Pretty Good Privacy

In virtually all distributed environments, electronic mail is the most heavily used network-based application. But current email services are roughly like "postcards", anyone who wants could pick it up and have a look as it's in transit or sitting in the recipients mailbox. PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. With the explosively growing reliance on electronic mail for every conceivable purpose, there grows a demand for authentication and confidentiality services.

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 is independent of government organizations and runs on a wide range of systems, in both free & commercial versions.

*There are **five** important services in PGP*

- *Authentication (Sign/Verify)*
- *Confidentiality (Encryption/Decryption)*
- *Compression*
- *Email compatibility*
- *Segmentation and Reassembly*
- ✓ The last three are **transparent** to the user

PGP Notations:

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

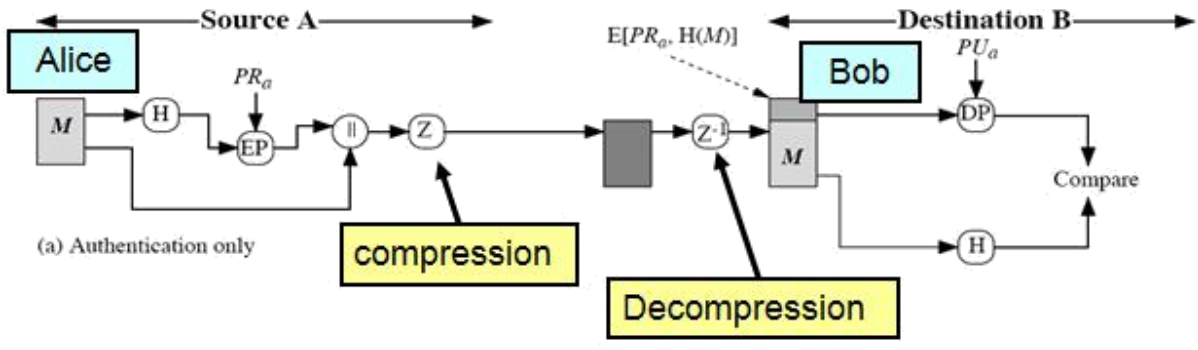
H = hash function

|| = concatenation

Z = compression using ZIP algorithm R64 =

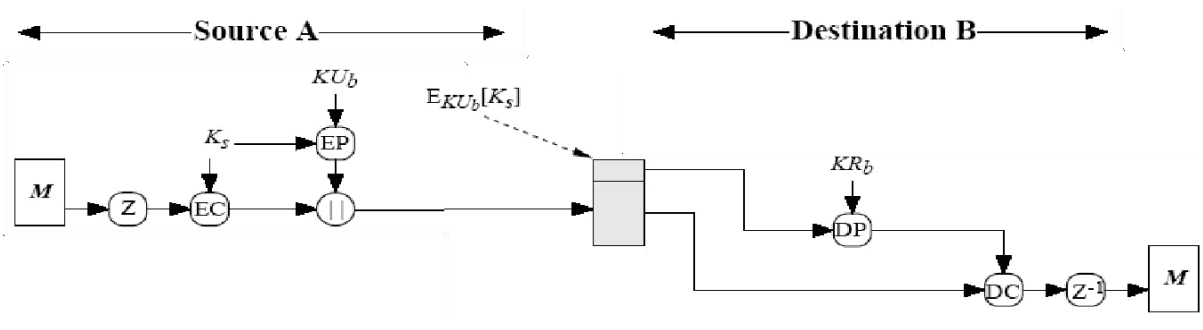
conversion to radix 64 ASCII format

PGP 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

PGP Operation- Confidentiality



Sender:

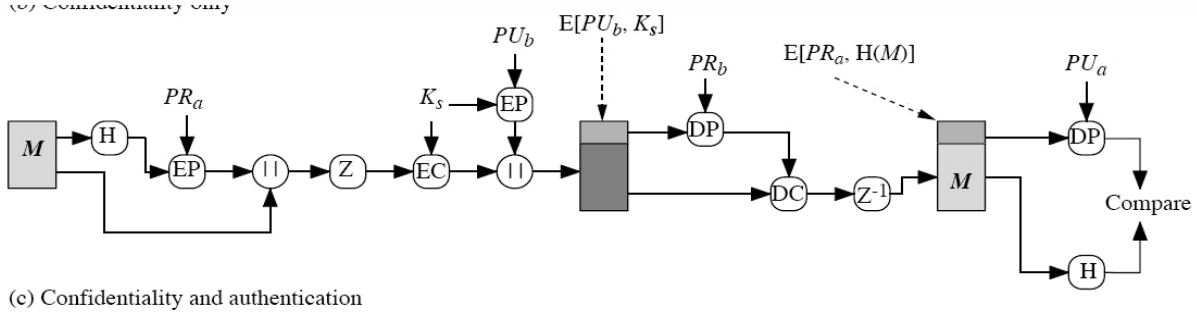
1. Generates message and a random number (session key) only for this message
2. Encrypts message with the session key using AES, 3DES, IDEA or CAST-128
3. Encrypts session key itself with recipient's public key using RSA
4. Attaches it to message

Receiver:

1. Recovers session key by decrypting using his private key
2. Decrypts message using the session key

Confidentiality service provides no assurance to the receiver as to the identity of sender (i.e. no authentication). Only provides confidentiality for sender that only the recipient can read the message (and no one else)

PGP Operation – Confidentiality & Authentication



(c) Confidentiality and authentication

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

PGP Operation – Compression

As a default, PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage. The placement of the compression algorithm, indicated by Z for compression and Z^{-1} for decompression is critical. The compression algorithm used is ZIP.

- The signature is generated before compression for two reasons:

  1. so that one can store only the uncompressed message together with signature for later verification
  2. Applying the hash function and signature after compression would constrain all PGP implementations to the same version of the compression algorithm as the PGP compression algorithm is not deterministic

- 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.

PGP Operation – Email Compatibility

When PGP is used, at least part of the block to be transmitted is encrypted, and thus consists of a stream of arbitrary 8-bit octets. However many electronic mail systems only permit the use of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters. It uses radix-64 conversion, in which each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message by 33%, but still an overall compression of about one-third can be achieved.

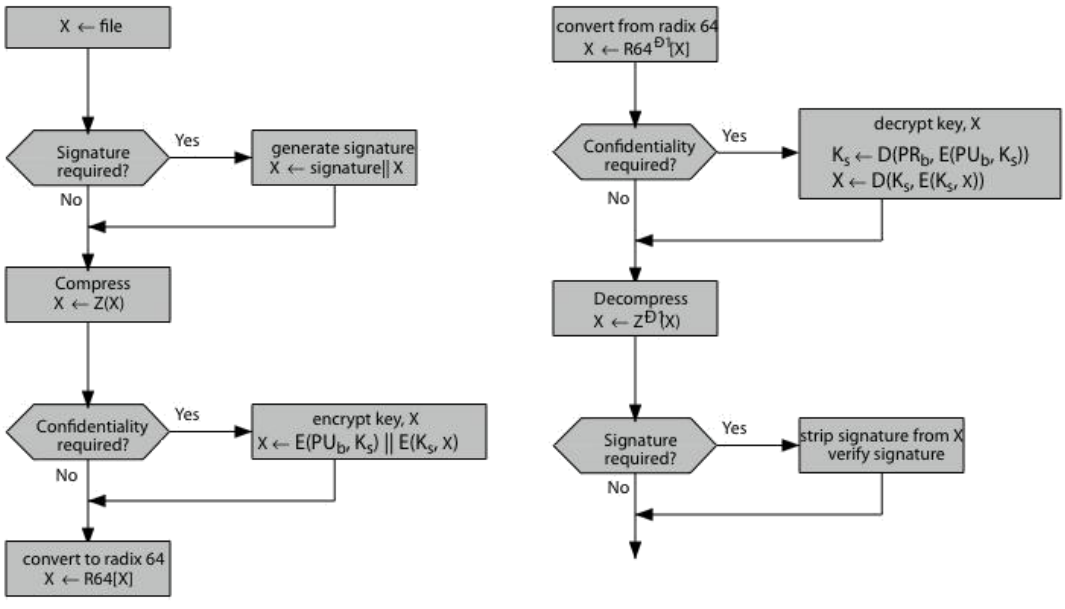
PGP Operation - Segmentation/Reassembly

E-mail facilities often are restricted to a maximum message length. For example, 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 of the other processing, including the radix-64 conversion. Thus, the session key component and signature component appear only once, at the beginning of the first segment. Reassembly at the receiving end is required before verifying signature or decryption

PGP Operations – Summary

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



(a) Generic Transmission Diagram (from A)

(b) Generic Reception Diagram (to B)

Cryptographic Keys and Key Rings

PGP makes use of 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:

1. a means of generating unpredictable session keys is needed.
2. a user is allowed to have multiple public-key/private-key pairs.
3. Each PGP entity must maintain a file of its own public/private key pairs as well as a file of public keys of correspondents.

PGP Session Keys

Each session key is associated with a single message and is used only for the purpose of encrypting and decrypting that message. Random numbers are generated using the algorithm specified in ANSI X12.17, with inputs based on keystroke input from the user, where both the keystroke timing and the actual keys struck are used to generate a randomized stream of numbers.

Key Identifiers

In PGP, any given user may have multiple public/private key pairs. That means, a user may have many public/private key pairs at his disposal. He wishes to encrypt or sign a message using one of his keys. But, the problem of informing the other party, which key he has used arises. Attaching the whole public key every time is inefficient. Rather PGP uses ***a key identifier*** based on the least significant 64-bits of the key, which will very likely be

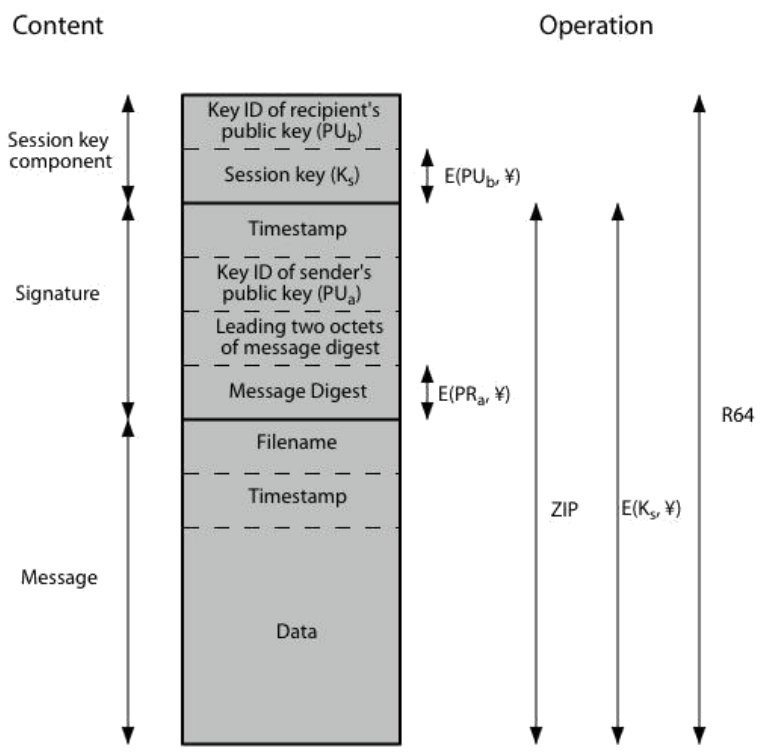
unique. That is, the key ID of public PU_a is (PU_a mod 2^{64}). Then only the much shorter key ID would need to be transmitted with any message. A key ID is also required for the PGP digital signature.

PGP Message Format

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:

- *Timestamp*: The time at which the signature was made.
- *Message digest*: The 160-bit SHA-1 digest, encrypted with the sender's private signature key.
- *Leading two octets of message digest*: 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.
- *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



Notation:

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

The *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

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 user's public/private key pairs - their ***private-key ring***; and one to store the public keys of other known users, their ***public-key ring***.

General Structure of Private- and Public-Key Rings

a) Private-Key Ring

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
. . .	. . .	. . .	. . .	. . .

The Private-Key ring can be viewed 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 ring is intended to be stored only on the machine of the user that created and owns the key pairs, and that it be accessible only to that user, it makes sense to make the value of the private key as secure as possible. Accordingly, the private key itself is not stored in the key ring. Rather, this 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. . 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.

b) <u>Public-key Ring</u>

This data structure is used to store public keys of other users that are known to this user.

Public-Key Ring

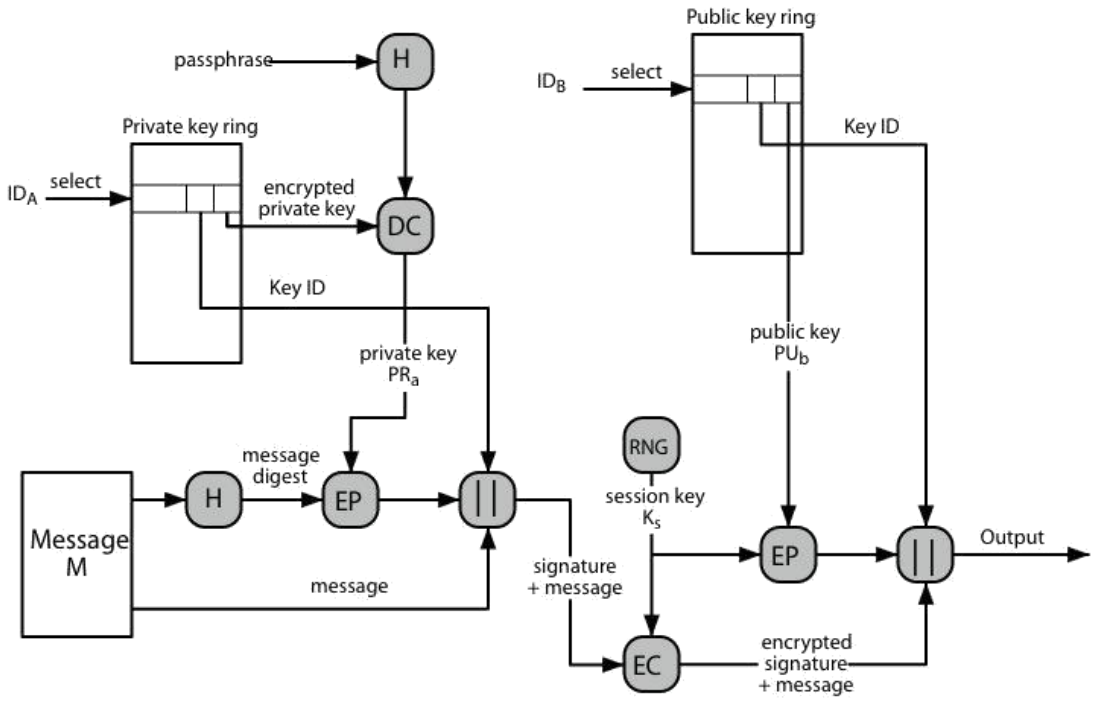
Timestamp	Key ID*	Public Key	Owner Trust	User ID*	Key Legitimacy	Signature(s)	Signature Trust(s)
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •
T_i	$PU_i \bmod 2^{64}$	PU_i	trust_flag_i	User i	trust_flag_i		
• • •	• • •	• • •	• • •	• • •	• • •	• • •	• • •

- 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

PGP Message Transmission and Reception

Message transmission

The following figure shows the steps during message transmission assuming that the message is to be both signed and encrypted.



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

The sending PGP entity performs the following steps:

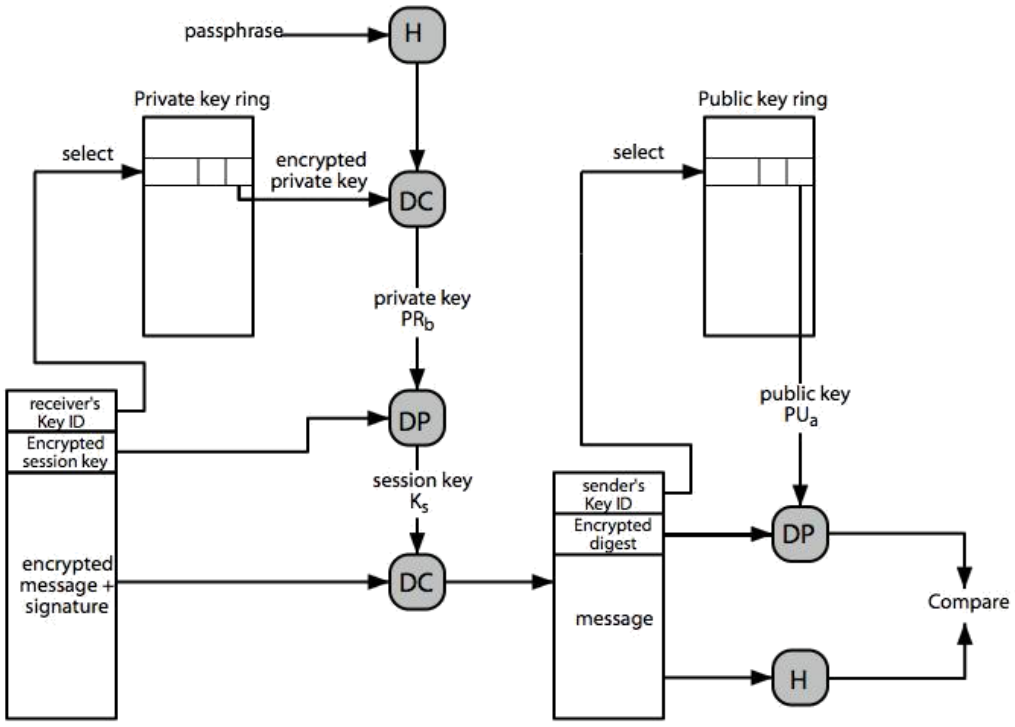
1. Signing the message

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

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.

Message Reception



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

The receiving PGP entity performs the following steps:

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

a. 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.
b. PGP recovers the transmitted message digest.
c. PGP computes the message digest for the received message and compares it to the transmitted message digest to authenticate.

Public Key Management

PGP contains a clever, efficient, interlocking set of functions and formats to provide an effective confidentiality and authentication service and also addresses the problem of public-key management.

Various Approaches for Public Key Management

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:

1. B could store her public key (PU_b) on a floppy disk and hand it to A. This is a very secure method but has obvious practical limitations.
2. B could transmit her key in an e-mail message to A. A could have PGP generate a 160-bit SHA-1 digest of the key and display it in hexadecimal format; this is referred to as the "*fingerprint*" of the key. A could then call B and ask her to dictate the fingerprint over the phone. If the two fingerprints match, the key is verified. This is a more practical approach and for this A has to recognize the voice of B over the telephone.
3. 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. D generates an SHA-1 digest of this certificate, encrypts it with her private key, and attaches the signature to the certificate. Because only D could have created the signature, no one else can create a false public key and pretend that it is signed by D. The signed certificate could be sent directly to A by B or D, or could be posted on a bulletin board.
4. 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.

The Use of Trust

PGP provides a better way of using trust, utilizing trust information and linking trust with public keys. The information about trust is stored in a ‘trust flag byte’. Its structure consists of three fields:

1. key legitimacy field – KEYLEGITFIELD
2. signature trust field – SIGTRUST FIELD
3. owner trust field – OWNERTRUST FIELD

Key Legitimacy Field

It is computed by PGP. This field specifies the level of PGP's trust about the validity of user's public key. Based on the extent of trust, the user ID is bound to the key. A KEYLEGIT field can hold the following information:

1. unknown or undefined trust
2. key ownership not trusted
3. marginal trust in key ownership
4. complete trust in key ownership

A WARNONLY bit is set if user wants only to be warned when key that is not fully validated is used for encryption

Signature Trust Field

A key ring owner collects all the signatures that are related to the entries. Each signature has its own signature-trust-field that specifies the level of PGP user's trust towards the signer, so that all its public keys can be certified. A SIGTRUST FIELD can hold values like:

1. undefined trust
2. unknown user
3. usually not trusted to sign other keys
4. usually trusted to sign other keys
5. always trusted to sign other keys
6. this key is present in secret key ring (ultimate trust)

It also has a CONTIG bit that is set if signature tends to a contiguous trusted certification path that will ultimately reach the trusted key ring owner

Owner Trust Field

Each entry in the public key ring represents a public key that is related to a particular owner along with a owner-trust-field. This field specifies the extent of trust towards the public key, so that it can be used to sign other public-key-certificates. User is supposed to assign this field. An OWNERTRUST field can hold values like:

1. undefined trust
2. unknown user
3. usually not trusted to sign other keys
4. usually trusted to sign other keys
5. always trusted to sign other keys
6. this key is present in secret key ring (ultimate trust)

It also has a BUCKSTOP bit that is automatically set, if the key is present in the secret key ring.

Operation of Trust Processing

Consider the public key ring of User-A, then the operation of trust processing is described as follows:

1. 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.
2. 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. Otherwise, PGP computes a weighted sum of the trust values. ~~A weight of 1/X is given to signatures that are always trusted and 1/Y to signatures that are usually trusted, where X and Y are user-configurable parameters. When the total of weights of the introducers of a key/UserID combination reaches 1, the binding is considered to be trustworthy, and the key legitimacy value is set to complete. Thus, in the absence of ultimate trust, at least X signatures that are always trusted or Y signatures that are usually trusted or some combination is needed.~~

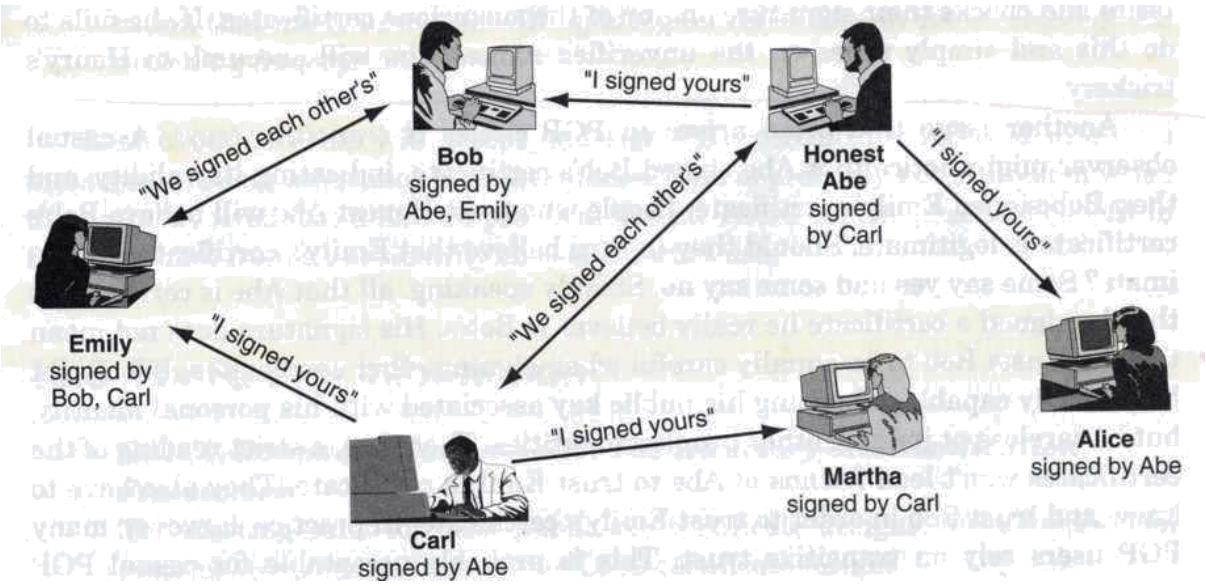
PGP scans the public key ring in a top-down manner for assuring consistency. Each OWNERTRUST field is scanned by PGP for all signatures with the authorization of that owner in order to update SIGTRUST field, so that it becomes equal to the OWNERTRUST field. To start this process, it selects the keys with ‘ultimate trust’ first and then determines all the KEYLEGIT fields that are based on the attached signatures.

Revoking Public Keys

When a user suspects that his opponent might have acquired his unencrypted private key or if he doesn’t want to use the same key for a long period, he must revoke(cancel) his current public key. In order to revoke a public key, the owner will have to issue a signed key revocation certificate. To sign this certificate, corresponding private key is used. This certificate is similar to that of the general signature certificates except that, this certificate is used for revoking its public key. The owner will then broadcast this certificate as soon as possible so that others can update their public key rings.

PGP "Web of Trust"

The idea behind the various trust fields in the public key ring is to establish a "Web of Trust" among a community of users.



If Alice trusts <u>only</u> Abe to sign certificates, then she won't believe certificates from Martha or Emily are genuine. If she also trusts Bob's judgment about signing certificates, she can trust Emily's certificate; if she also trusts Carl, she can trust everyone's certificate.

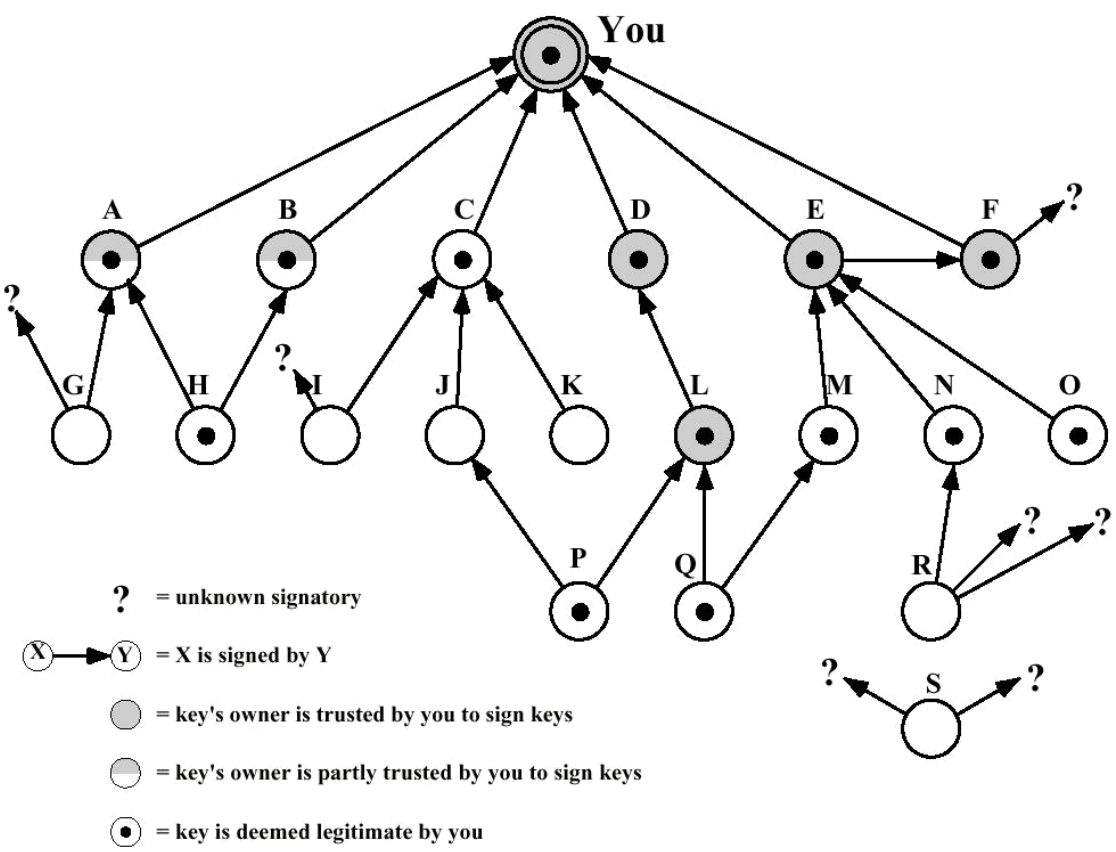


Figure 12.7 PGP Trust Model Example

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 email standard. MIME allows encoding of binary data to textual form for transport over traditional RFC822 email systems. S/MIME is defined in a number of documents, most importantly RFCs 3369, 3370, 3850 and 3851 and S/MIME support is now included in many modern mail agents.

RFC 822

RFC 822 defines a format for text messages that are sent using electronic mail and it has been the standard for Internet-based text mail message. The overall structure of a message that conforms to RFC 822 is very simple. A message consists of some number of header lines (the header) followed by unrestricted text (the body). The header is separated from the body by a blank line. A header line usually consists of a keyword, followed by a colon, followed by the keyword's arguments; the format allows a long line to be broken up into several lines. The most frequently used keywords are *From, To*, *Subject*, and *Date*.

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.

Problems with RFC 822 and SMTP

- Executable files or other binary objects must be converted into ASCII. Various schemes exist (e.g., Unix UUencode), but a standard is needed
- Text data that includes special characters (e.g., Hungarian text) cannot be transmitted as SMTP is limited to 7-bit ASCII
- Some servers reject mail messages over a certain size
- Some common problems exist with the SMTP implementations which do not adhere completely to the SMTP standards defined in RFC 821. They are:
    - delete, add, or reorder CR and LF characters
    - truncate or wrap lines longer than 76 characters
    - remove trailing white space (tabs and spaces)
    - pad lines in a message to the same length
    - convert tab characters into multiple spaces

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 822 implementations and the specification is provided in RFC's 2045 through 2049.

The MIME specification includes the following elements:

1. Five new message header fields are defined, which provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that protect the content from alteration by the mail system.

MIME - New header fields

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

MIME Content Types

The bulk of the MIME specification is concerned with the definition of a variety of content types. There are seven different major types of content and a total of 15 subtypes. In general, a content type declares the general type of data, and the subtype specifies a particular format for that type of data.

For the text type of body, 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. This boundary should not appear in any parts of the message. Each boundary starts on a new line and consists of two hyphens followed by the boundary value. The final boundary, which indicates the end of the last part, also has a suffix of two hyphens. Within each part, there may be an optional ordinary MIME header. There are four subtypes of the multipart type, all of which have the same overall syntax.

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

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. 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.

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.

The MIME standard defines two methods of encoding data. The Content-Transfer-Encoding field can actually take on six values. Three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. 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. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

The quoted-printable transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters. The base64 transfer encoding, also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail transport programs.

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.

Native Form	The body to be transmitted is created in the system's native format. The native character set is used and, where appropriate, local end-of-line conventions are used as well. The body may be a UNIX-style text file, or a Sun raster image, or a VMS indexed file, or audio data in a system-dependent format stored only in memory, or anything else that corresponds to the local model for the representation of some form of information. Fundamentally, the data is created in the "native" form that corresponds to the type specified by the media type.
Canonical Form	The entire body, including "out-of-band" information such as record lengths and possibly file attribute information, is converted to a universal canonical form. The specific media type of the body as well as its associated attributes dictate the nature of the canonical form that is used. Conversion to the proper canonical form may involve character set conversion, transformation of audio data, compression, or various other operations specific to the various media types. If character set conversion is involved, however, care must be taken to understand the semanties of the media type, which may have strong implications for any character set conversion (e.g. with regard to syntactically meaningful characters in a text subtype other than "plain").

S/MIME Functionality

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

Functions

S/MIME provides the following functions:

- **Enveloped data**: This consists of encrypted content of any type and encrypted-content encryption keys for one or more recipients.
- **Signed data**: A digital signature is formed by taking the message digest of the content to be signed and then encrypting that with the private key of the signer. The content plus signature are then encoded using base64 encoding. A signed data message can only be viewed by a recipient with S/MIME capability.
- **Clear-signed data**: As with signed data, a digital signature of the content is formed. However, in this case, only the digital signature is encoded using base64. As a result, recipients without S/MIME capability can view the message content, although they cannot verify the signature.
- **Signed and enveloped data**: Signed-only and encrypted-only entities may be nested, so that encrypted data may be signed and signed data or clear-signed data may be encrypted.

Cryptographic Algorithms

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.

The following table summarizes the cryptographic algorithms used in S/MIME.

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 RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with one-time session key.	Sending and receiving agents MUST support encryption with triple DES Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code	Receiving agents MUST support HMAC with SHA-1. Receiving agents SHOULD support HMAC with SHA-1.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys; in fact, S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal. As an alternative, RSA, can be used for both signatures and session key encryption. These are the same algorithms used in PGP and provide a high level of security. For the hash function used to create the digital signature, the specification requires the 160-bit SHA-1 but recommends receiver support for the 128-bit MD5 for backward compatibility with older versions of S/MIME. As there is justifiable concern about the security of MD5, SHA-1 is clearly the preferred alternative.

A sending agent has two decisions to make. First, the sending agent must determine if the receiving agent is capable of decrypting using a given encryption algorithm. Second, if the receiving agent is only capable of accepting weakly encrypted content, the sending agent must decide if it is acceptable to send using weak encryption. To support this decision process, a sending agent may announce its decrypting capabilities in order of preference any message that it sends out. A receiving agent may store that information for future use.

The following rules, in the following order, should be followed by a sending agent:

1. If the sending agent has a list of preferred decrypting capabilities from an intended recipient, it SHOULD choose the first (highest preference) capability on the list that it is capable of using.
2. If the sending agent has no such list of capabilities from an intended recipient but has received one or more messages from the recipient, then the outgoing message SHOULD use the same encryption algorithm as was used on the last signed and encrypted message received from that intended recipient.
3. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is willing to risk that the recipient may not be able to decrypt the message, then the sending agent SHOULD use tripleDES.
4. If the sending agent has no knowledge about the decryption capabilities of the intended recipient and is not willing to risk that the recipient may not be able to decrypt the message, then the sending agent MUST use RC2/40.

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown below:

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 Content Types

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. 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).

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 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.

The steps for preparing an envelopedData MIME entity are as follows:

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

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

3. For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate, [3] an identifier of the algorithm used to encrypt the session key, and the encrypted session key.

4. Encrypt the message content with the session key.

The RecipientInfo blocks followed by the encrypted content constitute the envelopedData. 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.

Clear Signing

Clear signing is achieved using the multipart content type with a signed subtype. This signing process does not involve transforming the message to be signed, so that the message is sent "in the clear." Thus, 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 micalg parameter indicates the type of message digest used. 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.

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. S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists, needed to verify incoming signatures and to encrypt outgoing messages. But certificates are signed by trusted 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.
- **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.

S/MIME – Certification Authorities

"Certificate Authority" (CA), or "Trust Center", is the name used for an organisation that acts as the agent of trust in a PKI (Public Key Infrastructure) and also for the piece of software. PKI needed for secure use of public key based protocols

A CA performs 5 main functions:

- Verifies users' identities - this may be done by the CA itself, or on its behalf by a Local Registration Authority (LRA)
- Issues users with keys (though sometimes users may generate their own key pair)
- Certifies users' public keys
- Publishes users'certificates
- Issues certificate revocation lists (CRLs)

VeriSign Certificates

There are several companies that provide certification authority (CA) services. VeriSign provides a CA service that is intended to be compatible with S/MIME and a variety of other applications. VeriSign issues X.509 certificates with the product name VeriSign Digital ID. The information contained in a Digital ID depends on the type of Digital ID and its use. At a minimum, each Digital ID contains

- Owner's public key
- Owner's name or alias
- Expiration date of the Digital ID

- Serial number of the Digital ID
- Name of the certification authority that issued the Digital ID
- Digital signature of the certification authority that issued the Digital ID

Digital IDs can also contain 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.

- 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.

Enhanced Security Services

Three enhanced security services have been proposed in an Internet draft. The three services are as follows:

- Signed receipts

APPENDIX•Securitylabels

- Secure mailing lists

Radix-64 Conversion

Both PGP and S/MIME make use of an encoding technique referred to as radix-64 conversion. This technique maps arbitrary binary input into printable character output. The form of encoding has the following relevant characteristics:

1. The range of the function is a character set that is universally representable at all sites, not a specific binary encoding of that character set.
2. The character set consists of 65 printable characters, one of which is used for padding. With 2^6 = 64 available characters, each character can be used to represent 6 bits of input
3. No control characters are included in the set
4. The hyphen character ("-") is not used.

6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding	6-bit value	character encoding
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/
						(pad)	=

For example, consider the 24-bit raw text sequence 00100011 01011100 10010001, which can be expressed in hexadecimal as 235C91. We arrange this input in blocks of 6 bits:

001000 110101 110010 010001

The extracted 6-bit decimal values are 8, 53, 50, 17. Looking these up in above table yields the radix-64 encoding as the following characters: I1yR. If these characters are stored in 8-bit ASCII format with parity bit set to zero, we have

01001001 00110001 01111001 01010010

In hexadecimal, this is 49317952. To summarize,

Input Data	
Binary representation	00100011 01011100 10010001
Hexadecimal representation	235C91
Radix-64 Encoding of Input Data	
Character representation	I1yR
ASCII code (8 bit, zero parity)	01001001 00110001 01111001 01010010
Hexadecimal representation	49317952

Assignment Questions

1. (a) Explain the following terms in relation with the e-mail software - PGP:
   i. E-mail compatibility
   ii. Segmentation and reassembly. [8+8]
   (b) Describe how authentication and confidentiality are handled in S/MIME.

2. (a) Explain the importance and usage of the following in relation to PGP:
   i. Session key
   ii. Signature
   iii. Public / Private keys.
   (b) Describe how S/MIME works towards emerging as an industry standard for e-mail security at commercial and organizational use levels. [8+8]

3. (a) Explain how the exchange of secret key takes place between 'X' and 'Y' users with PGP.
   (b) List limitations of SMTP and MIME Write about the SMIME messages

4. (a) Explain why PGP generates a signature before applying the compression.
   (b) Discuss the requirement of segmentation and reassembly function in PGP.
   (c) write about MIME Content types.

5. (a) Explain the general format of a PGP message with a pictorial representation.
   (b) What is a Certification Authority and explain its role in S/MIME.

6. (a) Compare and contrast the key management in PGP and S/MIME.
   (b) Write about how PGP messages are created.

7. (a) What is Radix-64 format? Explain how both PGP and S/MIME perform the Radix-64 conversion is performed.
   (b) Describe the five principal services that Pretty Good Privacy (PGP) provides.

8. (a) Describe PGP session key generation
   (b) Explain the functionality of S/MIME