UNIT- IV

User Authentication: Remote user Authentication Principles, Kerberos, Electronic mail security: Pretty Good Privacy (PGP), S/MIME.

IP Security: IP Security Overview,IP Security Policy, Encapsulating Security Payload, Combining Security Associations, Internet Key Exchange.

Remote user Authentication Principles : In most computer security contexts, user authentication is the fundamental building block and the primary line of defense. User authentication is the basis for most types of access control and for user accountability. A typical item of authentication information associated with this user ID is a password, which is kept secret (known only to Alice and to the system).

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

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

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

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

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

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

Note: For network-based user authentication, the most important methods involve cryptographic keys and something the individual knows, such as a password.

Mutual Authentication : An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness.

To prevent masquerade and to prevent compromise of session keys, essential identification and session-key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose.

The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. The following examples of **replay attacks**:

- **Simple replay:** The opponent simply copies a message and replays it later.
- **Repetition that can be logged:** An opponent can replay a timestamped message within the valid time window.
- **Repetition that cannot be detected:** This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- **Backward replay without modification:** This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order.
one of the following two general approaches is used:

• **Timestamps:** Party A accepts a message as fresh only if the message contains a **timestamp** that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.

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

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400.

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

Kerberos

Kerberos is an authentication service developed as part of Project Athena at MIT. It addresses the threats posed in an open distributed environment in which users at workstations wish to access services on servers distributed throughout the network. Some of these threats are:

A user may gain access to a particular workstation and pretend to be another user operating from that workstation.

A user may alter the network address of a workstation so that the requests sent from the altered workstation appear to come from the impersonated workstation.

A user may eavesdrop on exchanges and use a replay attack to gain entrance to a server or to disrupt operations.

Two versions of Kerberos are in current use: Version-4 and Version-5. The first published report on Kerberos listed the following requirements:

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 Two versions of Kerberos are in common use: Version 4 is most widely used version. Version 5 corrects some of the security deficiencies of Version 4.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(c) Client/Server Authentication Exchange to obtain service

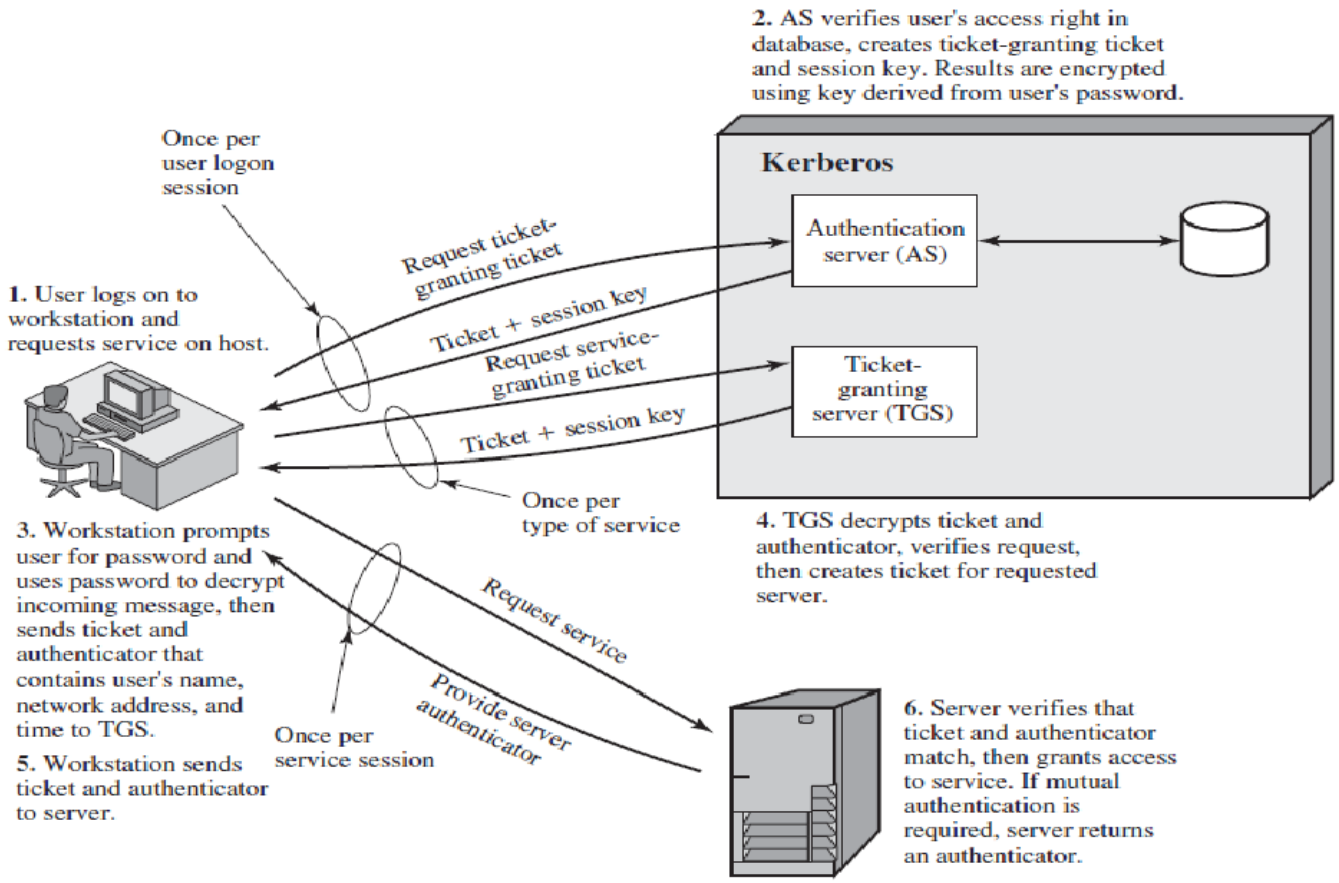


Figure 15.1 Overview of Kerberos

There is a problem of captured ticket-granting tickets and the need to determine that the ticket presenter is the same as the client for whom the ticket was issued. An efficient way of doing this is to use a session encryption key to secure information.

Message (1) includes a timestamp, so that the AS knows that the message is timely. Message (2) includes several elements of the ticket in a form accessible to C. This enables C to confirm that this ticket is for the TGS and to learn its expiration time. Note that the ticket does not prove anyone's identity but is a way to distribute keys securely. It is the authenticator that proves the client's identity. Because the authenticator can be used only once and has a short lifetime, the threat of an opponent stealing both the ticket and the authenticator for presentation later is countered. C then sends the TGS a message that includes the ticket plus the ID of the requested service (message 3). The reply from the TGS, in message (4), follows the form of message (2). C now has a reusable service-granting ticket for V. When C presents this ticket, as shown in message (5), it also sends an authenticator.

Kerberos version 5

Kerberos Version 5 is specified in RFC 1510 and provides a number of improvements over version 4 in the areas of environmental shortcomings and technical deficiencies. It includes some new elements such as:

Realm: Indicates realm of the user

Options Times

– From: the desired start time for the ticket
– Till: the requested expiration time
– Rtime: requested renew-till time
Nonce: A random value to assure the response is fresh

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

The following figure shows the authentication messages where service is being requested from another domain. The ticket presented to the remote server indicates the realm in which the user was originally authenticated. The server chooses whether to honor the remote request. One problem presented by the foregoing approach is that it does not scale well to many realms, as each pair of realms need to share a key.

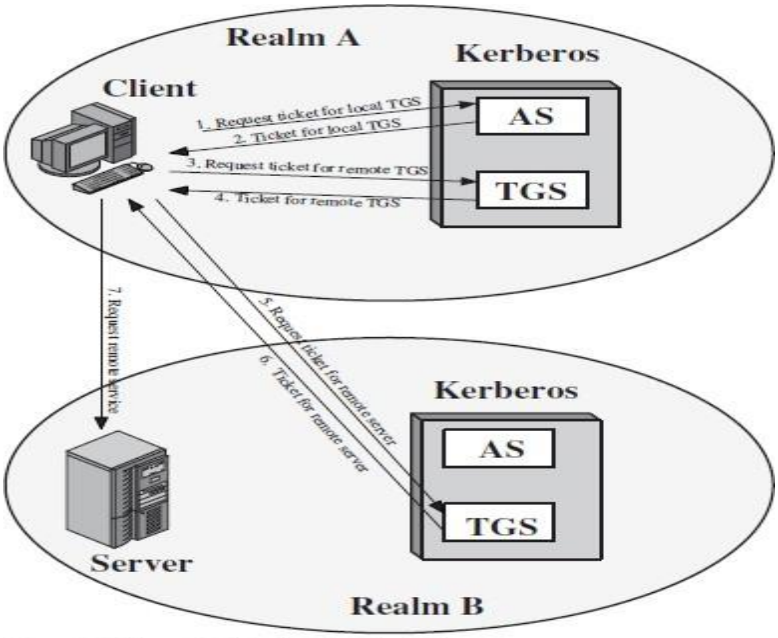


Figure 15.2 Request for Service in Another Realm

(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, [$K_{c,tgs}$ || $Times$ || $Nonce_1$ || $Realm_{tgs}$ || ID_{tgs}])

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

(a) Authentication Service Exchange to obtain ticket-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($K_{c,tgs}$, [$K_{c,v}$ || $Times$ || $Nonce_2$ || $Realm_v$ || ID_v])

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

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

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

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

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

(6) V → C $E_{K_{c,v}}$ [TS_2 || Subkey || Seq#]

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

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

(c) Client/Server Authentication Exchange to obtain service

Remote user Authentication using Asymmetric Encryption

This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

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

This protocol is compact but, as before, requires the synchronization of clocks. Another approach, proposed by Woo and Lam [WOO92a], makes use of nonces.

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

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

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

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_A accounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

Public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality ,authentication, or both. These approaches require that either the sender know the recipient's public key (confidentiality), the recipient know the sender's public key (authentication), or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

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

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

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 13.2:

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

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

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

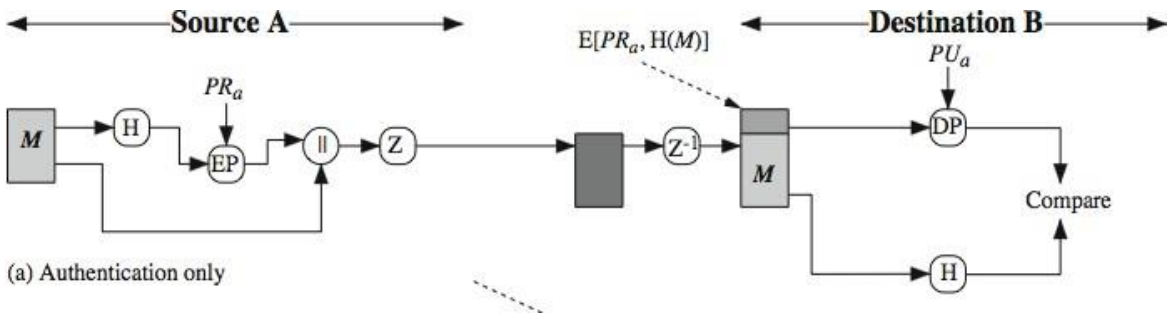
Electronic Mail Security: Pretty Good Privacy (PGP)

The Pretty Good Privacy (PGP) secure email program, is a remarkable phenomenon, has grown explosively and is now widely used. Largely the effort of a single person, Phil Zimmermann, who selected the best available crypto algorithms to use & integrated them into a single program, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. It runs on a wide range of systems, in both free & commercial versions.

- widely used de facto secure email program
- developed by Phil Zimmermann
- selected best available crypto algs to use
- integrated into a single program
- on Unix, PC, Macintosh and other systems
- originally free, now also have commercial versions available

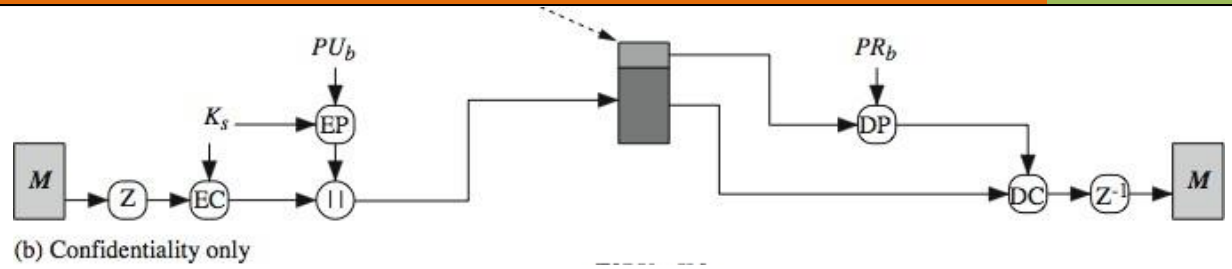
PGP Operation – Authentication

1. sender creates message
2. make SHA-1160-bit hash of message
3. attached RSA signed hash to message
4. receiver decrypts & recovers hash code
5. receiver verifies received message hash



PGP Operation – Confidentiality

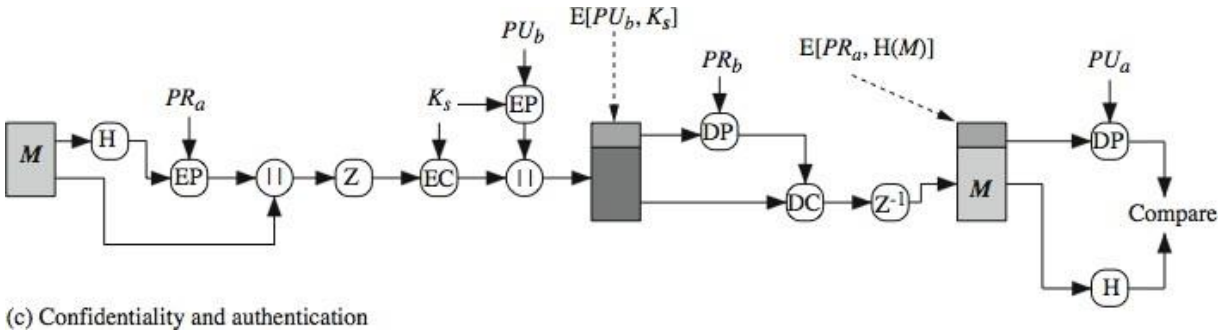
1. sender forms 128-bit random session key
2. encrypts message with session key
3. attaches session key encrypted with RSA
4. receiver decrypts & recovers session key
5. session key is used to decrypt message



(b) Confidentiality only

PGP Operation – Confidentiality & Authentication

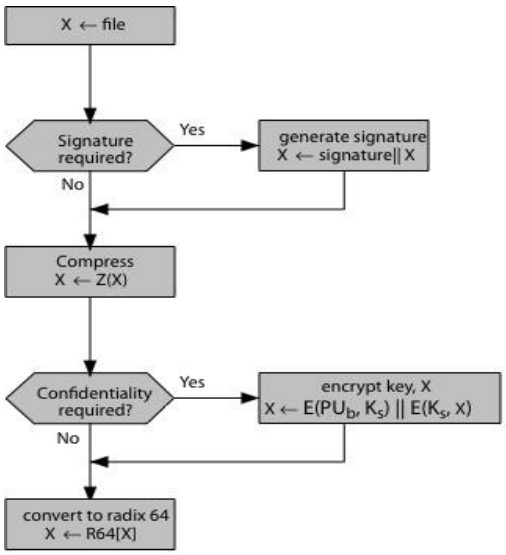
- can use both services on same message
  - create signature & attach to message
  - encrypt both message & signature
  - attach RSA/ElGamal encrypted session key



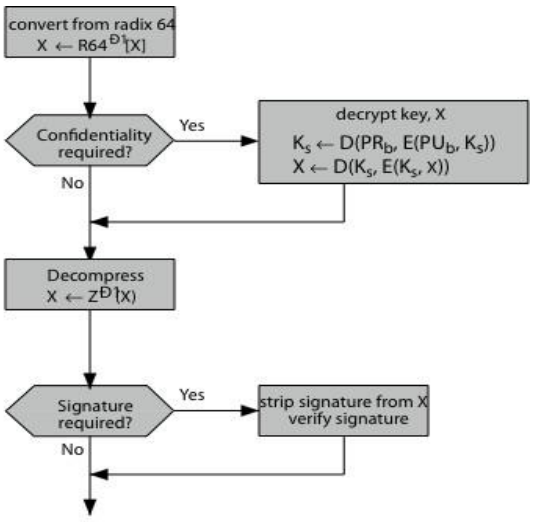
(c) Confidentiality and authentication

PGP Operation – Compression

By default PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage. The signature is generated before compression for the reasons shown. The compression algorithm used is ZIP, and is indicated by Z for compression and Z–1 for decompression in Figure 18.1. It is described in Online Appendix O.



(a) Generic Transmission Diagram (from A)



(b) Generic Reception Diagram (to B)

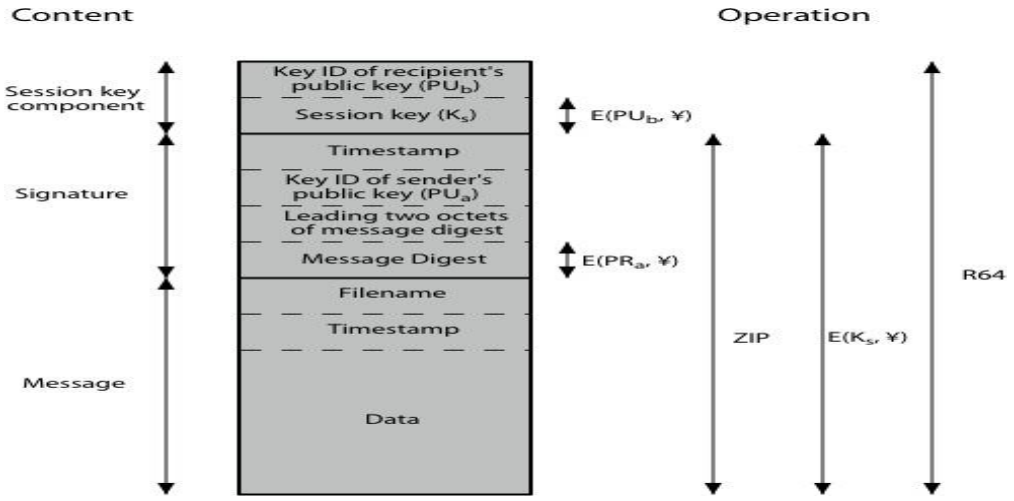
PGP Session Keys

PGP makes use of four types of keys: one-time session symmetric keys, public keys, private keys, and passphrase-based symmetric keys.

PGP Public & Private Keys

Since many public/private keys may be in use with PGP, there is a need to identify which key is actually used to encrypt the session key for any specific message. You could just send the full public-key with every message, but this is inefficient. Rather PGP use a key identifier based on the least significant 64-bits of the key, which will very likely be unique. Then only the much shorter key ID would need to be transmitted with any message. A key ID is also required for the PGP digital signature.

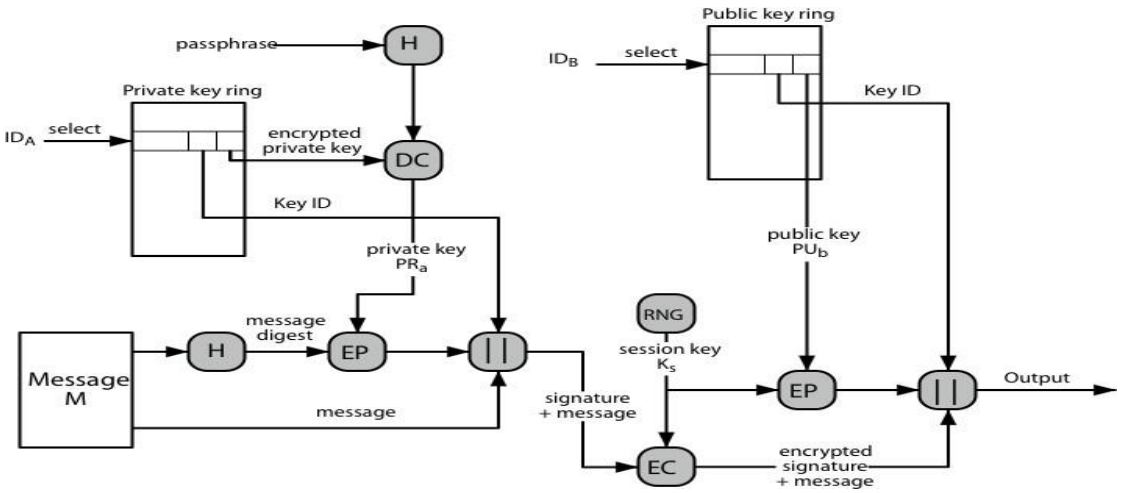
PGP Message Format



PGP Key Rings

Keys & key IDs are critical to the operation of PGP. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. PGP uses a pair of data structures, one to store the users public/private key pairs - their private-key ring; and one to store the public keys of other known users, their public-key ring. The private keys are kept encrypted using a block cipher, with a key derived by hashing a pass-phrase which the user enters whenever that key needs to be used. As in any system based on passwords, the security of this system depends on the security of the password, which should be not easily guessed but easily remembered.

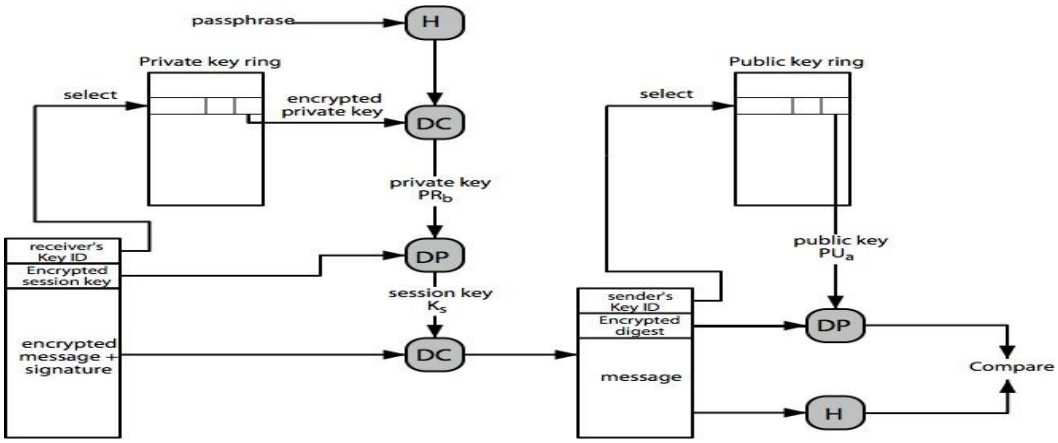
PGP Message Generation



Stallings Figure 18.5 illustrates how these key rings are used in message transmission to implement the various PGP crypto services (ignoring compression and radix-64 conversion for simplicity). The sending PGP entity performs the following steps:

1. Signing the message:
   a. PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.
   b. PGP prompts the user for the passphrase to recover the unencrypted private key.
   c. The signature component of the message is constructed.
2. Encrypting the message:
   a. PGP generates a session key and encrypts the message.
   b. PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.
   c. The session key component of the message is constructed.

PGP Message Reception



S/MIME. S/MIME (Secure/Multipurpose Internet Mail Extensions

S/MIME (Secure/Multipurpose Internet Mail Extension) is a security enhancement to the MIME Internet e-mail format standard, which in turn provided support for varying content types and multi-part messages over the text only support in the original Internet RFC822 (now RFC5322) email standard. See text for discussion of these extensions. MIME is specified in RFCs 2045 through 2049. MIME allows encoding of binary data to textual form for transport over traditional RFC822 email systems. S/MIME support is now included in many modern mail agents.

S/MIME Functions

- enveloped data
  - encrypted content and associated keys
- signed data
  - encoded message + signed digest
- clear-signed data
  - cleartext message + encoded signed digest
- signed & enveloped data
  - nesting of signed & encrypted entities

S/MIME Cryptographic Algorithms

- digital signatures: DSS & RSA
- hash functions: SHA-1 & MD5
- session key encryption: ElGamal & RSA
- message encryption: AES, Triple-DES, RC2/40 and others
- MAC: HMAC with SHA-1
- have process to decide which algs to use

S/MIME Messages

S/MIME secures a MIME entity with a signature, encryption, or both. A MIME entity may be an entire message or one or more of the subparts of the message. The MIME entity plus some security related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce a PKCS, which refers to a set of public-key cryptography specifications issued by RSA Laboratories. A PKCS object is then treated as message content and wrapped in MIME. A range of S/MIME content-types are specified, as shown. See text for details of how these are used.

- S/MIME secures a MIME entity with a signature, encryption, or both
- forming a MIME wrapped PKCS(public-key cryptography specifications) object
- have a range of content-types:
  - enveloped data
  - signed data
  - clear-signed data
  - registration request
  - certificate only message

S/MIME Enhanced Security Services

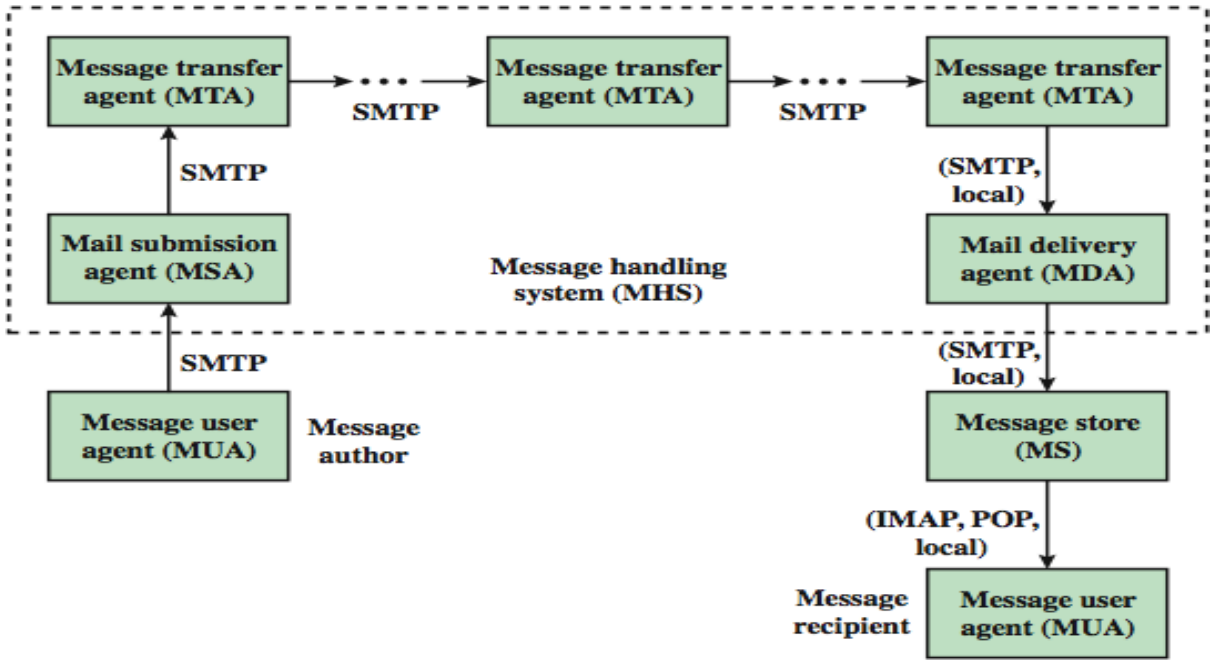
As of this writing, three enhanced security services have been proposed in an Internet draft, and may change or be extended. The three services are:

• **Signed receipts:** may be requested in a SignedData object to provide 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:** may be included in the authenticated attributes of a SignedData object, and is a set of security information regarding the sensitivity of the content that is protected by S/MIME encapsulation. They may be used for access control, indicating which users are permitted access to an object

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

Internet Mail Architecture



To understand to operation of DKIM, it is useful to have a basic grasp of the Internet mail architecture, as illustrated in Stallings Figure 18.9. At its most fundamental level, the Internet mail architecture consists of a user world in the form of Message User Agents (MUA), and the transfer world, in the form of the Message Handling Service (MHS), which is composed of Message Transfer Agents (MTA). A MUA is usually housed in the user's computer, and referred to as a client email program, or on a local network email server. The MHS accepts a message from one User and delivers it to one or more other users, creating a virtual MUA-to-MUA exchange environment. The MSA accepts the message submitted by an MUA and enforces the policies of the hosting domain and the requirements of Internet standards. This function may be co-located with the MUA or be a separate functional model. In the latter case, the Simple Mail Transfer Protocol (SMTP) is used between the MUA and the MSA. The MTA relays mail for one application-level hop. Relaying is performed by a sequence of MTAs, until the message reaches a destination MDA. The MDA is responsible for transferring the message from the MHS to the MS. An MS can be located on a remote server or on the same machine as the MUA. Typically, an MUA retrieves messages from a remote server using POP (Post Office Protocol) or IMAP (Internet Message Access Protocol). Also an administrative management domain (ADMD) is an Internet email provider. The Domain Name System (DNS) is a directory lookup service that provides a mapping between the name of a host on the Internet and its numerical address.