KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module-2 (E-mail Security)

Pretty Good Privacy (PGP) – Operational Description, Cryptographic keys and key rings, Message format, PGP message generation, PGP message reception, Public key management.

S/MIME – Functionality, Messages, Certificate processing, Enhanced security services.

Electronic Mail Security

- In all distributed environments, electronic mail is the most heavily used network based application.
- The demand for the authentication and confidentiality services grows with the explosive use of electronic mail for different purposes.
- currently message contents are not secure
- may be inspected either in transit
- or by suitably privileged users on destination system

Email Security Enhancements

- Confidentiality
  - protection from disclosure
- Authentication
  - of sender of message
- message integrity
  - protection from modification
- non-repudiation of origin
  - protection from denial by sender

Electronic Mail Security

- Two schemes
  - Pretty Good Privacy (PGP)
  - Secure/Multipurpose Internet Mail Extension
  - (S/MIME)

Pretty Good Privacy(PGP)

- Developed by Phil Zimmermann
- Selected best available cryptographic algorithms as the best building blocks to use
- Integrated these algorithms into a single program
- Originally free, now also have commercial versions available

NOTATION

K_s =session key used in symmetric encryption scheme

PR_a =private key of user A, used in public-key encryption scheme

PU_a =public key of user A, used in public-key encryption scheme

EP = public-key encryption

DP = public-key decryption

EC = symmetric encryption

DC = symmetric decryption

H = hash function

|| = concatenation

Z = compression using ZIP algorithm

R64 = conversion to radix 64 ASCII format

PGP OPERATIONAL DESCRIPTION

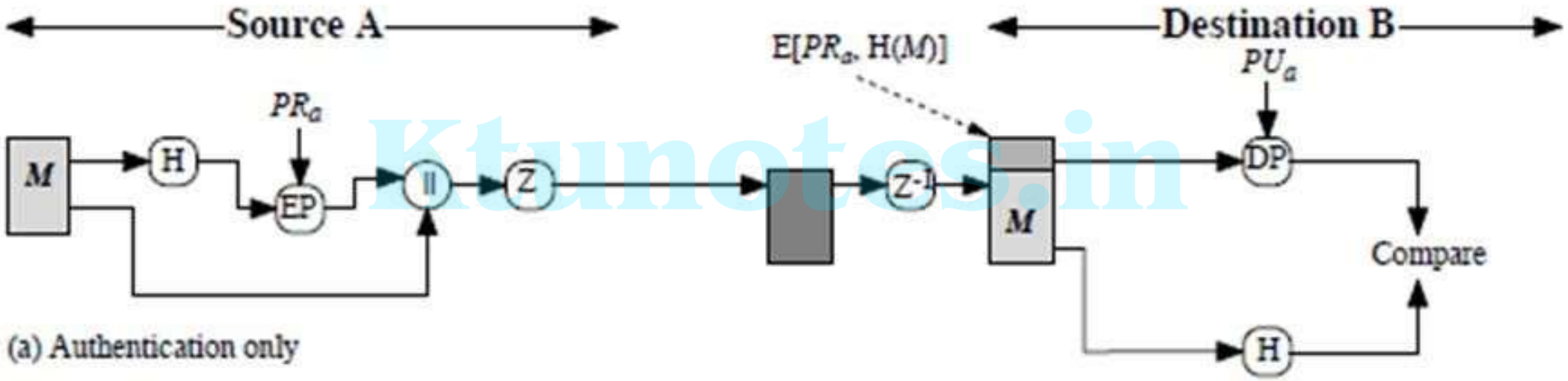
PGP consists of five services

- Authentication
- Confidentiality
- Compression
- e-mail Compatibility
- Segmentation

PGP Operation-Authentication

1. Sender creates message
2. Use SHA-1 to generate 160-bit hash of message
3. Signed hash with RSA using sender's private key, and is attached to message
4. Receiver uses RSA with sender's public key to decrypt and recover hash code
5. Receiver verifies received message using hash of it and compares with decrypted hash code

Authentication



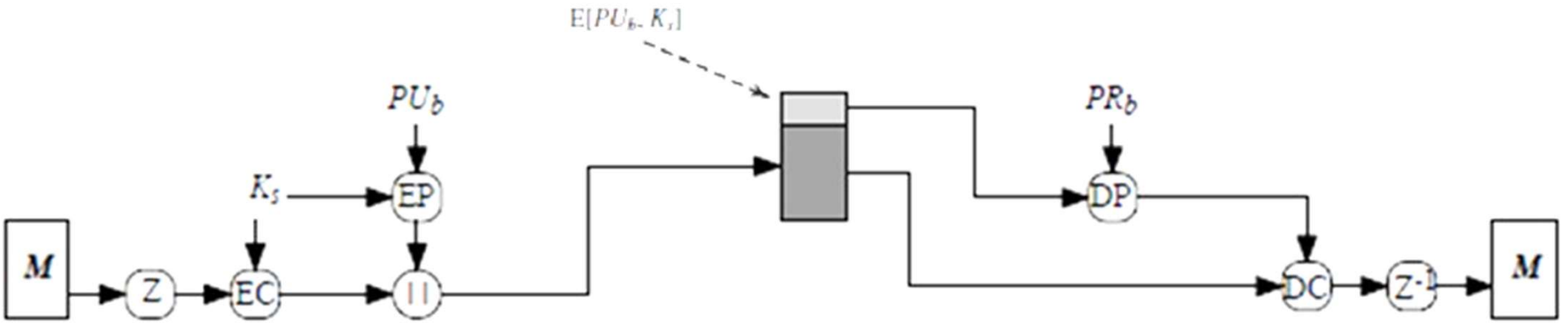
(a) Authentication only

Confidentiality

1. Sender generates message and 128-bit random number as session key for it
2. Encrypt message using CAST-128 / IDEA / 3DES with session key

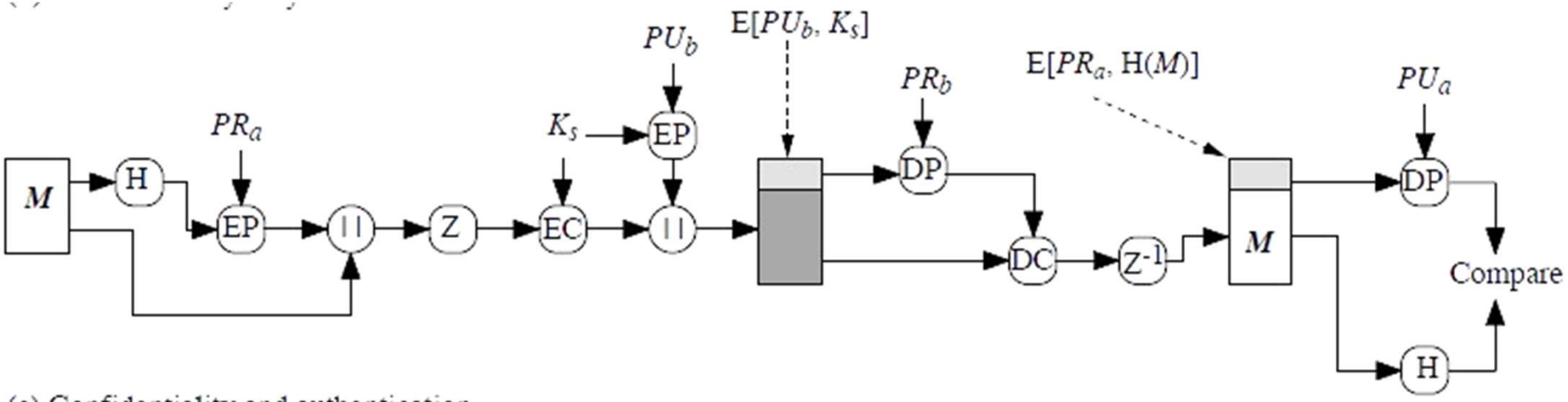
- Session key encrypted using RSA with recipient's public key, & attached to msg
- Receiver uses RSA with private key to decrypt and recover session key
- Session key is used to decrypt message

Confidentiality



(b) Confidentiality only

Confidentiality and Authentication



(c) Confidentiality and authentication

Compression

- By default PGP compresses message after signing but before encrypting
  - so can store uncompressed message & signature for later verification
  - & because compression is non deterministic
- uses ZIP compression algorithm

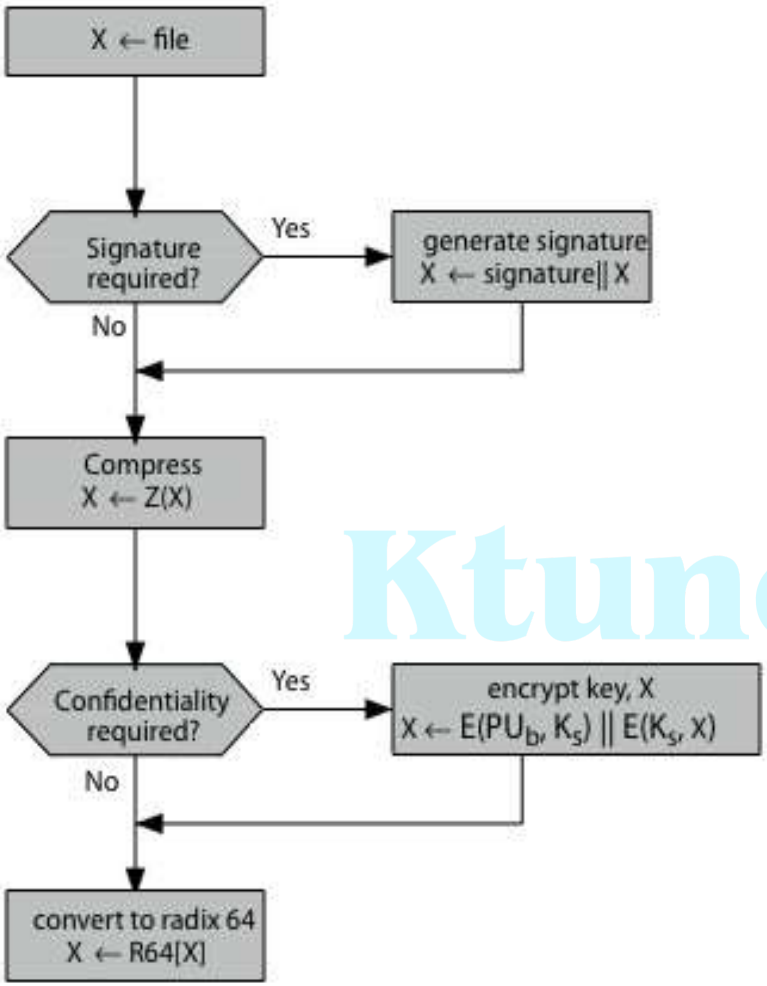
- The signature is generated before compression for two reasons:

a. 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.

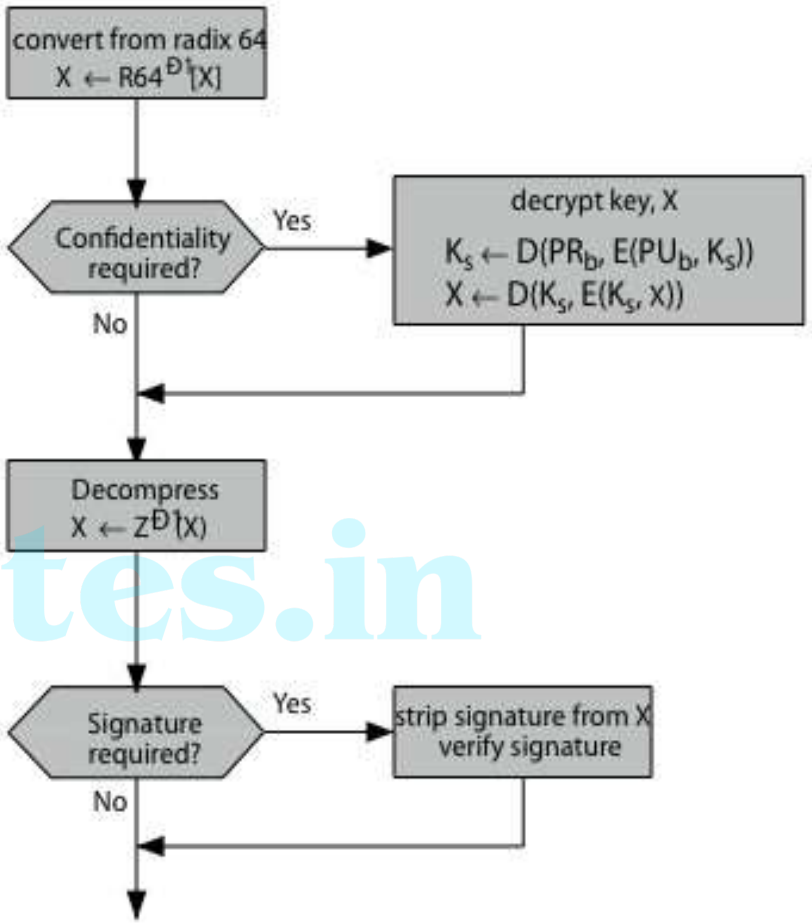
b. Even if one were willing to generate dynamically a recompressed message for verification, PGP's compression algorithm presents a difficulty. The algorithm is not deterministic; various implementations of the algorithm achieve different tradeoffs in running speed versus compression ratio and, as a result, produce different compressed forms. However, these different compression algorithms are interoperable because any version of the algorithm can correctly decompress the output of any other version.

e-mail Compatibility

- When using PGP will have binary data to send (encrypted message etc)
- However email was designed only for text
- Hence PGP must encode raw binary data into printable ASCII characters
- Uses radix-64 algorithm
  - Maps 3 bytes to 4 printable chars
  - Also appends a CRC
- PGP also segments messages if too big
  - It is reassembled by receiver before doing any operations



(a) Generic Transmission Diagram (from A)



(b) Generic Reception Diagram (to B)

Transmission and Reception of PGP Messages – Summary

CRYPTOGRAPHIC KEYS AND KEY RINGS

- PGP uses four types of keys
- One –time session symmetric keys, public keys, private keys and passphrase-based symmetric keys.

Three separate requirements can be identified with respect to these keys:

- A means of generating unpredictable session keys is needed.
- We would like to allow a user to have multiple public-key/private-key pairs. One reason is that the user may wish to change his or her key pair from time to time. Thus, some means is needed for identifying particular keys.
- Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents

Session Key Generation

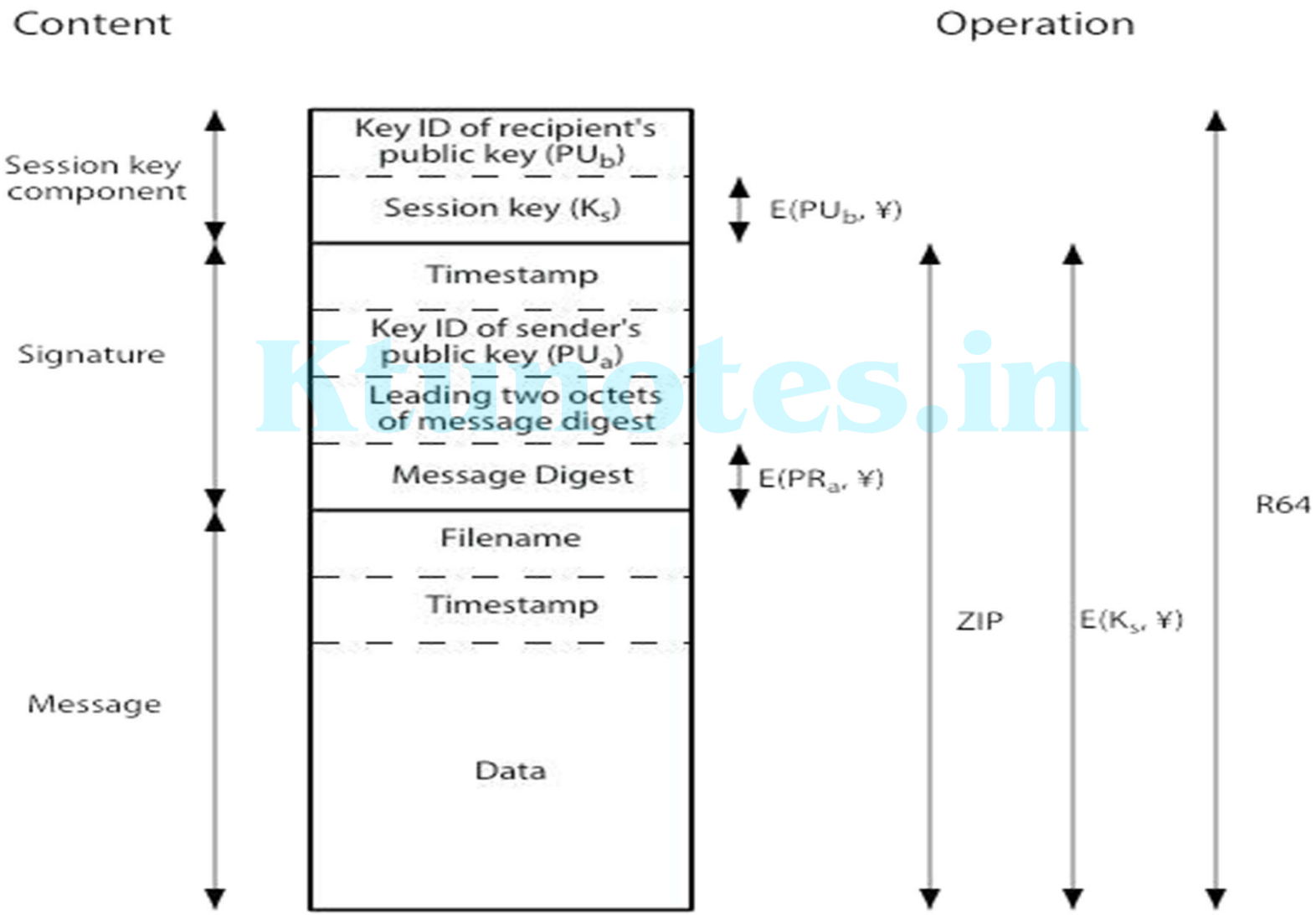
- Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message.
- Random 128-bit numbers are generated using CAST-128 itself. The input to the random number generator consists of a 128-bit key and two 64-bit blocks that are treated as plaintext to be encrypted.
- The algorithm that is used is based on the one specified in ANSI X12.17.
- The "plaintext" input to the random number generator, consisting of two 64-bit blocks, is itself derived from a stream of 128-bit randomized numbers. These numbers are based on keystroke input from the user. Both the keystroke timing and the actual keys struck are used to generate the randomized stream.

PGP Public & Private Keys

Key Identifiers

- Since many public/private keys may be in use, need to identify which is actually used to encrypt session key in a message
    - could send full public-key with every message
    - but this is inefficient
- Rather use a key identifier based on key
    - is least significant 64-bits of the key
    - will very likely be unique
- Also use key ID in signatures

PGP Message Format



A message consists of three components: the message component, a signature (optional), and a session key component (optional).

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

The ***signature component*** includes the following:

1. **Timestamp**: The time at which the signature was made.
2. • **Message digest**: The 160-bit SHA-1 digest, encrypted with the sender's private signature key. The digest is calculated over the signature timestamp concatenated with the data portion of the message component.
3. • **Leading two octets of message digest**: To enable the recipient to determine if the correct public key was used to decrypt the message digest for authentication, by comparing this plaintext copy of the first two octets with the first two octets of the decrypted digest. These octets also serve as a 16-bit frame check sequence for the message.
4. • **Key ID of sender's public key**: Identifies the public key that should be used to decrypt the message digest and, hence, identifies the private key that was used to encrypt the message digest.

The ***session key*** component includes the session key and the identifier of the recipient's public key that was used by the sender to encrypt the session key. The entire block is usually encoded with radix-64 encoding.

PGP Key Rings

- We have seen how key IDs are critical to the operation of PGP and that two key IDs are included in any PGP message that provides both confidentiality and authentication. These keys need to be stored and organized in a systematic way for efficient and effective use by all parties. The scheme used in PGP is to provide a pair of data structures at each node, one to store the public/private key pairs owned by that node and one to store the public keys of other users known at this node. These data structures are referred to, respectively, as the private-key ring and the public-key ring

PGP Key Rings

- Each PGP user has a pair of keyrings:
  - public-key ring contains all the public-keys of other PGP users known to this user, indexed by key ID
  - private-key ring contains the public/private key pair(s) for this user, indexed by key ID & encrypted keyed from a hashed passphrase
- Security of private keys thus depends on the pass-phrase security

Private-Key Ring

Timestamp	Key ID*	Public Key	Encrypted Private Key	User ID*
• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	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	$PU_i \bmod 2^{64}$	PU_i	$trust\_flag_i$	User i	$trust\_flag_i$		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

* = field used to index table

Figure 18.4 General Structure of Private- and Public-Key Rings

Figure shows the general structure of a private-key ring.
We can view the ring as a table, in which each row represents one of the public/private key pairs owned by this user.
Each row contains the following entries:

- Timestamp: The date/time when this key pair was generated.

- Key ID: The least significant 64 bits of the public key for this entry.

- Public key: The public-key portion of the pair.

- Private key: The private-key portion of the pair; this field is encrypted.

- User ID: Typically, this will be the user's e-mail address (e.g., stallings@acm.org). However, the user may choose to associate a different name with each pair (e.g., Stallings, WStallings, WilliamStallings, etc.) or to reuse the same User ID more than once

The private key is encrypted using CAST-128 (or IDEA or 3DES). The procedure is as follows:

1. The user selects a passphrase to be used for encrypting private keys.

2. When the system generates a new public/private key pair using RSA, it asks the user for the passphrase. Using SHA-1, a 160-bit hash code is generated from the passphrase, and the passphrase is discarded.

3. The system encrypts the private key using CAST-128 with the 128 bits of the hash code as the key. The hash code is then discarded, and the encrypted private key is stored in the private-key ring.

Subsequently, when a user accesses the private-key ring to retrieve a private key, he or she must supply the passphrase. PGP will retrieve the encrypted private key, generate the hash code of the passphrase, and decrypt the encrypted private key using CAST-128 with the hash code.

Figure also shows the general structure of a public-key ring. This data structure is used to store public keys of other users that are known to this user

- Timestamp: The date/time when this entry was generated.

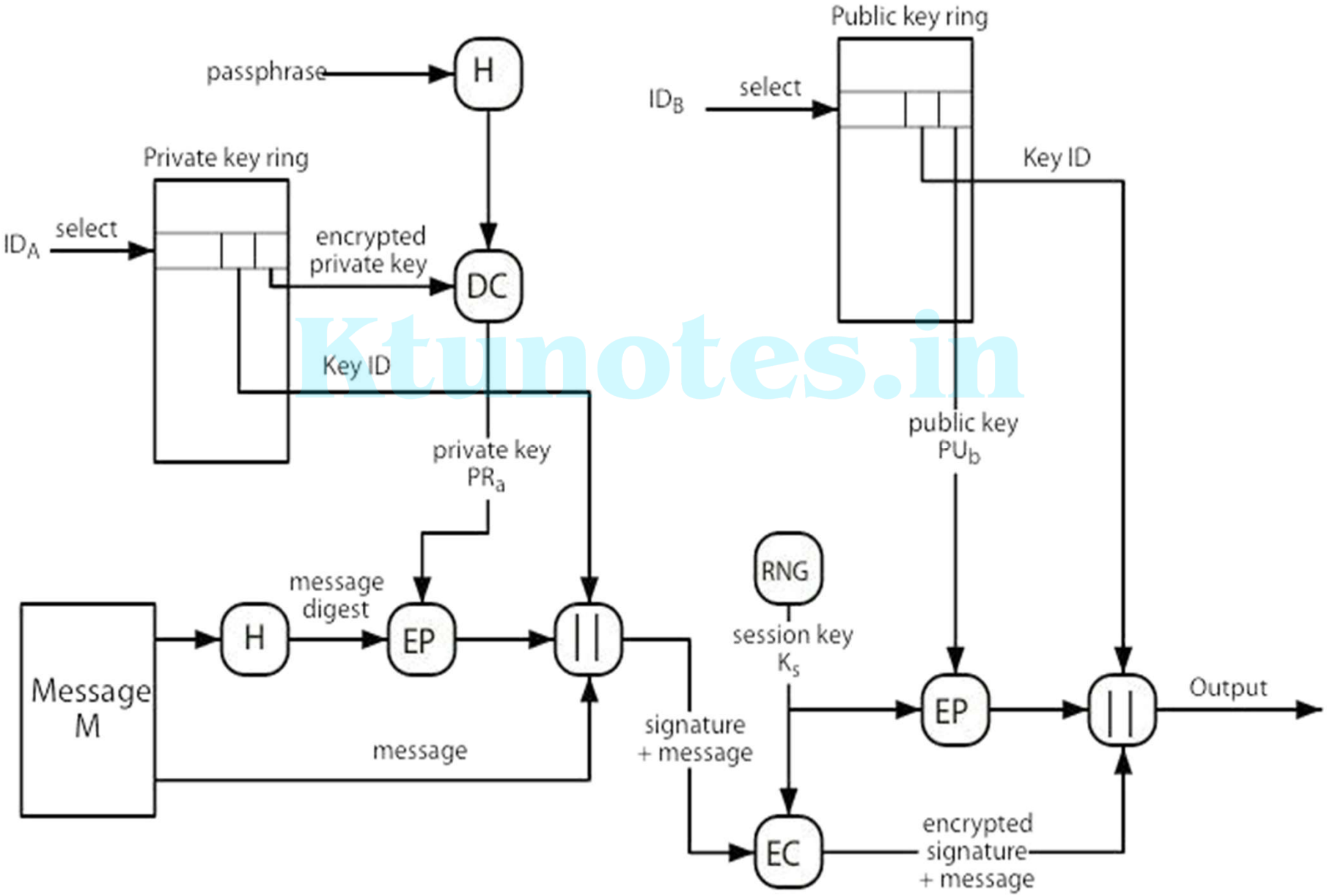
- Key ID: The least significant 64 bits of the public key for this entry.

- Public Key: The public key for this entry.

- User ID: Identifies the owner of this key. Multiple user IDs may be associated with a single public key. The public-key ring can be indexed by either User ID or Key ID

PGP Message Generation

PGP Message Generation (from User A to User B; no compression or radix 64 conversion)



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.

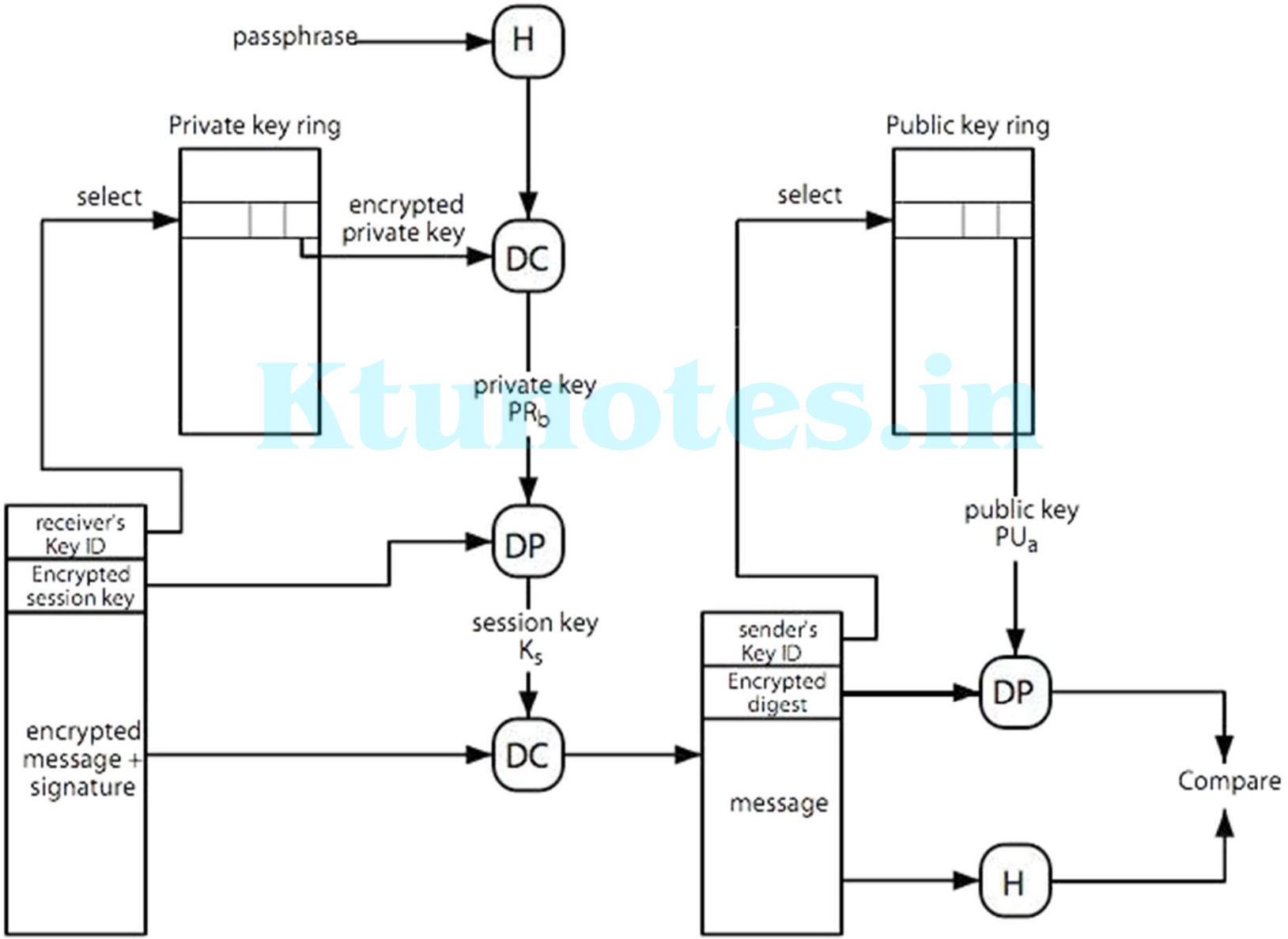
   b. PGP prompts the user for the passphrase to recover the unencrypted private key.

   c. The signature component of the message is constructed.

2. ***Encrypting the message***

- PGP generates a session key and encrypts the message.
- PGP retrieves the recipient's public key from the public-key ring using her_userid as an index.
- The session key component of the message is constructed.

PGP Message Reception



1. ***Decrypting the message***

a. 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.
b. PGP prompts the user for the passphrase to recover the unencrypted private key.
c. PGP then recovers the session key and decrypts the message.

2. Authenticating the message

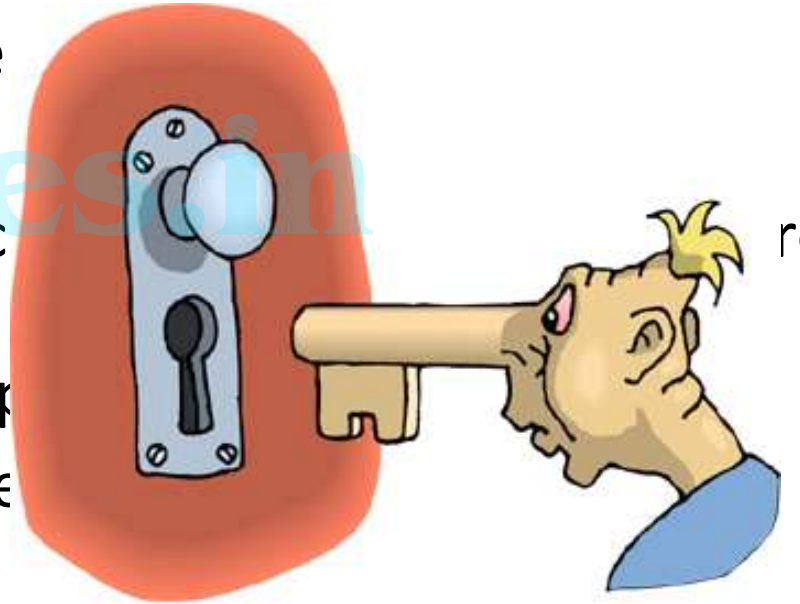
- PGP retrieves the sender's public key from the public-key ring, using the Key ID field in the signature key component of the message as an index.
- PGP recovers the transmitted message digest.
- PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate

PGP Key Management

- Rather than relying on certificate authorities
- In PGP every user is own CA
  - can sign keys for users they know directly
- Forms a "web of trust"
  - trust keys have signed
  - can trust keys others have signed if have a chain of signatures to them
- Key ring includes trust indicators
- Users can also revoke their keys

PGP(Pretty Good Privacy)

- Key Management for PGP
  - Public keys for encrypting session keys / verifying signatures.
  - Private keys for decrypting session keys / creating signatures.
  - Where do these keys come from and on what basis can they be trusted?
- PGP adopts a trust model called the
- No centralised authority
- Individuals sign one another's public re stored along with keys in key rings.
- PGP computes a *trust level* for each p
- Users interpret trust level for themse



PGP

Security of PGP

- There are many known attacks against PGP.
- Attacks against crypto algorithms are not the main threat
- IDEA is considered strong, and while cryptoanalysis advances, it should be strong still for some time.
- RSA may or may not be strong. There are recent rumors of possible fast factorization algorithms..
- The main threats are much more simple.
- An attacker may socially engineer himself into a web of trust, or some trustable person may change. Then he could falsify public keys. This breaks most of the security.
- PGP binaries can be corrupted when they are obtained.
- The PGP binaries can be modified in the computer.
- The passphrase can be obtained by a Trojan. Weak passphrases can be cracked.
- On multiuser system, access to the secret key can be obtained.

Module-2 (E-mail Security)

Pretty Good Privacy (PGP) – Operational Description, Cryptographic keys and key rings, Message format, PGP message generation, PGP message reception, Public key management.

S/MIME – Functionality, Messages, Certificate processing, Enhanced security services.

Secure/Multipurpose Internet Mail Extension (S/MIME)

- Security enhancement to MIME email
  - Original internet RFC822 email was text only
  - MIME provided support for varying content types and multi-part messages
  - With encoding of binary data to textual form
  - S/MIME added security enhancements

RFC 822

- Defines format for text messages sent using e-mail
- Messages have :
  - Envelop
  - Contents
- RFC 822 applies only to contents
- Contents consists of
  - header field,
  - followed by a blank line and
  - the body

RFC 822: Sample

```
Date: October 8, 2009 2:15:49 PM EDT
From: "William Stallings" <ws@shore.net>
Subject: The Syntax in RFC 5322
To: Smith@Other-host.com
Cc: Jones@Yet-Another-Host.com

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

Limitations of RF822/SMTP

- Can’t transmit exe files or binary objects
- National language characters not supported
- Mails messages limited to a certain size
- Doesn’t support non textual data

MIME(MULTIPURPOSE INTERNET MAIL EXTENSION)

MIME includes

1. Five New header fields
2. A number of content formats
3. Transfer encoding for protection against alteration

MIME: Header fields

1. MIME - version
2. Content - type
3. Content-Transfer-Encoding
4. Content-ID
5. Content-Description

- **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

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

MIME Transfer Encoding

- To provide reliable delivery
- Two methods of encoding data:
  1. Quoted –printable
  2. Base64
- Another encoding is x-token
- Application specific encoding

MIME Transfer Encodings

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

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 Functions

- Enveloped data
  - Encrypted content and associated keys
- Signed data
  - Encoded message + signed digest
- Clear-signed data
  - Clear text message + encoded signed digest
- Signed & enveloped data
  - Nesting of signed & encrypted entities

- **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.

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
- Have process to decide which algorithms to use

Function	Requirement
Create a message digest to be used in forming a digital signature. Encrypt message digest to form digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility. Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. 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 Sending agents SHOULD support encryption with AES.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Rece ing a ents SHOULD support HMAC with

S/MIME Messages

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

Table 15.7. S/MIME Content Types

(This item is displayed on page 468 in the print version)

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 signedData	An entity containing only public- key certificates.
	pkcs 7-mime	CompressedData	A compressed S/MIME entity
	pkcs 7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

S/MIME Message Preparation

- 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.
- The MIME entity is prepared according to the normal rules for MIME message preparation. Then the MIME entity plus some security-related data, such as algorithm identifiers and certificates, are processed by S/MIME to produce what is known as a PKCS object.
- A PKCS object is then treated as message content and wrapped in MIME (provided with appropriate MIME headers). This process should become clear as we look at specific objects and provide examples

- 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. For most cases, the result of applying the security algorithm will be to produce an object that is partially or totally represented in arbitrary binary data.

- This will then be wrapped in an outer MIME message and transfer encoding can be applied at that point, typically base64.

S/MIME content-types:

i. Enveloped data
ii. Signed data
iii. Clear-signed data
iv. Registration request
v. Certificate only message

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 BER format consists of arbitrary octet strings and is therefore binary data. Such an object should be transfer encoded with base64 in the outer MIME message. We first look at envelopedData.

Enveloped data

I. Generate a pseudorandom session key for a particular symmetric encryption algorithm.
II. For each recipient, encrypt the session key with the recipients public RSA key.
III. For each recipient, prepare **RecipientInfo** block containing recipient's public-key certificate, id of algo used to encrypt the session key, encrypted session key.
IV. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. This information is then encoded into base64. A sample message (excluding the RFC 822 headers) is the following:

```
Content-Type: application/pkcs7-mime; smime-type=enveloped-
       data; name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
rfvbnj75.6tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```


To recover the encrypted message, the recipient first 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.

Signed data

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

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.

Clear Signing

- Achieved using the multipart content type with a signed subtype
- Message content 'is clear'
  - Messages are not encoded
  - Signature part is encoded
  - Thus, recipients with MIME capability but not S/MIME capability are able to read the incoming message

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

Certificate-Only Messages

- 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

- **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. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities

•**User Agent Role**

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

● **Key generation**: The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of 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.

Enhanced Security Services

- As of this writing, three enhanced security services have been proposed in an Internet draft. The details of these may change, and additional services may be added. The three services are as follows.
- **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. In essence, the recipient signs the entire original message plus original (sender's) signature and appends the new signature to form a new S/MIME 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 permitted access to an object. Other uses include priority (secret, confidential, restricted, and so on) or role based, describing which kind of people can see the information (e.g., patient's health-care team, medical billing agents, etc.).

- **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