce from message (1), and TGS identifying information.

The ticket itself includes the session key, identifying information for the client, the requested time values, and flags that reflect the status of this ticket and the requested options.

These flags introduce significant new functionality to version 5. For now, we defer a discussion of these flags and concentrate on the overall structure of the version 5 protocol.

Let us now compare the ticket-granting service exchange for versions 4 and 5. We see that message (3) for both versions includes an authenticator, a ticket, and the name of the requested service.

In addition, version 5 includes requested times and options for the ticket and a nonce, all with functions similar to those of message (1).

The authenticator itself is essentially the same as the one used in version 4.

Message (4) has the same structure as message (2), returning a ticket plus information needed by

the client, the latter encrypted with 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 as follows:

**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 use may be sequence numbered to detect replays.

If mutual authentication is required, the server responds with message (6). This message includes the timestamp from the authenticator. Note that in version 4, the timestamp was incremented by one. This is not necessary in version 5 because the nature of the format of messages is such that it is not possible for an opponent to create message (6) without knowledge of the appropriate encryption keys.

***Ticket Flags***

The flags field included in tickets in version 5 supports expanded functionality compared to that available in version 4.

## 4.2 X.509 Certificates

**Overview:**

**issued by a Certification Authority (CA), containing:**

–version (1, 2, or 3)

–serial number (unique within CA) identifying certificate

–signature algorithm identifier

–issuer X.500 name (CA)

–period of validity (from - to dates)

–subject X.500 name (name of owner)

–subject public-key info (algorithm, parameters, key)

–issuer unique identifier (v2+)

–subject unique identifier (v2+)

–extension fields (v3)

–signature (of hash of all fields in certificate)

**notation CA<<A>> denotes certificate for A signed by CA**

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

X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME), IP Security and SSL/TLS and SET

X.509 is based on the use of public-key cryptography and digital signatures. The standard does not

dictate the use of a specific algorithm but recommends RSA. The digital signature scheme is assumed to require the use of a hash function.

**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.

**Version:**

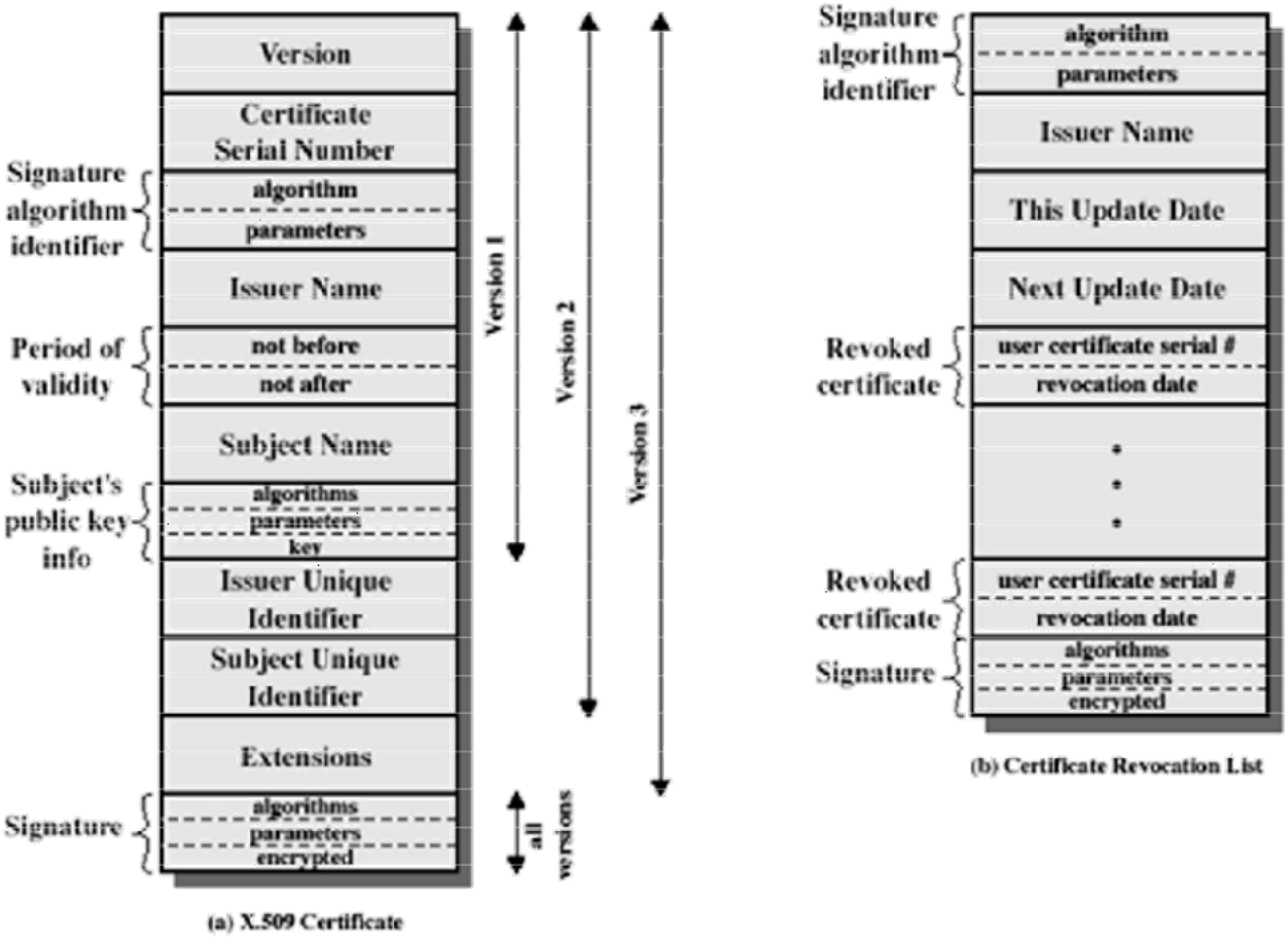
Differentiates among successive versions of the certificate format; the default is version 1. If the Issuer Unique Identifier or Subject Unique Identifier are present, the value must be version 2. If one or more extensions are present, the version must be version 3.

**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 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 identifie

The standard uses the following notation to define a certificate: CA<<A>> = CA {V, SN, AI, CA, TA, A, Ap} 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

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.

ecause 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.

If there is a large community of users, it may not be practical for all users to subscribe to the same CA. Because it is the CA that signs certificates, each participating user must have a copy of the CA's own public key to verify signatures. This public key must be provided to each user in an absolutely secure (with respect to integrity and authenticity) way so that the user has confidence in the associated certificates. Thus, with many users, it may be more practical for there to be a number of CAs, each of which securely provides its public key to some fraction of the users.

Now suppose that A has obtained a certificate from certification authority X1 and B has obtained a certificate from CA X2. If A does not securely know the public key of X2, then B's certificate, issued by X2, is useless to A.

A can read B's certificate, but A cannot verify the signature. However, if the two CAs have securely exchanged their own public keys, the following procedure will enable A to obtain B's public key:

A obtains, from the directory, the certificate of X2 signed by X1. Because A securely knows X1's public key, A can obtain X2's public key from its certificate and verify it by means of X1's signature on the certificate.

A then goes back to the directory and obtains the certificate of B signed by X2
Because A now has a trusted copy of X2's public key, A can verify the signature
and securely obtain B's public key.

A has used a chain of certificates to obtain B's public key. In the notation of X.509, this chain is expressed as

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>>... XN<<B>>

In this case, each pair of CAs in the chain (Xi, Xi+1) must have created certificates for each other.

All these certificates of CAs by CAs need to appear in the directory, and the user needs to know how they are linked to follow a path to another user's public-key certificate. X.509 suggests that CAs be arranged in a hierarchy so that navigation is straightforward.

Figure 14.5, taken from X.509, is an example of such a hierarchy. The connected circles indicate the hierarchical relationship among the CAs; the associated boxes indicate certificates maintained in the directory for each CA entry. The directory entry for each CA includes two types of certificates:

Forward certificates: Certificates of X generated by other CAs

Reverse certificates: Certificates generated by X that are the certificates of other CAs

**CA Hierarchy Use**

In the example given below , user A can acquire the following certificates from the directory to establish a certification path to B:

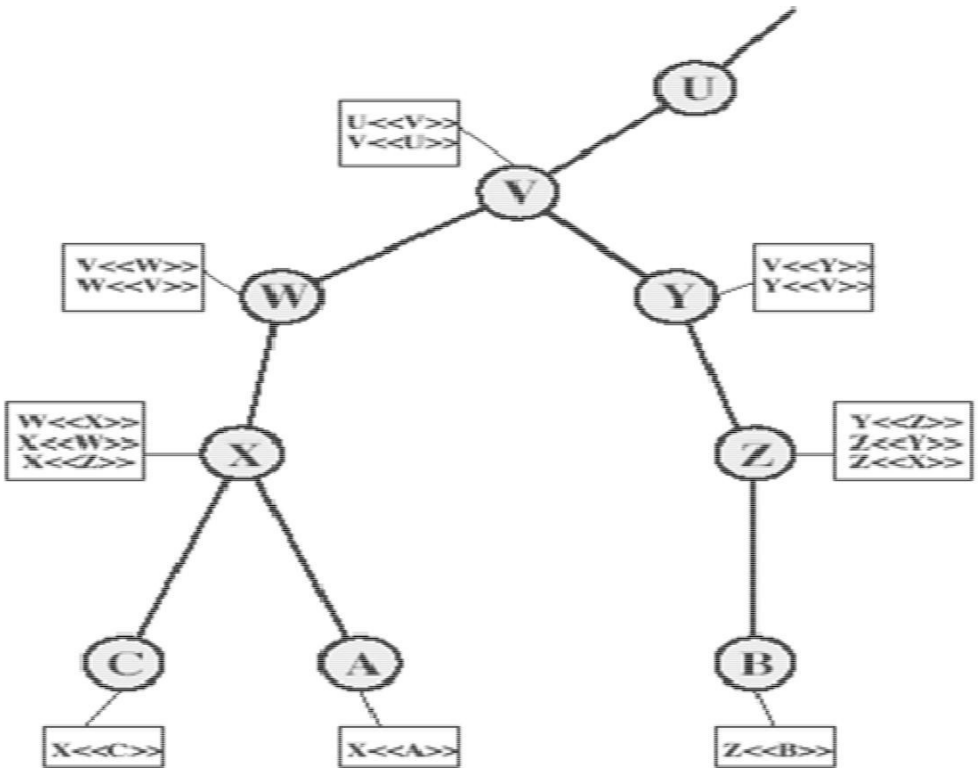
X<<W>> W <<V>> V <<Y>> <<Z>> Z <<B>>

When A has obtained these certificates, it can unwrap the certification path in sequence to recover a trusted copy of B's public key. Using this public key, A can send encrypted

Messages to B. If A wishes to receive encrypted messages back from B, or to sign messages sent to B, then B will require A's public key, which can be obtained from the following certification path:

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

B can obtain this set of certificates from the directory, or A can provide them as part of its initial message to B.




**Certificate Revocation**

Certificates have a period of validity

may need to revoke before expiry, for the following reasons eg: 1.

user's private key is compromised

2.User is no longer certified by this CA

3. CA's certificate is compromised

CA’s maintain list of revoked certificates 1.

the Certificate Revocation List (CRL)

users should check certs with CA’s CRL

**Authentication Procedures**

X.509 includes three alternative authentication procedures:

**One-Way Authentication**

**Two-Way Authentication**

**Three-Way Authentication**

all use public-key signatures

**One-Way Authentication**

1 message ( A->B) used to establish

– the identity of A and that message is from A
– message was intended for B
– integrity & originality of message

• message must include timestamp, nonce, B's identity and is signed by A

**Two-Way Authentication**

• 2 messages (A->B, B->A) which also establishes in addition:

– the identity of B and that reply is from B
– that reply is intended for A
– integrity & originality of reply

reply includes original nonce from A, also timestamp and nonce from B

**Three-Way Authentication**

3 messages (A->B, B->A, A->B) which enables above authentication without synchronized clocks

has reply from a back to B containing signed copy of nonce from B

means that timestamps need not be checked or relied upon

## 4.3 X.509 Version 3

The X.509 version 2 format does not convey all of the information that recent design and implementation experience has shown to be needed. [FORD95] lists the following requirements not satisfied by version 2:

The Subject field is inadequate to convey the identity of a key owner to a public- key user.

The Subject field is also inadequate for many applications, which typically recognize entities by an Internet e-mail address, a URL, or some other Internet- related identification.

There is a need to indicate security policy information. There is a need to limit the damage that can result from a faulty or malicious CA by setting constraints on the applicability of a particular certificate.

It is important to be able to identify different keys used by the same owner at different times.

The certificate extensions fall into three main categories: key and policy information, subject and issuer attributes, and certification path constraints.

### 4.3.1 Key and Policy Information

These extensions convey additional information about the subject and issuer keys, plus indicators of certificate policy.. For example, a policy might be applicable to the authentication of electronic data interchange (EDI) transactions for the trading of goods within a given price range.

This area includes the following:

**Authoritykeyidentifier:** Identifies the public key to be used to verify the signature on this certificate or CRL.

**Subjectkeyidentifier:** Identifies the public key being certified. Useful for subject key pair updating.

**Key usage:** Indicates a restriction imposed as to the purposes for which, and the policies under which, the certified public key may be used.

**Private-key usage period:** Indicates the period of use of the private key corresponding to the public key.. For example, with digital signature keys, the usage period for the signing private key is typically shorter than that for the verifying public key.

**Certificate policies:** Certificates may be used in environments where multiple policies apply.

**Policy mappings:** Used only in certificates for CAs issued by other CAs.

### 4.3.2 Certificate Subject and Issuer Attributes

These extensions support alternative names, in alternative formats, for a certificate subject or certificate issuer and can convey additional information about the certificate subject, to increase a certificate user's confidence that the certificate subject is a particular person or entity. For example, information such as postal address, position within a corporation, or picture image may be required. The extension fields in this area include the following:

**Subject alternative name:** Contains one or more alternative names, using any of a variety of forms

**Subject directory attributes:** Conveys any desired X.500 directory attribute values for the subject of this certificate.

### 4.3.3 Certification Path Constraints

These extensions allow constraint specifications to be included in certificates issued for CAs by other CAs.The extension fields in this area include the following:

**Basic constraints:** Indicates if the subject may act as a CA. If so, a certification path length constraint may be specified.

**Name constraints**: Indicates a name space within which all subject names in subsequent certificates in a certification path must be located.

**Policy constraints**: Specifies constraints that may require explicit certificate policy identification or inhibit policy mapping for the remainder of the certification path.

## 4.4 ELECTRONIC MAIL SECURITY PRETTY GOOD PRIVACY (PGP)

**PGP provides the confidentiality and authentication service that can be used for electronic mail and file storage applications.** The steps involved in PGP are

Select the best available cryptographic algorithms as building blocks.

Integrate 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.

Make the package and its documentation, including the source code, freely available via the internet, bulletin boards and commercial networks.

Enter into an agreement with a company to provide a fully compatible, low cost

commercial version of PGP.

**PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.**

It is available free worldwide in versions that run on a variety of platform.

It is based on algorithms that have survived extensive public review and are considered extremely secure.

e.g., RSA, DSS and Diffie Hellman for public key encryption CAST-128, IDEA and 3DES for conventional encryption SHA-1 for hash coding.

it has a wide range of applicability.

It was not developed by, nor it is controlled by, any governmental or standards organization.

**Operational description**

The actual operation of PGP consists of five services: authentication, confidentiality, compression, e-mail compatibility and segmentation.

**1. Authentication**

The sequence for authentication is as

follows: The sender creates the message

SHA-1 is used to generate a 160-bit hash code of the message

The hash code is encrypted with RSA using the sender's private key and the result is prepended to the message

The receiver uses RSA with the sender's public key to decrypt and recover the hash code.

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.

**2. Confidentiality**

Confidentiality is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the conventional encryption algorithm CAST-128 may be used. The 64-bit cipher feedback (CFB) mode is used.

In PGP, each conventional key is used only once. That is, a new key is generated as a random 128-bit number for each message. Thus although this is referred to as **a session key**, it is in reality a **one time key**. To protect the key, it is encrypted with the receiver's public key.

The sequence for confidentiality is as follows:

The sender generates a message and a random 128-bit number to be used as a session key for this message only.

The message is encrypted using CAST-128 with the session key.

· The session key is encrypted with RSA, using the receiver's public key and is prepended to the message.

· The receiver uses RSA with its private key to decrypt and recover the session key.

The session key is used to decrypt the message.

**Confidentiality and authentication**

Here 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 plus the signature is encrypted using CAST-128 and the session key is encrypted using RSA.

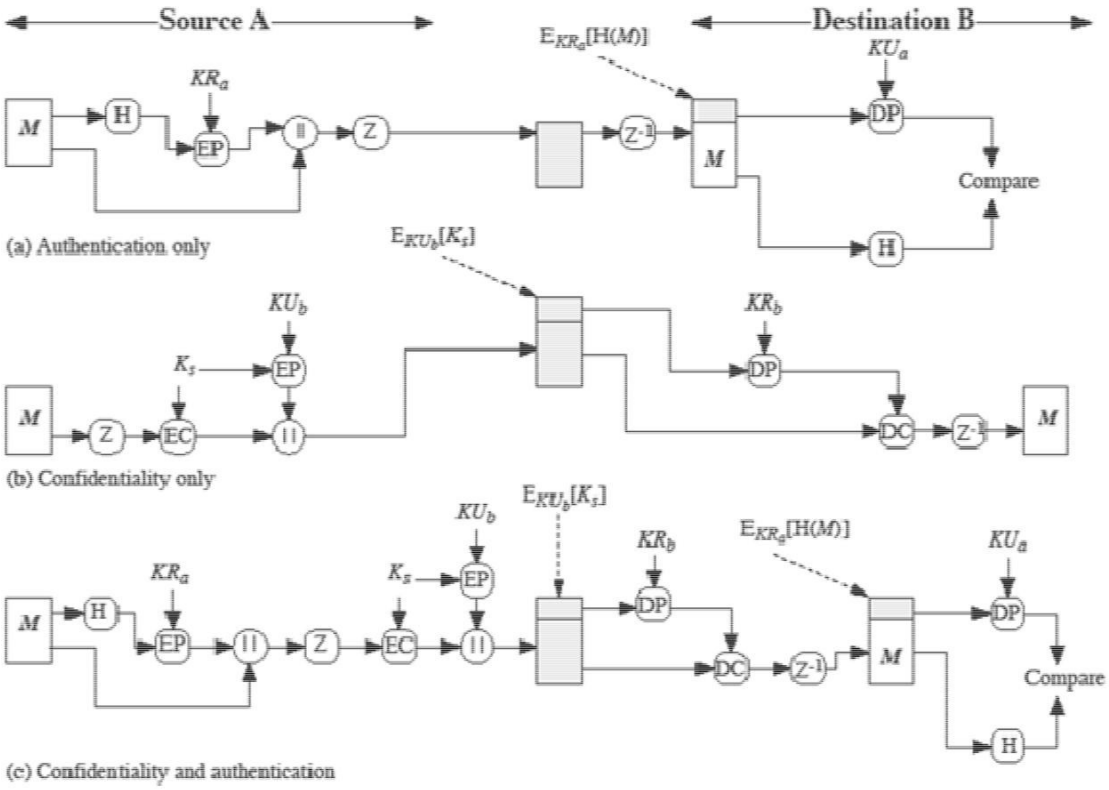



**Figure 15.1 PGP Cryptographic Functions**

**3. Compression**

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

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.

Even if one were willing to generate dynamically a recompressed message fro verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and as a result, produce different compression forms.

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.

**4. e-mail compatibility**

Many electronic mail systems only permit the use of blocks consisting of ASCII texts. 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.

e.g., consider the 24-bit (3 octets) raw text sequence 00100011 01011100 10010001, we can express this input in block of 6-bits to produce 4 ASCII characters.

001000 110101 110010 010001

I L Y R=> corresponding ASCII characters

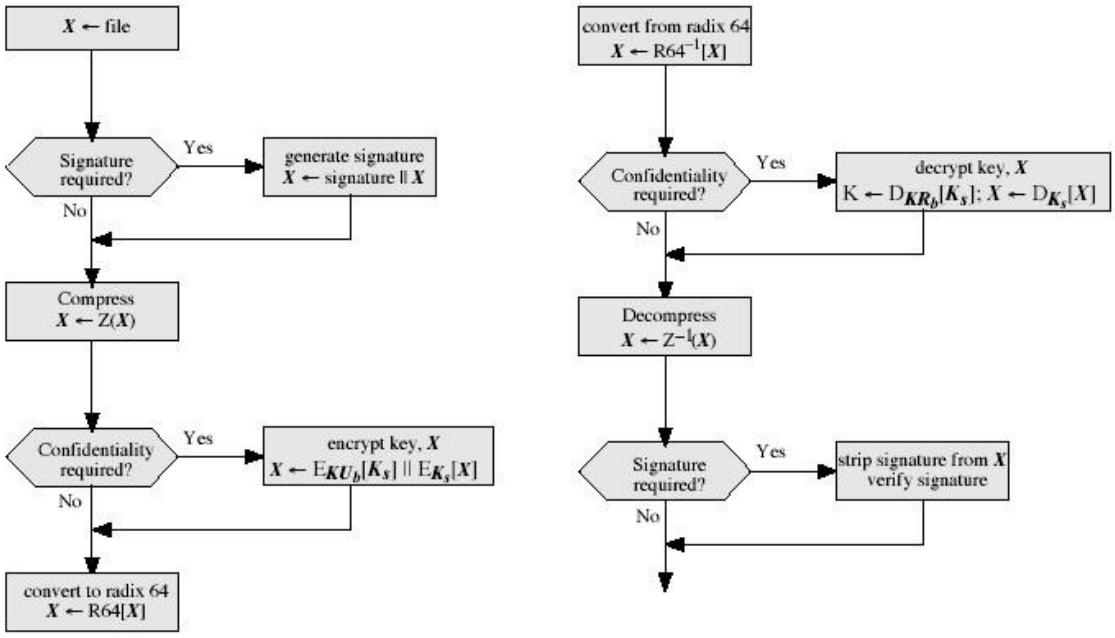
**5. Segmentation and reassembly**

E-mail facilities often are restricted to a maximum length. E.g., many of the facilities accessible through the internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.

To accommodate this restriction, PGP automatically subdivides a message that is too large into

segments that are small enough to send via e-mail. The segmentation is done after all the other processing, including the radix-64 conversion. At the receiving end, PGP must strip off all e-mail headers and reassemble the entire original block before performing the other steps.

PGP Operation Summary:




(a) Generic Transmission Diagram (from A)

(b) Generic Reception Diagram (to B)

## 4.5 Cryptographic keys and key rings

Three separate requirements can be identified with respect to these

keys: A means of generating unpredictable session keys is needed.

It must allow a user to have multiple public key/private key pairs.

Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

We now examine each of the requirements in turn.

**1. Session key generation**

Each session key is associated with a single message and is used only for the purpose of encryption and decryption of that message. Random 128-bit numbers are generated using CAST-128 itself. The input to the random number

generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted. Using cipher feedback mode, the CAST-128 produces two 64-bit cipher text blocks, which are concatenated to form the 128-bit session key. The plaintext input to CAST-128 is itself derived from a stream of 128-bit randomized numbers. These numbers are based on the keystroke input from the user.

**2. Key identifiers**

If multiple public/private key pair are used, then how does the recipient know which of the public keys was used to encrypt the session key? One simple solution would be to transmit the public key with the message but, it is unnecessary wasteful of space. Another solution would be to associate an identifier with each public key that is unique at least within each user.

The solution adopted by PGP is to assign a key ID to each public key that is, with very high probability, unique within a user ID. The key ID associated with each public key consists of its least significant 64 bits. i.e., the key ID of public key KUa is $(KUa \bmod 2^{64})$.

**A message consists of three components**.

**Message component** – includes actual data to be transmitted, as well as the filename and a timestamp that specifies the time of creation.

**Signature component** – includes the following

o Timestamp – time at which the signature was made.

o Message digest – hash code.

O Two octets of message digest – to enable the recipient to determine if the correct public key was used to decrypt the message.

Key ID of sender's public key – identifies the public key

**Session key component** – includes session key and the identifier of the recipient public key.

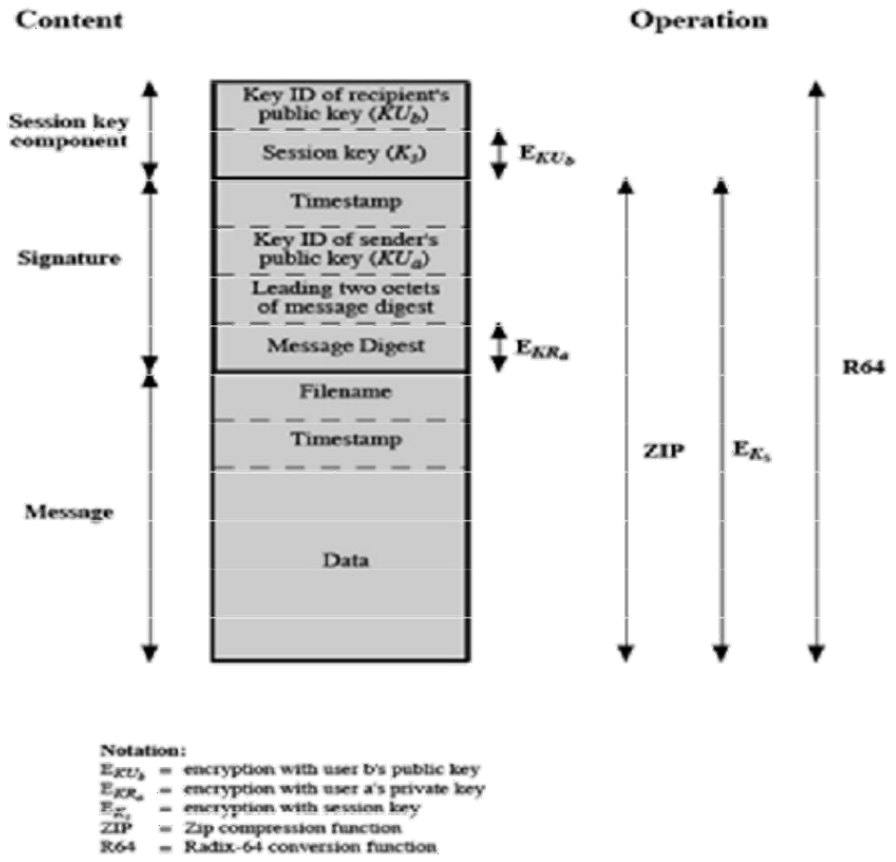



**Figure 15.3 General Format of PGP Message (from A to B)**

### 3. Key rings

PGP provides a pair of data structures at each node, one to store the public/private key pair owned by that node and one to store the public keys of the other users known at that node. These data structures are referred to as private key ring and public key ring.

**The general structures of the private and public key rings are shown below: Timestamp** – the date/time when this entry was made.

**Key ID** – the least significant bits of the public key.

**Public key –** public key portion of the pair. **Private key** – private key portion of the pair. **User ID** – the owner of the key.

**Key legitimacy field** – indicates the extent to which PGP will trust that this is a valid public key for this user.

**Private Key Ring**

| Timestamp | Key ID* | Public Key | Encrypted Private Key | User ID* |
|---|---|---|---|---|
| • • • | • • • | • • • | • • • | • • • |
| $T_i$ | $PU_i$ mod $2^{64}$ | $PU_i$ | E(H($P_i$), $PR_i$) | User $i$ |
| • • • | • • • | • • • | • • • | • • • |

**Public Key Ring**

| Timestamp | Key ID* | Public Key | Owner Trust | User ID* | Key Legitimacy | Signature(s) | Signature Trust(s) |
|---|---|---|---|---|---|---|---|
| • • • | • • • | • • • | • • • | • • • | • • • | • • • | • • • |
| $T_i$ | $PU_i$ mod $2^{64}$ | $PU_i$ | trust_flag$_i$ | User $i$ | trust_flag$_i$ | | |
| • • • | • • • | • • • | • • • | • • • | • • • | • • • | • • • |

* = field used to index table

**Figure 15.4 General Structure of Private and Public Key Rings**

**Signature trust field –** indicates the degree to which this PGP user trusts the signer to certify public key.

**Owner trust field** – indicates the degree to which this public key is trusted to sign other public key certificates.

**PGP message generation**

First consider message transmission and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps:

1. **signing the message**

PGP retrieves the sender's private key from the private key ring using user ID as an index. If user ID was not provided, the first private key from the ring is retrieved.

PGP prompts the user for the passpharse (password) to recover the unencrypted private key.

The signature component of the message is constructed.

2. **encrypting the message**

PGP generates a session key and encrypts the message.

PGP retrieves the recipient's public key from the public key ring using user

ID as index.

The session key component of the message is constructed. The receiving PGP entity performs the following steps:

**Decrypting the message**

PGP retrieves the receiver's private key from the private key ring, using the key ID field in the session key component of the message as an index.

PGP prompts the user for the passpharse (password) to recover the unencrypted private key.

PGP then recovers the session key and decrypts the message.

2. **Authenticating the message**

PGP retrieves the sender's public key from the public key ring, using the key ID field in the signature key component of the message as an index.

PGP recovers the transmitted message digest.

PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

**Public-Key Management**

This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. PGP provides a structure for solving this problem, with several suggested options that may be used.

***Approaches to Public-Key Management***

The essence of the problem is this: User A must build up a public-key ring containing the public keys of other users to interoperate with them using PGP. Suppose that A's key ring contains a public key attributed to B but that the key is, in fact, owned by C. This could happen if, for

example, A got the key from a bulletin board system (BBS) that was used by B to post the public key but that has been compromised by C. The result is that two threats now exist. First, C can send messages to A and forge B's signature, so that A will accept the message as coming from B. Second, any encrypted message from A to B can be read by C.

A number of approaches are possible for minimizing the risk that a user's public-key ring contains false public keys. Suppose that A wishes to obtain a reliable public key for B. The following are some approaches that could be used:

Physically get the key from B. B could store her public key (PUb) on a floppy disk and hand it to A.
Verify a key by telephone. If A can recognize B on the phone, A could call B and ask her to dictate the key, in radix-64 format, over the phone.
Obtain B's public key from a mutual trusted individual D. For this purpose, the introducer, D, creates a signed certificate. The certificate includes B's public key, the time of creation of the key, and a validity period for the key.
Obtain B's public key from a trusted certifying authority. Again, a public key certificate is created and signed by the authority. A could then access the authority, providing a user name and receiving a signed certificate.
For cases 3 and 4, A would already have to have a copy of the introducer's public key and trust that this key is valid. Ultimately, it is up to A to assign a level of trust to anyone who is to act as an introducer.

***The Use of Trust***

Although PGP does not include any specification for establishing certifying authorities or for establishing trust, it does provide a convenient means of using trust, associating trust with public keys, and exploiting trust information.

The basic structure is as follows. Each entry in the public-key ring is a public-key certificate.
Associated with each such entry is a key legitimacy field that indicates the extent to which PGP will trust that this is a valid public key for this user; the higher the level of trust, the stronger is the binding of this user ID to this key.
This field is computed by PGP.Also associated with the entry are zero or more signatures that the key ring owner has collected that sign this certificate. In turn, each signature has associated

with it a signature trust field that indicates the degree to which this PGP user trusts the signer to certify public keys.The key legitimacy field is derived from the collection of signature trust fields in the entry.Finally, each entry defines a public key associated with a particular owner, and an owner trust field is included that indicates the degree to which this public key is trusted to sign other public-key certificates; this level of trust is assigned by the user.

The three fields mentioned in the previous paragraph are each contained in a structure referred to as a trust flag byte.

Suppose that we are dealing with the public-key ring of user A. We can describe the operation of the trust processing as follows:

When A inserts a new public key on the public-key ring, PGP must assign a value to the trust flag that is associated with the owner of this public key. If the owner is A, and therefore this public key also appears in the private-key ring, then a value of ultimate trust is automatically assigned to the trust field. Otherwise, PGP asks A for his assessment of the trust to be assigned to the owner of this key, and A must enter the desired level. The user can specify that this owner is unknown, untrusted, marginally trusted, or completely trusted.

When the new public key is entered, one or more signatures may be attached to it.

More signatures may be added later. When a signature is inserted into the entry, PGP searches the public-key ring to see if the author of this signature is among the known public-key owners. If so, the OWNERTRUST value for this owner is assigned to the SIGTRUST field for this signature. If not, an unknown user value is assigned.

3. The value of the key legitimacy field is calculated on the basis of the signature trust fields present in this entry. If at least one signature has a signature trust value of ultimate, then the key legitimacy value is set to complete.

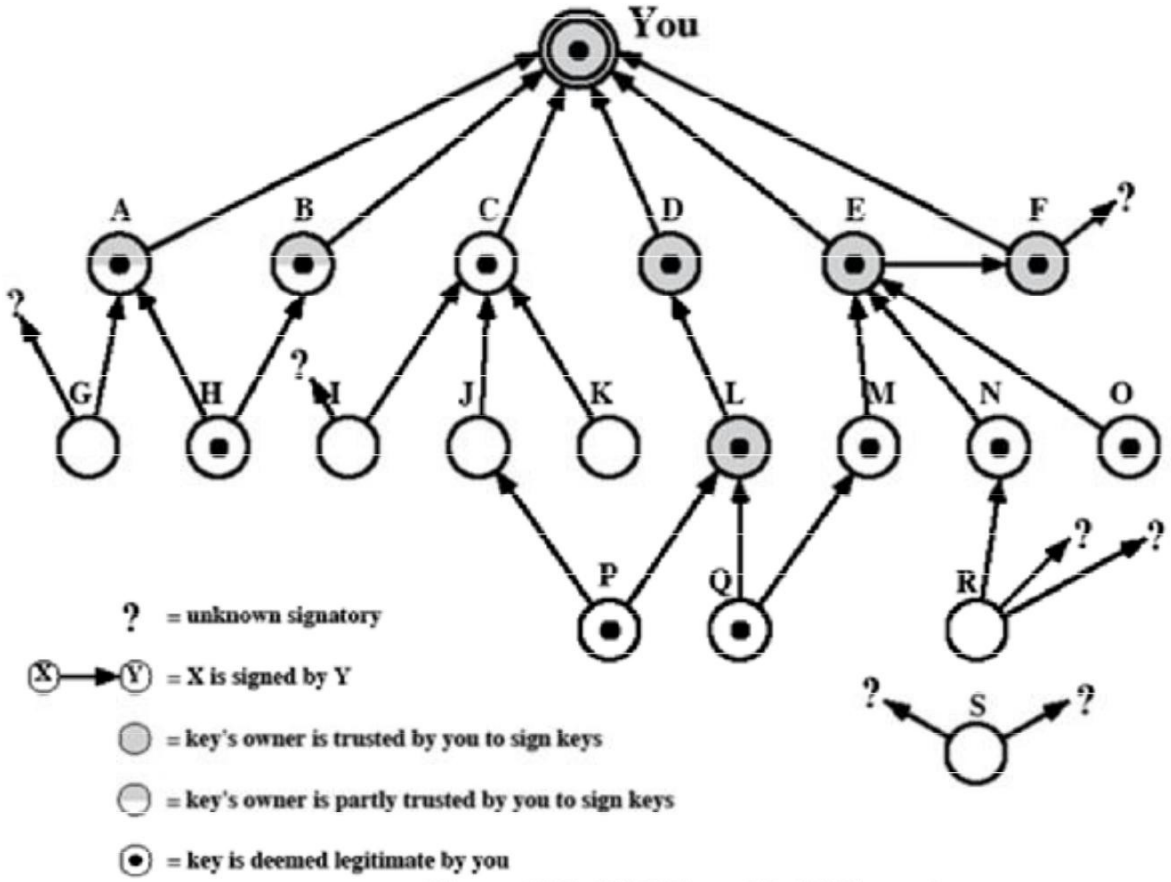



Figure 15.7 PGP Trust Model Example

The node labeled "You" refers to the entry in the public-key ring corresponding to this user. This key is legitimate and the OWNERTRUST value is ultimate trust. Each other node in the key ring has an OWNERTRUST value of undefined unless some other value is assigned by the user. In this example, this user has specified that it always trusts the following users to sign other keys: D, E, F, L. This user partially trusts users A and B to sign other keys.

So the shading, or lack thereof, of the nodes in Figure 15.7 indicates the level of trust assigned by this user. The tree structure indicates which keys have been signed by which

other users. If a key is signed by a user whose key is also in this key ring, the arrow joins the signed key to the signatory. If the key is signed by a user whose key is not present in this key ring, the arrow joins the signed key to a question mark, indicating that the signatory is unknown to this user.

Note that all keys whose owners are fully or partially trusted by this user have been signed by this user, with the exception of node L.

1. We assume that two partially trusted signatures are sufficient to certify a key. Hence, the key for user H is deemed legitimate by PGP because it is signed by A and B, both of whom are partially trusted.
2. A key may be determined to be legitimate because it is signed by one fully trusted or two partially trusted signatories, but its user may not be trusted to sign other keys. For example, N's key is legitimate because it is signed by E, whom this user trusts, but N is not trusted to sign other keys because this user has not assigned N that trust value. Therefore, although R's key is signed by N, PGP does not consider R's key legitimate. This situation makes perfect sense. If you wish to send a private message to some individual, it is not necessary that you trust that individual in any respect. It is only necessary that you are sure that you have the correct public key for that individual.
3. Figure 15.7 also shows an example of a detached "orphan" node S, with two unknown signatures. Such a key may have been acquired from a key server. PGP cannot assume that this key is legitimate simply because it came from a reputable server. The user must declare the key legitimate by signing it or by telling PGP that it is willing to trust fully one of the key's signatories.

## 4.6 S/MIME

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, based on technology from RSA Data Security. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851.

**Multipurpose Internet Mail Extensions**

MIME is an extension to the RFC 822 framework that is intended to address some of the problems and limitations of the use of SMTP (Simple Mail Transfer Protocol) or some other mail transfer protocol and RFC 822 for electronic mail. Following are the limitations of SMTP/822 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 with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.

SMTP servers may reject mail message over a certain size.

SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.

SMTP gateways to X.400 electronic mail networks cannot handle nontextual data included in X.400 messages.

Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC

Common problems include:

o Deletion, addition, or reordering of carriage return and linefeed

o Truncating or wrapping lines longer than 76 characters

o Removal of trailing white space (tab and space characters)

o Padding of lines in a message to the same length

oConversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations. The specification is provided in RFCs 2045 through 2049.

***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.

**A number of content formats** are defined, thus standardizing representations that support multimedia electronic mail.

**Transfer encodings** are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

In this subsection, we introduce the five message header fields. The next two subsections deal with content formats and transfer encodings.

**The five header fields defined in MIME are as follows:**

**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

**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 bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

Table 15.3 lists the content types specified in RFC 2046. There are seven different major types of content and a total of 15 subtypes

| **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 |
| | 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. |
| | octet- stream | General binary data consisting of 8-bit bytes. |

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 boundary, that defines the delimiter between body parts. 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. For example, the moderator of a group might collect e- mail messages from participants, bundle these messages, and send them out in one encapsulating MIME message.

The message type provides a number of important capabilities in MIME. The message/rfc822 subtype indicates that the body is an entire message, including header and body. Despite the name of this subtype, the encapsulated message may be not only a simple RFC 822 message, but also any MIME message.

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.

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. As with the other message types, the message/external-body subtype has an outer header and an encapsulated message with its own header. The only necessary field in the outer header is the Content-Type field, which identifies this as a message/external-body subtype. The inner header is the message header for

the encapsulated message. The Content-Type field in the outer header must include an access-type parameter, which indicates the method of access, such as FTP (file transfer protocol).

The application type refers to other kinds of data, typically either uninterpreted binary data or information to be processed by a mail-based application.

***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 objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content-Transfer- Encoding field can actually take on six values, as listed in Table 15.4. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used, for which a name is to be supplied. The two actual encoding schemes defined are quoted-printable and base64.

***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 |
| 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. |

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.

***Canonical Form***

An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system.

**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. We then look in more detail at this capability by examining message formats and message preparation.

***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.

## 4.7 Cryptographic Algorithms

hash functions: SHA-1 & MD5

digital signatures: DSS & RSA

session key encryption: ElGamal & RSA

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

have a procedure to decide which algorithms to use.

Table 15.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology, taken from RFC 2119 to specify the 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.

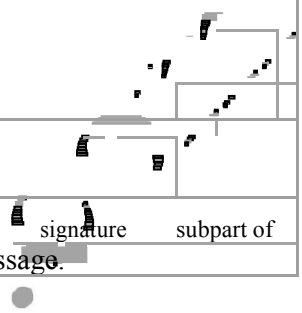
**S/MIME Messages**

S/MIME makes use of a number of new MIME content types, which are shown in Table 15.7. All of the new application types use the designation PKCS. This refers to a set of public-key cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

| Function | Requirement |
|---|---|
| Create a message digest to be used in forming a digital signature.<br><br>Encrypt message digest to form digital signature. | MUST support SHA-1.<br><br>Receiver SHOULD support MD5 for backward compatibility.<br><br>Sending and receiving agents MUST support DSS.<br>Sending agents SHOULD support RSA encryption.<br>Receiving agents SHOULD support verification of |
| Encrypt session key for transmission with message. | Sending and receiving agents SHOULD support Diffie-Hellman.<br><br>Sending and receiving agents MUST support RSA |
| Encrypt message for transmission with one-time session key. | Sending and receiving agents MUST support encryption with triple DES<br><br>Sending agents SHOULD support encryption with AES. |

| | |
|---|---|
| Create a message authentication code | Receiving agents MUST support HMAC with SHA-1.<br><br>Receiving agents SHOULD support HMAC with |

| 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 | pkcs 7- mime | signedData | A signed S/MIME entity. |
| | pkcs 7- mime | envelopedData | An encrypted S/MIME entity. |
| | pkcs 7- mime | degenerate | |
| | | | |
| | pkcs 7- signature | signedData | The content type of the signature subpart of a multipart/signed message. |



We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

### 4.7.1 SECURING A MIME ENTITY

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message (except for the RFC 822 headers), or if the MIME content type is multipart, then a MIME entity is one or more of the subparts of the message. In all cases, the message to be sent is converted to canonical form. In particular, for a given type and subtype, the appropriate canonical form is used for the message content. For a multipart message, the appropriate canonical form is used

for each subpart.

The use of transfer encoding requires special attention.

***i)EnvelopedData***

An application/pkcs7-mime subtype is used for one of four categories of S/MIME processing, each with a unique smime-type parameter. In all cases, the resulting entity, referred to as an object, is represented in a form known as Basic Encoding Rules (BER), which is defined in ITU-T Recommendation X.209. The steps for preparing an envelopedData MIME entity are as follows:

Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or triple DES).

For each recipient, encrypt the session key with the recipient's public RSA key.

For each recipient, prepare a block known as Recipient Info that contains an identifier ofthe recipient'spublic-keycertificate,[3] an identifierofthealgorithm usedto encrypt the session key, and the encrypted session key.

This is an X.509 certificate, discussed later in this section.

**4.**Encrypt the message content with the session key.

TheRecipientInfoblocksfollowed bytheencryptedcontent constitute the envelopedData.

This information is then encoded into base64. To recover the encrypted message, therecipientfirst strips off the base64 encoding. Then the recipient's private key is used to recover the session key. Finally, the message content is decrypted with the session key.

***ii)SignedData***

The signedData smime-type can actually 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 as follows:

Select a message digest algorithm (SHA or MD5).

Compute the message digest, or hash function, of the content to be signed.

Encrypt the message digest with the signer's private key.

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. The signedData entity may also include a set of public-key certificates sufficient to constitute a chain from a recognized root or top-level certification authority to the signer. This information is then encoded into base64.

To recover the signed message and verify the signature, the recipient first strips off the base64 encoding. Then the signer's public key is used to decrypt the message digest. The recipient independently computes the message digest and compares it to the decrypted message digest to verify the signature.

***iii)Clear Signing***

C l e a r signing is achieved using the multipart content type with a signed subtype.

As was mentioned, this signing process does not involve transforming the message to be signed, so that the message is sent "in the clear."

T h u s , recipients with MIME capability but not S/MIME capability are able to read the incoming message.

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 means that if the first part is not 7bit, then it needs to be encoded using base64 or quoted- printable. Then this part is processed in the same manner as signedData, but in this case an object with signedData format is created that has an empty message content field. This object is a detached signature. It is then transfer encoded using base64 to become the second part of the multipart/signed message. This second part has a MIME content type of application and a subtype of pkcs7-signature

The protocol parameter indicates that this is a two-part clear-signed entity. The receiver can verify the signature by taking the message digest of the first part and comparing this to the message digest recovered from the signature in the second part.

***Registration Request***

Typically, 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.

The certification request includes certificationRequestInfo block, followed by an identifier of the public-key encryption algorithm, followed by the signature of the certificationRequestInfo block, made using the sender's private key.

The certificationRequestInfo block includes a name of the certificate subject (the entity whose public key is to be certified) and a bit-string representation of the user's public key.

***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 smime-type parameter of degenerate. The steps involved are the same as those for creating a signedData message, except that there is no message content and the signerInfo field is empty.

### 4.7.2 S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509 The key- management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists.

****User Agent Role***

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

**Keygeneration:**Theuserofsomerelatedadministrativeutility (e.g.,one associated withLANmanagement)MUSTbecapableofgeneratingakey pairfrom agoodsourceof nondeterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of 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. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

****VeriSign Certificates***

There are several companies that provide certification authority (CA) services. For example, Nortel has designed an enterprise CA solution and can provide S/MIME support within an organization.

There are a number of Internet-based CAs, including VeriSign, GTE, and the U.S. Postal Service. Of these, the most widely used is the VeriSign CA service, a brief description of which we now provide.

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. As of early 1998, over 35,000 commercial Web sites were using VeriSign Server Digital IDs, and over a million consumer Digital IDs had been issued to users of Netscape and Microsoft browsers.

**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 other user-supplied information, including**

Address
E-mail address
Basic registration information (country, zip code, age, and gender)

**VeriSign** provides three levels, or classes, of security for public-key certificates. A user requests a certificate online at VeriSign's Web site or other participating Web sites. Class
1 and Class 2 requests are processed on line, and in most cases take only a few seconds to approve. Briefly, the following procedures are used:

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. Finally, confirmation is sent to the specified postal address alerting the user that a Digital ID has been issued in his or her name.

For Class 3 Digital IDs, VeriSign requires a higher level of identity assurance. An individual must prove his or her identity by providing notarized credentials or applying in person.

## 4.8 Enhanced Security Services

As of this writing, three enhanced security services have been proposed in an Internet

draft.: **Signed receipts:** A signed receipt may be requested in a SignedData object.

Returning a signed receipt provides proof of delivery to the originator of a message and allows the originator to demonstrate to a third party that the recipient received the message.

**Security labels**: A security label may be included in the authenticated attributes of a SignedData object. A security label is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. The labels may be used for access control, by indicating which users are permitte 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.

## 4.9 Key Management

all cryptographic systems have the problem of how to securely and reliably distribute the keys used

in many cases, failures in a secure system are due not to breaking the algorithm, but to breaking the key distribution scheme

ideally the distribution protocol should be formally verified, recent advances make this more achievable

possible key distribution techniques include:

physical delivery by secure courier eg code-books used submarines

one-time pads used by diplomatic missions

registration name and password for computers

authentication key server (private key, eg Kerberos) have an on-line server trusted by all clients server has a unique secret key shared with each client server negotiates keys on behalf of clients

public notary (public key, eg SPX) have an off-line server trusted by all clients

server has a well known public key

server signs public key certificates for each client

***Authentication Protocols***

if using a key server, must use some protocol between user and server

this protocol should be validated, formal techniques exist to acheive this (Ban logic provers

### 4.9.1 Challenge-Response

- basic technique used to ensure a password is never sent in the clear
- given a client and a server share a key
  - server sends a random challenge vector
  - client encrypts it with private key and returns this
  - server verifies response with copy of private key

can repeat protocol in other direction to authenticate server to client (2-way authentication)

in simplest form, keys are physically distributed before secure comminications is required

in more complex forms, keys are stored in a central trusted key server

### 4.9.2 Needham-Schroeder

original third-party key distribution protocol

R M Needham, M D Schroeder, "Using Encryption for Authentication in Large Networks of Computers", CACM, 21(12), Dec 1978, pp993-998

- given Alice want to communicate with Bob, and have a Key Server S, protocol is:

Message 1 A -> S A, B, $N_a$

Message 2 S -> A $E_{Kas}\{N_a , B, K_{ab}, E_{Kbs}\{K_{ab}, A\} \}$

Message 3 A -> B $E_{Kbs}\{K_{ab}, A\}$

Message 4 B -> A $E_{Kab}\{N_b\}$

Message 5 A-> B $E_{Kab}\{N_b\text{-}1\}$

nb: $N_a$ is a random value chosen by Alice, $N_b$ random chosen by Bob

after this protocol runs, Alice and Bob share a secret session key $K_{ab}$ for secure communication

- including a timestamp in messages 1 to 3, which requires synchronized clocks (by Denning & Sacco 81)
- having A ask B for a random value Jb to be sent to S for return in $E_{Kbs}\{Kab, A, Jb\}$ (by Needham & Schroeder 87)

many other protocols exist but care is needed

### 4.9.3 KEY MANAGEMENT

Public-key encryption helps address key distribution problems

Have two aspects:

- Distribution of public keys
- Use of public-key encryption to distribute secret keys

**Distribution of Public Keys**

Distribution of Public Keys can be done in one of the four ways: Public announcement

Publicly available directory

Public-key authority

Public-key certificates

### 4.9.4 Public Announcement

Users distribute public keys to recipients or broadcast to community at large o

eg. Append PGP keys to email messages or post to news groups or email list

Major weakness is forgery

- Anyone can create a key claiming to be someone else and broadcast it
- Until forgery is discovered can masquerade as claimed user

### 4.9.5 Public-Key Certificates

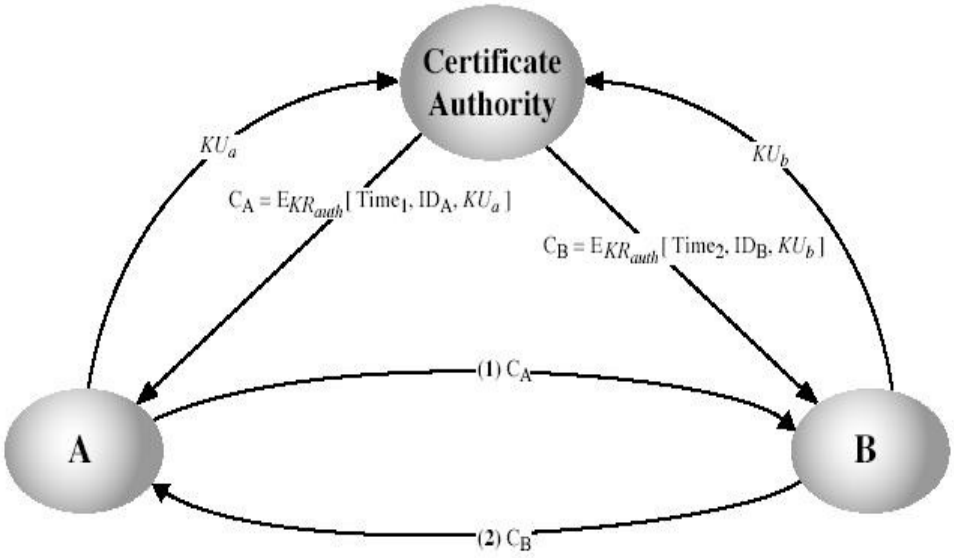
Certificates allow key exchange without real-time access to public-key authority

A certificate binds **identity** to **public key**

o Usually with other info such as period of validity, rights of use etc

With all contents **signed** by a trusted Public-Key or Certificate Authority (CA)

Can be verified by anyone who knows the public-key authorities public-key




**Publicly Available Directory**

Can obtain greater security by registering keys with a public directory

Directory must be trusted with properties:

o Contains {name, public-key} entries

o Participants register securely with directory

o Participants can replace key at any time

o Directory is periodically published

o Directory can be accessed electronically

Still vulnerable to tampering or forgery

**Public-Key Authority**

Improve security by tightening control over distribution of keys from directory

Has properties of directory

Requires users to know public key for the directory

• Users interact with directory to obtain any desired public key securely

o Does require real-time access to directory when keys are needed

#### 4.9.5.1 Kerberos - An Example of a Key Server

trusted key server system developed by MIT

provides centralised third-party authentication in a distributed network

access control may be provided for each computing resource in either a local or remote network (realm) has a Key Distribution Centre (KDC), containing a database of:

- principles (customers and services)
- encryption keys

basic third-party authentication scheme

KDC provides non-corruptible authentication credentials (tickets or tokens)


### 4.9.6 Kerberos - Initial User Authentication

user requests an initial ticket from KDC

used as basis for all remote access requests


**Kerberos - Request for a Remote Service**

user requests access to a remote service

- obtains a ticket from KDC protected with remote key
- sends ticket with request to remote server


**Kerberos - in practise**

currently have two Kerberos versions

- 4 : restricted to a single realm
- 5 : allows inter-realm authentication, in beta test

Kerberos v5 is an Internet standard specified in RFC1510, and used by many utilities to use Kerberos need to have a KDC on your network need to have applications running on all participating systems major problem - US export restrictions

Kerberos cannot be directly distributed outside the US in source format (& binary versions must obscure crypto routine entry points and have no encryption) else crypto libraries must be reimplemented locally

**X.509 - Directory Authentication Service**

part of CCITT X.500 directory services defines framework for authentication services

directory may store public-key certificates uses public-key cryptography and digital signatures algorithms not standardized but RSA is recommended

**X.509 Certificate**

issued by a Certification Authority (CA) each certificate contains:

- version
- serial number (unique within CA)

- algorithm identifier (used to sign certificate)
- issuer (CA)
- period of validity (from - to dates)
- subject (name of owner)
- public-key (algorithm, parameters, key)
- signature (of hash of all fields in certificate)

any user with access to CA can get any certificate from it

only the CA can modify a certificate

**CA Hierarchy**

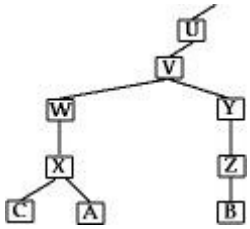
CA form a hierarchy

each CA has certificates for clients and parent

each client trusts parents certificates

enable verification of any certificate from one CA by users of all other CAs in hierarchy

X<<A>> means certificate for A signed by authority X




A acquires B certificate following chain:

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

B acquires A certificate following chain:

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

**Authentication Procedures**

X.509 includes three alternative authentication procedures

**One-Way Authentication**

1 message ( A->B) to establish

- identity of A and that messages is from A
- message intended for B
- integrity & originality of message

**Two-Way Authentication**

2 messages (A->B, B->A) which also establishes

- identity of B and that replay is from B
- reply intended for A
- integrity & originality of reply

**Three-Way Authentication**

3 messages (A->B, B->A, A->B) which enables

- above authentication without syncronised clocks

## 4.9.7 DIFFIE-HELLMAN KEY EXCHANGE

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.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. First, we define a primitive root of a prime number p as one whose power generate all the integers from 1 to (p-1) i.e., if „a‟ is a primitive root of a prime number p, then the numbers

$a \bmod p, a^2 \bmod p, \ldots a^{p-1} \bmod p$ are distinct and consists of integers from 1 to (p-1) in some permutation. For any integer „b‟ and a primitive root „a‟ of a prime number „p‟, we can find a unique exponent „i‟ such that

$b \equiv a^i \bmod p$ where $0 \leq i \leq (p-1)$

The exponent „i" is referred to as discrete logarithm. With this background, we can define

Diffie Hellman key exchange as follows:

There are publicly known numbers: a prime number „q" and an integer **α** that is primitive root of q. suppose users A and B wish to exchange a key. User A selects a random integer XA < q and computes YA = $\alpha^{XA}$ mod q. Similarly, user B independently selects a random integer XB < q and computes YB = $\alpha^{XB}$ mod q. Each side keeps the X value private and makes the Y value available publicly to the other side. User A computes the key as $K = (YB)^{XA}$ mod q and

User B computes the key as $K = (YA)^{XB}$ mod q

These two calculations produce identical results. $K = (YB)^{XA}$ mod q

$$(\alpha^{XB} \text{ mod } q)^{XA} \text{ mod } q$$

$$(\alpha^{XB})^{XA} \text{ mod } q$$

$$(\alpha^{XA})^{XB} \text{ mod } q$$

$$(\alpha^{XA} \text{ mod } q)^{XB} \text{ mod } q$$

$$(YA)^{XB} \text{ mod } q$$

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

The security of the algorithm lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

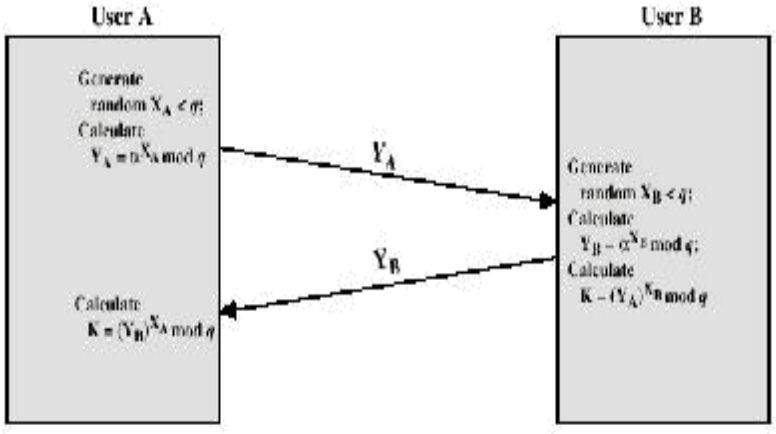



**Fig: Diffie Hellman Key exchange**

## 4 Security in Practise - Secure Email

email is one of the most widely used and regarded network services

currently message contents are not secure

- may be inspected either in transit
- or by suitably priviledged users on destination system

Email Privacy Enhancement Services

- confidentiality (protection from disclosure)
- authentication (of originator)
- message integrity (protection from modification)
- non-repudiation of origin
- (protection from denial by sender)

can't assume real-time access to a trusted key server

often implement using Email Encapsulation

## 4.10 PEM

Privacy Enhanced Mail

Internet standard for security enhancements to Internet (RFC822) email

- developed by a Working group of the IETF
- specified in RFC1421, RFC1422, RFC1423, RFC1424

uses message encapsulation to add features

confidentiality - DES encryption in CBC mode

integrity - DES encrypted MIC (MD2/MD5)

authentication - DES/RSA encrypted MIC

non-repudiation - RSA encrypted MIC

### 4.10.1 PEM - Key Management

central key server (private-key)

- requires access to on-line server

public-key certificates

- uses X.509 Directory Service Strong Authentication to protect key certificates
- signed by a Certification Authority (CA)
- CAs form a hierarchy to permit cross-validation of certificates
- CAs must be licenced by RSA Data Inc.
- currently only licensed in US/Canada

### 4.10.2 PGP

Pretty Good Privacy

widely used de facto secure email standard

- developed by Phil Zimmermann

- available on Unix, PC, Macintosh and Amiga systems
- free!!!!

confidentiality - IDEA encryption

integrity - RSA encrypted MIC (MD5)

authentication & non-repudiation - RSA encrypted MIC

uses grass-roots key distribution

- trusted introducers used to validate keys
- no certification authority hierarchy needed

### 4.10.3 PGP - In Use

all PGP functions are performed by a single program

must be integrated into existing email/news

each user has a keyring of known keys

- containing their own public and private keys (protected by a password)
- public keys given to you directly by a person
- public keys signed by trusted introducers

used to sign/encrypt your messages

used to validate messages received

### 4.10.4 Sample PGP Message

-----BEGIN PGP SIGNED MESSAGE-----

May all your signals trap
May your references be bounded
All memory aligned
Floats to ints be rounded

Lawrie

-----BEGIN PGP SIGNATURE-----

Version: 2.3

iQBzAgUBLdl1RILpoub8ek7fAQF2nwLuJwVPh8iiFrksXSCe6z37ZdV37pXvsYyz0WAnCBCdpu55
yId5/kVhmvusTo10zUHPssPwB99TQq9YsduSfkVeILjfJNJEuUWQkJl8dWvaB+IIEEodF0Xpbc23krn
uOA==

=hn90

-----END PGP SIGNATURE-----

**PGP - Issues**

were questions of legality, but PGP may now be legally used by anyone in the world:

- noncommercial use in US/Canada with licenced MIT version
- commercial use in US/Canada with Viacrypt version
- noncommercial use outside the US is probably legal with (non US sourced) international version
- commercial use outside the US requires an IDEA licence for the international version

is on-going legal battle in US over its original export between US govt and Phil Zimmermann

### 4.10.4.1 Security in Practice - SNMP

SNMP is a widely used network management protocol

comprises

- management station
- management agent with
- its management information base (MIB)
- linked by network management protocol (GET,SET)

SNMP v1 lacks any security (GET and SET open if there)

SNMP v2 includes security extensions for

- message authentication (keyed MD5)

- message secrecy (DES)

based on the SNMPv2 **party** (sender & receiver roles)

- used for access control & key management
- all associated information stored in a party MIB

assumes syncronised clocks (within a set interval)

### 4.10.5 User Authentication

user authentication (identity verification)

- convince system of your identity -

before it can act on your behalf

sometimes also require that the computer verify its identity with the user

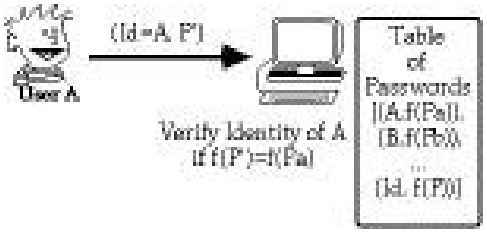
user authentication is based on three methods

- what you know -

what you have -

what you are

· all then involve some validation of information supplied against a table of possible values based on users claimed identity




### 4.10.6 What you Know

#### 4.10.6.1 Passwords or Pass-phrases

prompt user for a login name and password

verify identity by checking that password is correct

· on some (older) systems, password was stored in the clear (this is now regarded as insecure, since breakin compromises all users of the system)

- more often use a one-way function, whose output cannot easily be used to find the input

value either takes a fixed sized input (eg 8 chars)

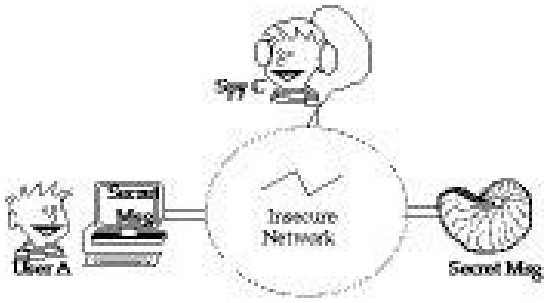
- or based on a hash function to accept a variable sized input to create the value

· important that passwords are selected with care to reduce risk of exhaustive search

Denning Computer (In) security Fig 2 & 3, pp111-12

## 4.10.7 One-shot Passwords

· one problem with traditional passwords is caused by eavesdropping theit transfer over an insecure network




one possible solution is to use one-shot (one-time) passwords

these are passwords used once only

future values cannot be predicted from older values

either generate a printed list, and keep matching list on system to be accessed (cf home banking)

or use an algorithm based on a one-way function $f$ (eg MD5) to generate previous values in series (eg SKey)

start with a secret password s, and number N

- $p_{(0)} = f^{N}(s)$

next password in series is

- $p_{(1)} = f^{N-1}(s)$

- must reset password after N uses

generally good only for infrequent access