

# E-mail Security

## PGP, S/MIME

### Certificates and PKI

---

---

This chapter discusses two protocols providing security services for e-mails:

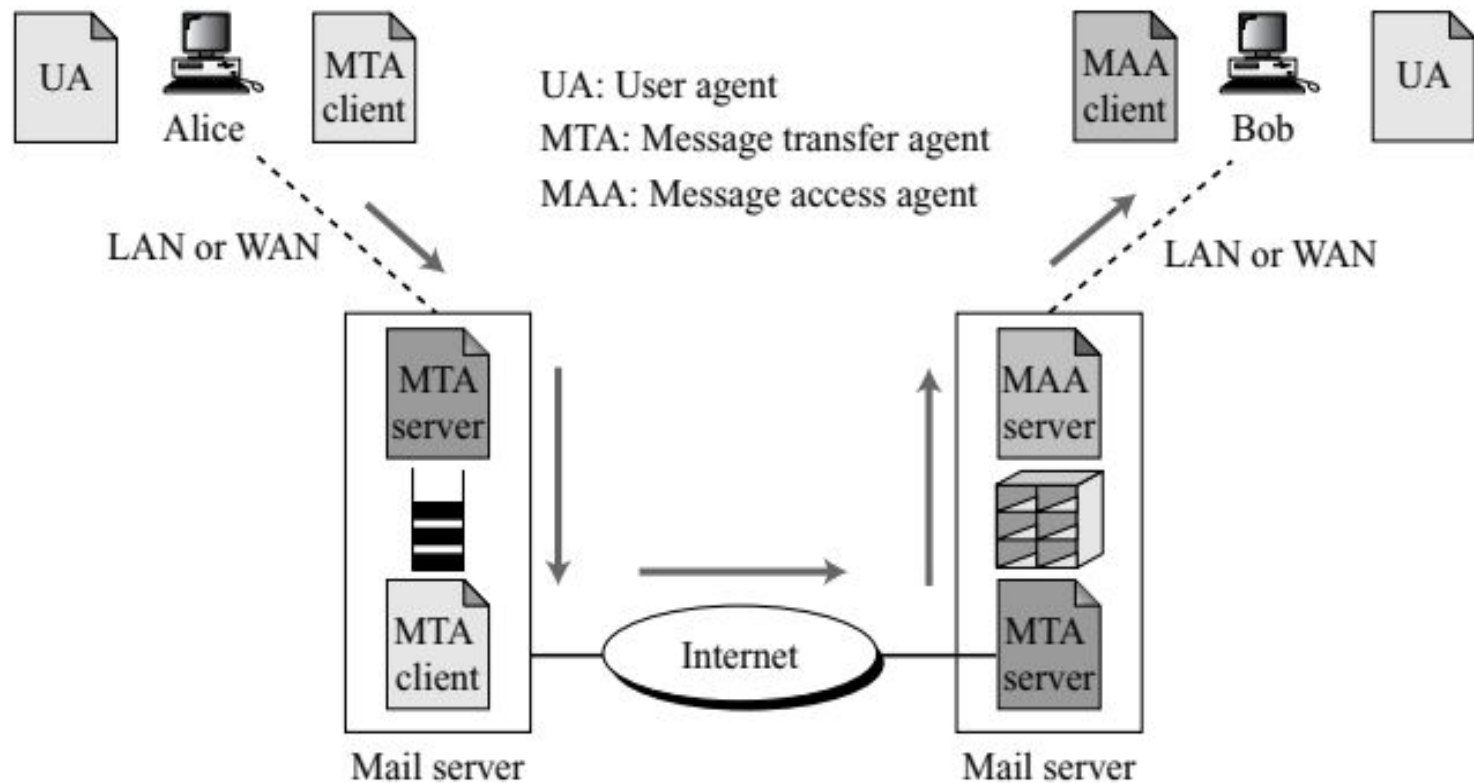
- ❑ **Pretty Good Privacy (PGP)**

- ❑ **Secure/Multipurpose Internet Mail Extension (S/MIME)**

Understanding each of these protocols requires the general understanding of the e-mail system.

# E-mail Architecture

---



---

There are several important points about the architecture of the e-mail system.

- ❑ The sending of an e-mail from Alice to Bob is a store-retrieve activity. Alice can send an e-mail today; Bob, being busy, may check his e-mail three days later. During this time, the e-mail is stored in Bob's mailbox until it is retrieved.
- ❑ The main communication between Alice and Bob is through two application programs: the MTA client at Alice's computer and the MAA client at Bob's computer.
- ❑ The MTA client program is a push program; the client pushes the message when Alice needs to send it. The MAA client program is a pull program; the client pulls the messages when Bob is ready to retrieve his e-mail.
- ❑ Alice and Bob cannot directly communicate using an MTA client at the sender site and an MTA server at the receiver site. This requires that the MTA server be running all the time, because Bob does not know when a message will arrive. This is not practical, because Bob probably turns off his computer when he does not need it.

# E-mail Security

---

- ❑ Sending an e-mail is a one-time activity.
- ❑ The nature of this activity is different from those we discussed in last two chapters.
- ❑ In IPSec or SSL, we assume that the two parties create a session between themselves and exchange data in both directions.
- ❑ In e-mail, there is no session. Alice and Bob cannot create a session.
- ❑ Alice sends a message to Bob; sometime later, Bob reads the message and may or may not send a reply.
- ❑ We discuss the security of a unidirectional message because what Alice sends to Bob is totally independent from what Bob sends to Alice.

# Cryptographic Algorithms and Keys

---

- ❑ In e-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.
- ❑ In e-mail security, the encryption/decryption is done using a symmetric-key algorithm, but the secret key to decrypt the message is encrypted with the public key of the receiver and is sent with the message.

# Pretty Good Privacy

---

- Philip R. Zimmerman is the creator of PGP.
- PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications.

# Why Is PGP Popular?

---

- ❑ It is available free on a variety of platforms.
- ❑ Based on well known algorithms.
- ❑ Wide range of applicability
- ❑ Not developed or controlled by governmental or standards organizations



# Operational Description

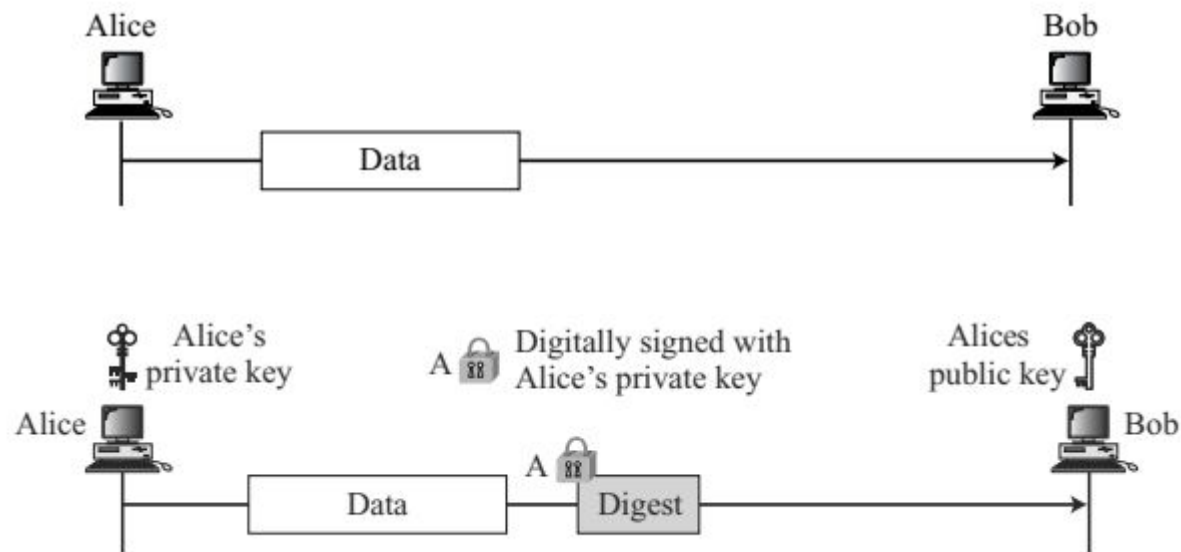
---

Consist of five services:

- Authentication
- Confidentiality
- Compression
- E-mail compatibility
- Segmentation

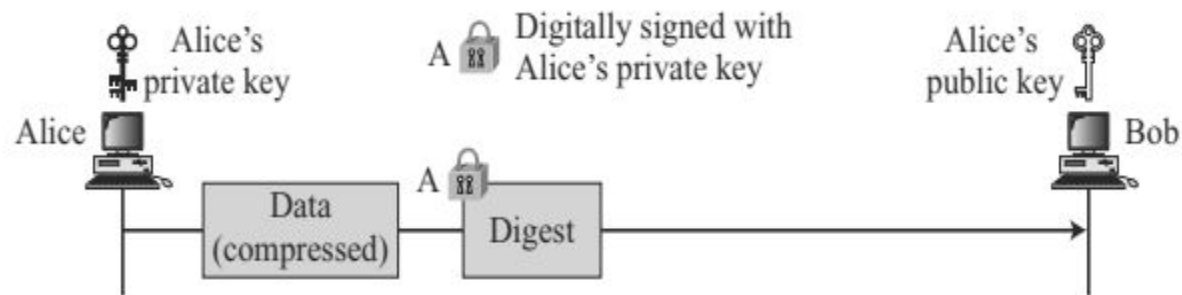
---

Let us first discuss the general idea of PGP, moving from a simple scenario to a complex one.

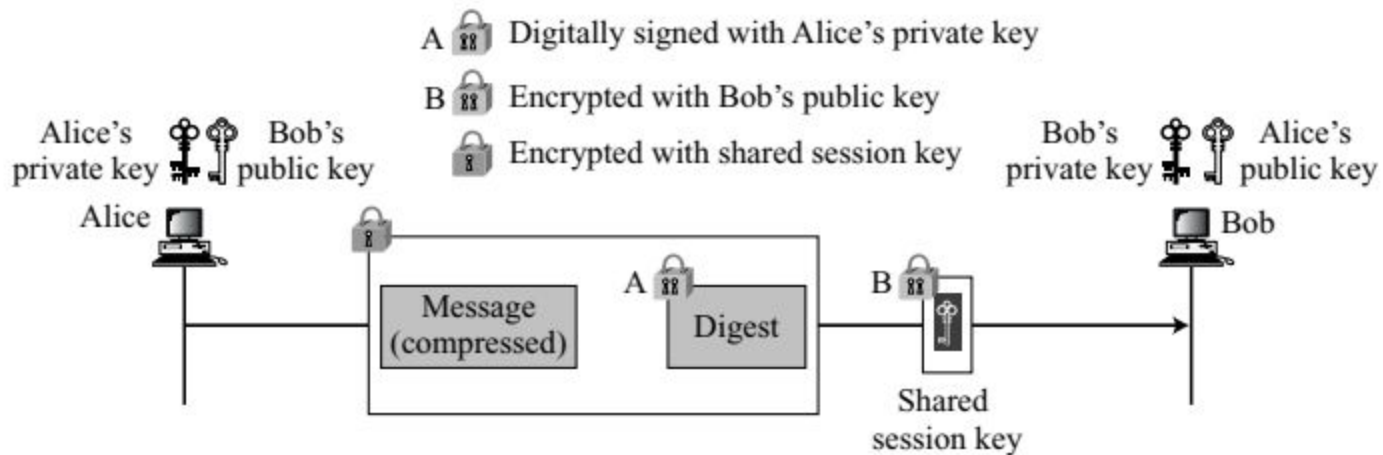


# Compression

---



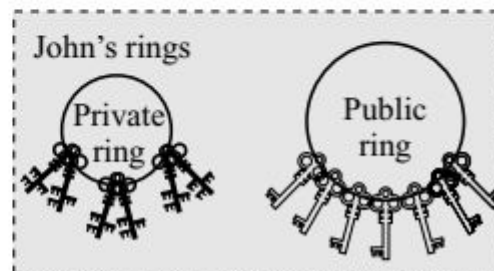
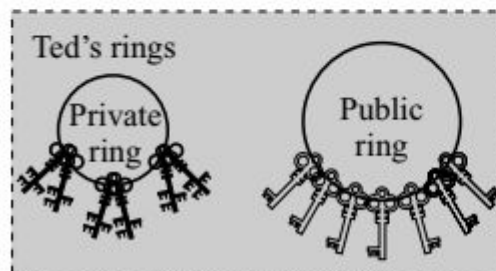
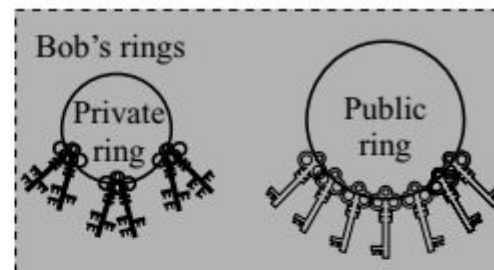
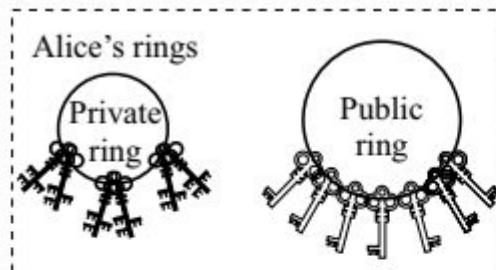
# Confidentiality with One-Time Session Key



# Key Rings

---

- ❑ Alice may need to send messages to many people; she needs key rings.
- ❑ In this case, Alice needs a ring of public keys, with a key belonging to each person with whom Alice needs to correspond (send or receive messages).
- ❑ In addition, the PGP designers specified a ring of private/public keys.
- ❑ One reason is that Alice may wish to change her pair of keys from time to time.
- ❑ Another reason is that Alice may need to correspond with different groups of people (friends, colleagues, and so on).
- ❑ Alice may wish to use a different key pair for each group.
- ❑ Therefore, each user needs to have two sets of rings: a ring of private/public keys and a ring of public keys of other people.



# Example

---

Alice, has several pairs of private/public keys belonging to her and public keys belonging to other people. Note that everyone can have more than one public key. Two cases may arise.

1. Alice needs to send a message to another person in the community.
  - She uses her private key to sign the digest.
  - She uses the receiver's public key to encrypt a newly created session key.
  - She encrypts the message and signed digest with the session key created.
2. Alice receives a message from another person in the community.
  - She uses her private key to decrypt the session key.
  - She uses the session key to decrypt the message and digest.
  - She uses her public key to verify the digest.

# PGP Algorithms

---

## Public-Key Algorithms

**Table 16.1** *Public-key algorithms*

<i>ID</i>	<i>Description</i>
1	RSA (encryption or signing)
2	RSA (for encryption only)
3	RSA (for signing only)
16	ElGamal (encryption only)
17	DSS
18	Reserved for elliptic curve
19	Reserved for ECDSA
20	ElGamal (for encryption or signing)
21	Reserved for Diffie-Hellman
100–110	Private algorithms



---

## Symmetric-Key Algorithms

**Table 16.2** *Symmetric-key algorithms*

<i>ID</i>	<i>Description</i>
0	No Encryption
1	IDEA
2	Triple DES
3	CAST-128
4	Blowfish
5	SAFER-SK128
6	Reserved for DES/SK
7	Reserved for AES-128
8	Reserved for AES-192
9	Reserved for AES-256
100–110	Private algorithms

---

**Table 16.3** *Hash Algorithms*

<i>ID</i>	<i>Description</i>
1	MD5
2	SHA-1
3	RIPE-MD/160
4	Reserved for double-width SHA
5	MD2
6	TIGER/192
7	Reserved for HAVAL
100–110	Private algorithms

---

## Compression Algorithms

**Table 16.4** *Compression methods*

<i>ID</i>	<i>Description</i>
0	Uncompressed
1	ZIP
2	ZLIP
100–110	Private methods

# PGP Certificates

---

In X.509, there is a single path from the fully trusted authority to any certificate.

In PGP, there can be multiple paths from fully or partially trusted authorities to any subject.

The entire operation of PGP is based on introducer trust, the certificate trust, and the legitimacy of the public keys.

# Introducer Trust Levels

---

- ❑ With the lack of a central authority, it is obvious that the ring cannot be very large if every user in the PGP ring of users has to fully trust everyone else.
- ❑ PGP allows different levels of trust.
- ❑ The number of levels is mostly implementation dependent, but for simplicity, let us assign three levels of trust to any introducer: none, partial, and full.
- ❑ The introducer trust level specifies the trust levels issued by the introducer for other people in the ring.
- ❑ For example, Alice may fully trust Bob, partially trust Anne, and not trust John at all.
- ❑ There is no mechanism in PGP to determine how to make a decision about the trustworthiness of the introducer; it is up to the user to make this decision.

# Certificate Trust Levels

---

- ❑ When Alice receives a certificate from an introducer, she stores the certificate under the name of the subject (certified entity).
- ❑ She assigns a level of trust to this certificate.
- ❑ The certificate trust level is normally the same as the introducer trust level that issued the certificate.
- ❑ Assume that Alice fully trusts Bob, partially trusts Anne and Janette, and has no trust in John.

# Scenario

---

- ❑ Bob issues two certificates, one for Linda (with public key K1) and one for Lesley (with public key K2). Alice stores the public key and certificate for Linda under Linda's name and assigns a full level of trust to this certificate. Alice also stores the certificate and public key for Lesley under Lesley's name and assigns a full level of trust to this certificate.
- ❑ Anne issues a certificate for John (with public key K3). Alice stores this certificate and public key under John's name, but assigns a partial level for this certificate.
- ❑ Janette issues two certificates, one for John (with public key K3) and one for Lee (with public key K4). Alice stores John's certificate under his name and Lee's certificate under his name, each with a partial level of trust. Note that John now has two certificates, one from Anne and one from Janette, each with a partial level of trust.
- ❑ John issues a certificate for Liz. Alice can discard or keep this certificate with a signature trust of none.

# Key Legitimacy

---

The purpose of using introducer and certificate trusts is to determine the legitimacy of a public key. Alice needs to know how legitimate the public keys of Bob, John, Liz, Anne, and so on are. PGP defines a very clear procedure for determining key legitimacy. The level of the key legitimacy for a user is the weighted trust levels of that user



---

For example, suppose we assign the following weights to certificate trust levels:

1. A weight of 0 to a nontrusted certificate
2. A weight of  $1/2$  to a certificate with partial trust
3. A weight of 1 to a certificate with full trust

- 
- ❑ Then to fully trust an entity, Alice needs one fully trusted certificate or two partially trusted certificates for that entity.
  - ❑ For example, Alice can use John's public key in the previous scenario because both Anne and Janette have issued a certificate for John, each with a certificate trust level of 1/2.
  - ❑ Note that the legitimacy of a public key belonging to an entity does not have anything to do with the trust level of that person.
  - ❑ Although Bob can use John's public key to send a message to him, Alice cannot accept any certificate issued by John because, for Alice, John has a trust level of none.

# Starting the Ring

---

- ❑ Alice can physically obtain Bob's public key. For example, Alice and Bob can meet personally and exchange a public key written on a piece of paper or to a disk.
- ❑ If Bob's voice is recognizable to Alice, Alice can call him and obtain his public key on the phone.
- ❑ A better solution proposed by PGP is for Bob to send his public key to Alice by e-mail. Both Alice and Bob make a 16-byte MD5 (or 20-byte SHA-1) digest from the key. The digest is normally displayed as eight groups of 4 digits (or ten groups of 4 digits) in hexadecimal and is called a fingerprint. Alice can then call Bob and verify the fingerprint on the phone.
- ❑ In PGP, nothing prevents Alice from getting Bob's public key from a CA in a separate procedure. She can then insert the public key in the public key ring.

# Key Ring Tables

---

## Private Key Ring Table



User ID	Key ID	Public key	Encrypted private key	Timestamp
⋮	⋮	⋮	⋮	⋮

- 
- ❑ User ID: The user ID is usually the e-mail address of the user.
  - ❑ Key ID: This column uniquely defines a public key among the user's public keys. In PGP, the key ID for each pair is the first (least significant) 64 bits of the public key. In other words, the key ID is calculated as  $(\text{key} \bmod 264)$ .
  - ❑ Public Key: This column just lists the public key belonging to a particular private key/public key pair.
  - ❑ Encrypted Private Key: This column shows the encrypted value of the private key in the private key/public key pair. Although Alice is the only person accessing her private ring, PGP saves only the encrypted version of the private key. We will see later how the private key is encrypted and decrypted.
  - ❑ Timestamp: This column holds the date and time of the key pair creation. It helps the user decide when to purge old pairs and when to create new ones.

---


Example-1: Let us show a private key ring table for Alice. We assume that Alice has only two user IDs, `alice@some.com` and `alice@anet.net`. We also assume that Alice has two sets of private/public keys, one for each user ID. Table 16.5 shows the private key ring table for Alice.

**Table 16.5**     *Private key ring table for Example 1*

<i>User ID</i>	<i>Key ID</i>	<i>Public Key</i>	<i>Encrypted Private Key</i>	<i>Timestamp</i>
<code>alice@anet.net</code>	AB13...45	AB13...45...59	<b>32452398...23</b>	031505-16:23
<code>alice@some.com</code>	FA23...12	FA23...12...22	<b>564A4923...23</b>	031504-08:11

# Public Key Ring Table

---

 Public ring	User ID	Key ID	Public key	Producer trust	Certificate(s)	Certificate trust(s)	Key Legitimacy	Timestamp
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

- 
- ❑ User ID. As in the private key ring table, the user ID is usually the e-mail address of the entity.
  - ❑ Key ID. As in the private key ring table, the key ID is the first (least significant) 64 bits of the public key.
  - ❑ Public Key. This is the public key of the entity.
  - ❑ Producer Trust. This column defines the producer level of trust. In most implementations, it can only be of one of three values: none, partial, or full.
  - ❑ Certificate(s). This column holds the certificate or certificates signed by other entities for this entity. A user ID may have more than one certificate.



- 
- ❑ Certificate Trust(s). This column represents the certificate trust or trusts. If Anne sends a certificate for John, PGP searches the row entry for Anne, finds the value of the producer trust for Anne, copies that value, and inserts it in the certificate trust field in the entry for John.
  - ❑ Key Legitimacy. This value is calculated by PGP based on the value of the certificate trust and the predefined weight for each certificate trust.
  - ❑ Timestamp. This column holds the date and time of the column creation.

---

Example 2: A series of steps will show how a public key ring table is formed for Alice.

1. Start with one row, Alice herself, as shown in Table 16.6. Use N (none), P (partial), and F (full) for the levels of trust. For simplicity, also assume that everyone (including Alice) has only one user ID.

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....

---

2. Now Alice adds Bob to the table. Alice fully trusts Bob, but to obtain his public key, she asks Bob to send the public key by e-mail as well as his fingerprint. Alice then calls Bob to check the fingerprint.

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....

---

3. Now Alice adds Ted to the table. Ted is fully trusted. However, for this particular user, Alice does not have to call Ted. Instead, Bob, who knows Ted's public key, sends Alice a certificate that includes Ted's public key

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....
Ted...	48...	48.....	F	Bob's	F	F	.....

---

4. Now Alice adds Anne to the list. Alice partially trusts Anne, but Bob, who is fully trusted, sends a certificate for Anne.

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....
Ted...	48...	48.....	F	Bob's	F	F	.....
Anne...	71...	71.....	P	Bob's	F	F	.....

---

5. Now Anne introduces John, who is not trusted by Alice

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. Trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....
Ted...	48...	48.....	F	Bob's	F	F	.....
Anne...	71...	71.....	P	Bob's	F	F	.....
John...	31...	31.....	N	Anne's	P	P	.....

---

6. Now Janette, who is unknown to Alice, sends a certificate for Lee. Alice totally ignores this certificate because she does not know Janette.

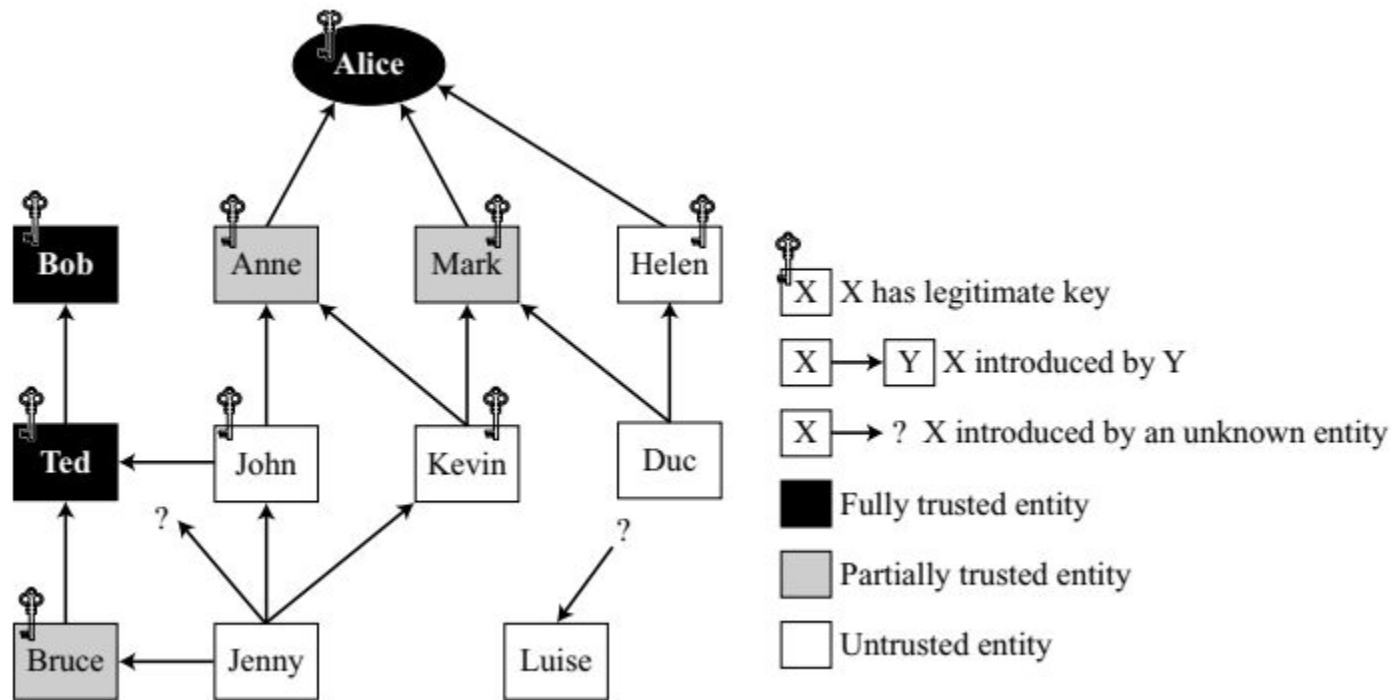
7. Now Ted sends a certificate for John (John, who is trusted by Ted, has probably asked Ted to send this certificate). Alice looks at the table and finds John's user ID with the corresponding key ID and public key. Alice does not add another row to the table; she just modifies the table

---

<i>User ID</i>	<i>Key ID</i>	<i>Public key</i>	<i>Prod. trust</i>	<i>Certificate</i>	<i>Cert. trust</i>	<i>Key legit.</i>	<i>Time-stamp</i>
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....
Ted...	48...	48.....	F	Bob's	F	F	.....
Anne...	71...	71.....	P	Bob's	F	F	.....
John...	31...	31.....	N	Anne's Ted's	P F	F	.....



# Trust Model in PGP



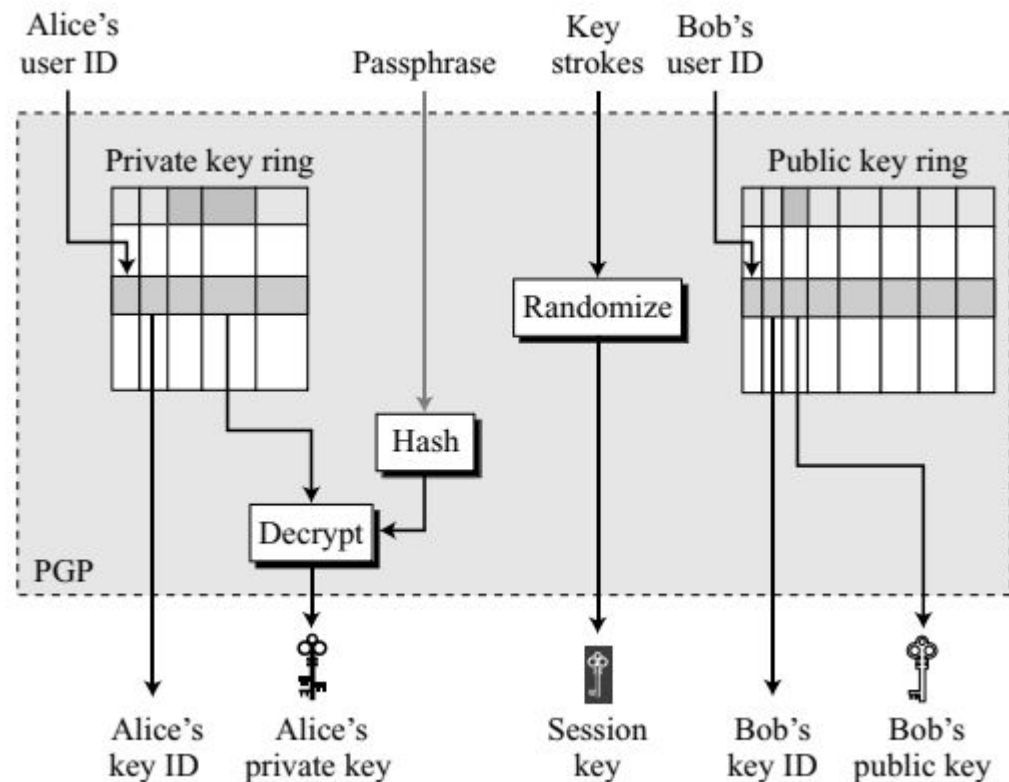
# Key Revocation

---

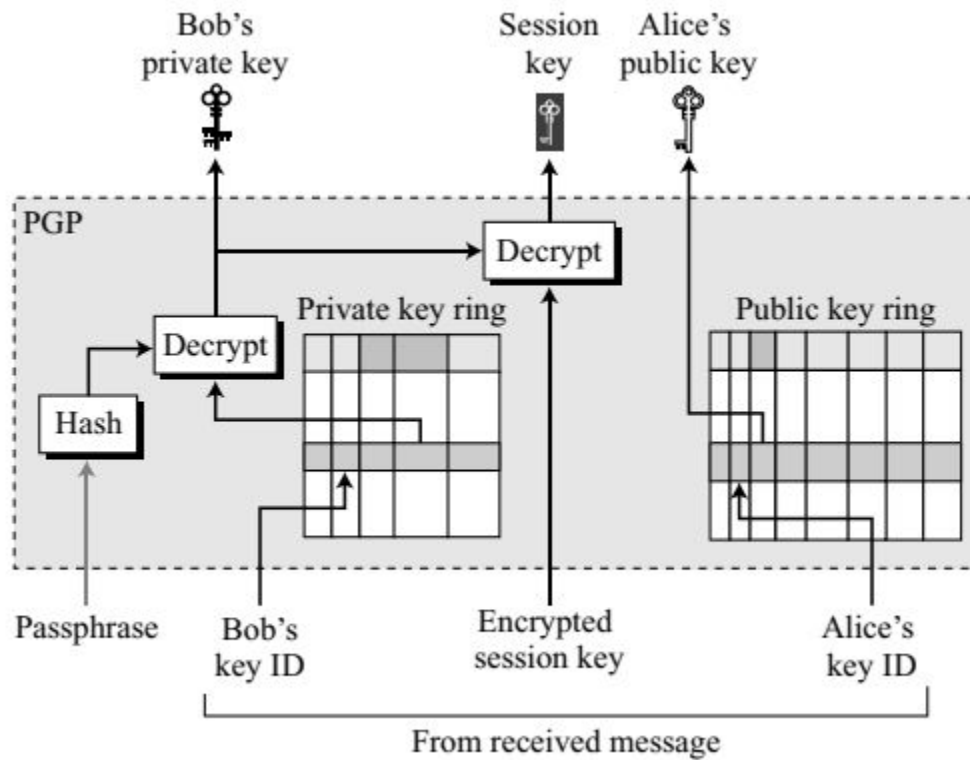
- ❑ It may become necessary for an entity to revoke his or her public key from the ring.
- ❑ This may happen if the owner of the key feels that the key is compromised (stolen, for example) or just too old to be safe.
- ❑ To revoke a key, the owner can send a revocation certificate signed by herself.
- ❑ The revocation certificate must be signed by the old key and disseminated to all the people in the ring that use that public key.

# Extracting Information from Rings

Sender Site

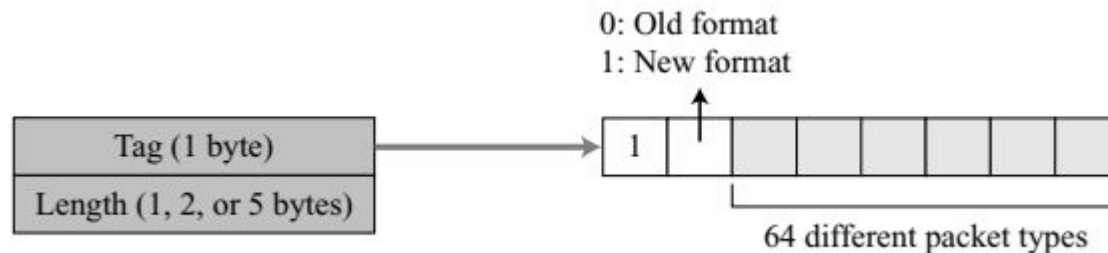


## Receiver Site



# PGP Packets

---



- ❑ Tag. The recent format for this field defines a tag as an 8-bit flag; the first bit (most significant) is always 1. The second bit is 1 if we are using the latest version. The remaining six bits can define up to 64 different packet types, as shown in Table 16.12.
- ❑ Length. The length field defines the length of the entire packet in bytes. The size of this field is variable; it can be 1, 2, or 5 bytes.

---

**Table 16.12** *Some commonly used packet types*

<i>Value</i>	<i>Packet type</i>
1	Session key packet encrypted using a public key
2	Signature packet
5	Private-key packet
6	Public-key packet
8	Compressed data packet
9	Data packet encrypted with a secret key
11	Literal data packet
13	User ID packet

- 
- ❑ If the value of the byte after the tag field is less than 192, the length field is only one byte. The length of the body (packet minus header) is calculated as:

body length = first byte

- ❑ If the value of the byte after the tag field is between 192 and 223 (inclusive), the length field is two bytes. The length of the body can be calculated as:

body length = (first byte - 192) << 8 + second byte + 192

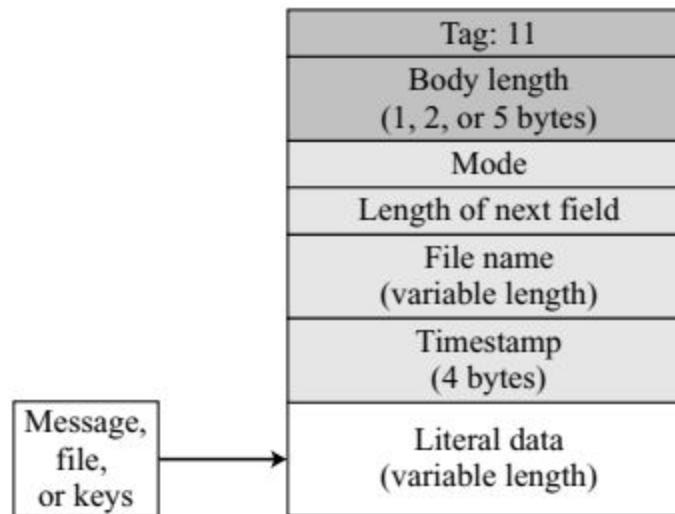
- ❑ If the value of the byte after the tag field is between 224 and 254 (inclusive), the length field is one byte. This type of length field defines only the length of part of the body (partial body length). The partial body length can be calculated as:  
partial body length = 1 << (first byte & 0x1F)

- 
- If the value of the byte after the tag field is 255, the length field consists of five bytes. The length of the body is calculated as  
Body length = second byte << 24 | third byte << 16 | fourth byte << 8 | fifth byte



# Literal Data Packet

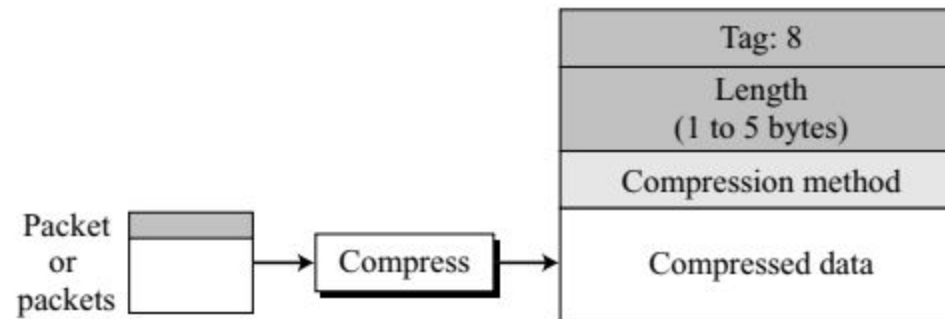
---



- 
- ❑ Mode. This one-byte field defines how data is written to the packet. The value of this field can be “b” for binary, “t” for text, or any other locally defined value.
  - ❑ Length of next field. This one-byte field defines the length of the next field (file name field).
  - ❑ File name. This variable-length field defines the name of the file or message as an ASCII string.
  - ❑ Timestamp. This four-byte field defines the time of creation or last modification of the message. The value can be 0, which means that the user chooses not to specify a time.
  - ❑ Literal data. This variable-length field carries the actual data (file or message) in text or binary (depending on the value of the mode field).

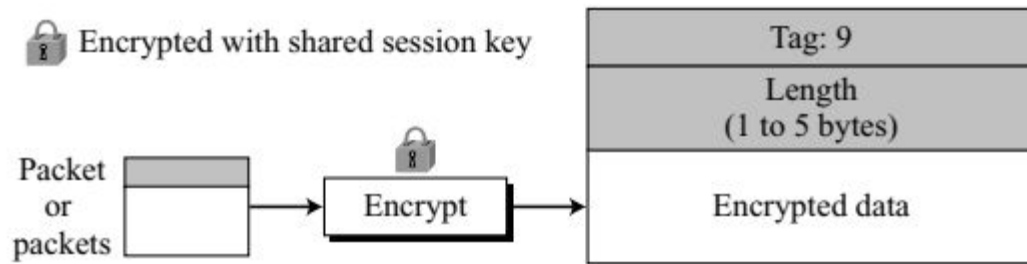
# Compressed Data Packet

---

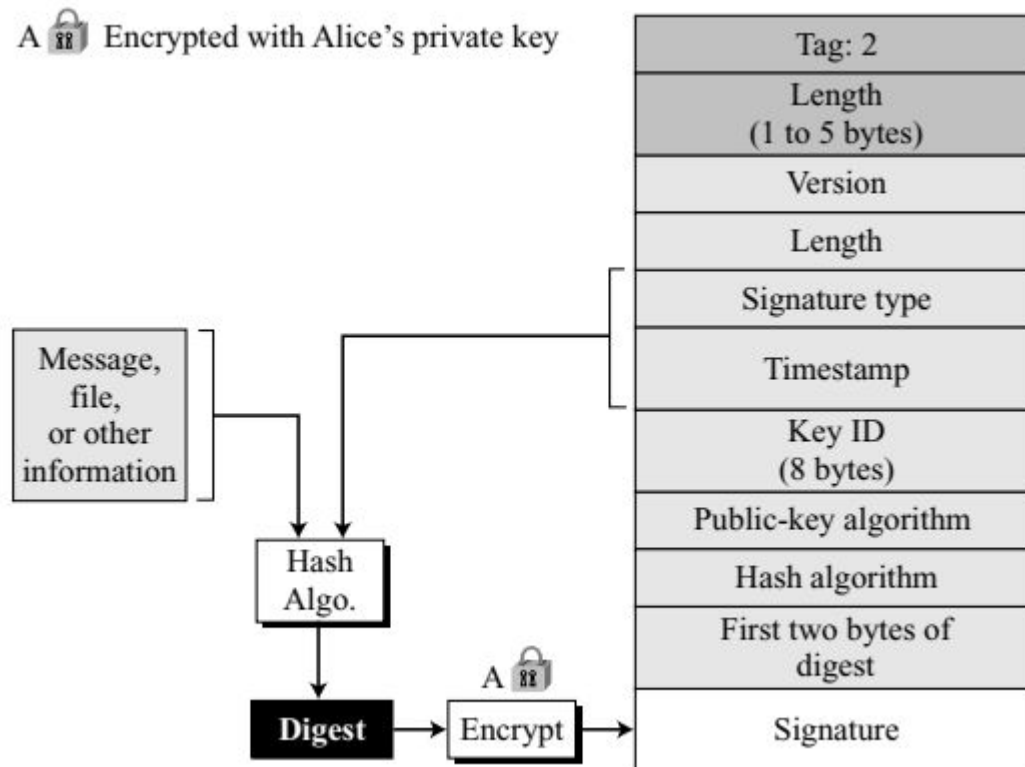


# Data Packet Encrypted with Secret Key

---



# Signature Packet



---

❑ Version. This one-byte field defines the PGP version that is being used.

❑ Length. This field was originally designed to show the length of the next two fields, but because the size of these fields is now fixed, the value of this field is 5.

❑ Signature type. This one-byte field defines the purpose of the signature, the document it signs.

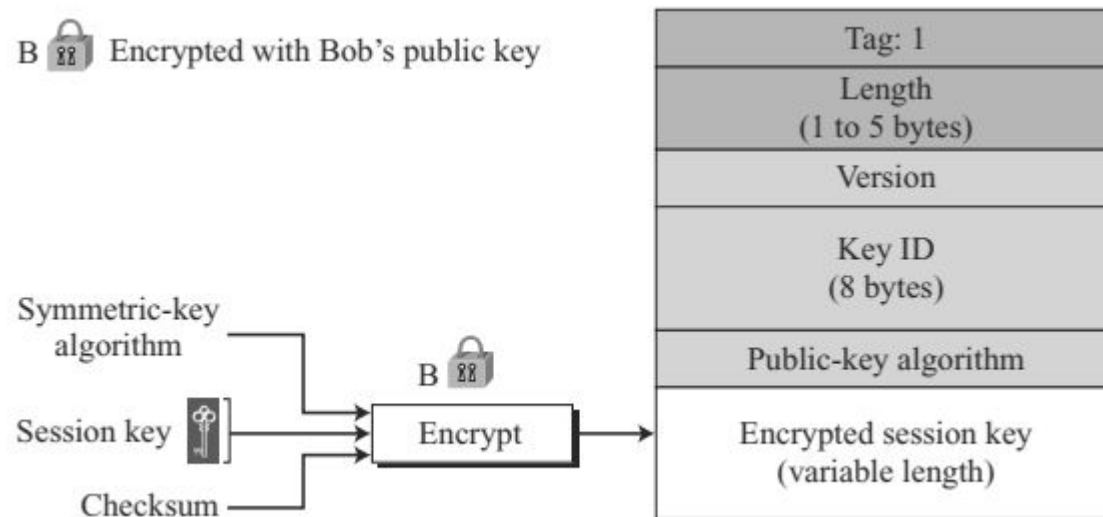
Timestamp. This four-byte field defines the time the signature was calculated.

❑ Key ID. This eight-byte field defines the public-key ID of the signer. It indicates to the verifier which signer public key should be used to decrypt the digest.

❑ Public-key algorithm. This one-byte field gives the code for the public-key algorithm used to encrypt the digest. The verifier uses the same algorithm to decrypt the digest.

- 
- ❑ Hash algorithm. This one-byte field gives the code for the hash algorithm used to create the digest.
  - ❑ First two bytes of message digest. These two bytes are used as a kind of checksum. They ensure that the receiver is using the right key ID to decrypt the digest.
  - ❑ Signature. This variable-length field is the signature. It is the encrypted digest signed by the sender.

# Session-Key Packet Encrypted with Public Key





# Public-Key Packet

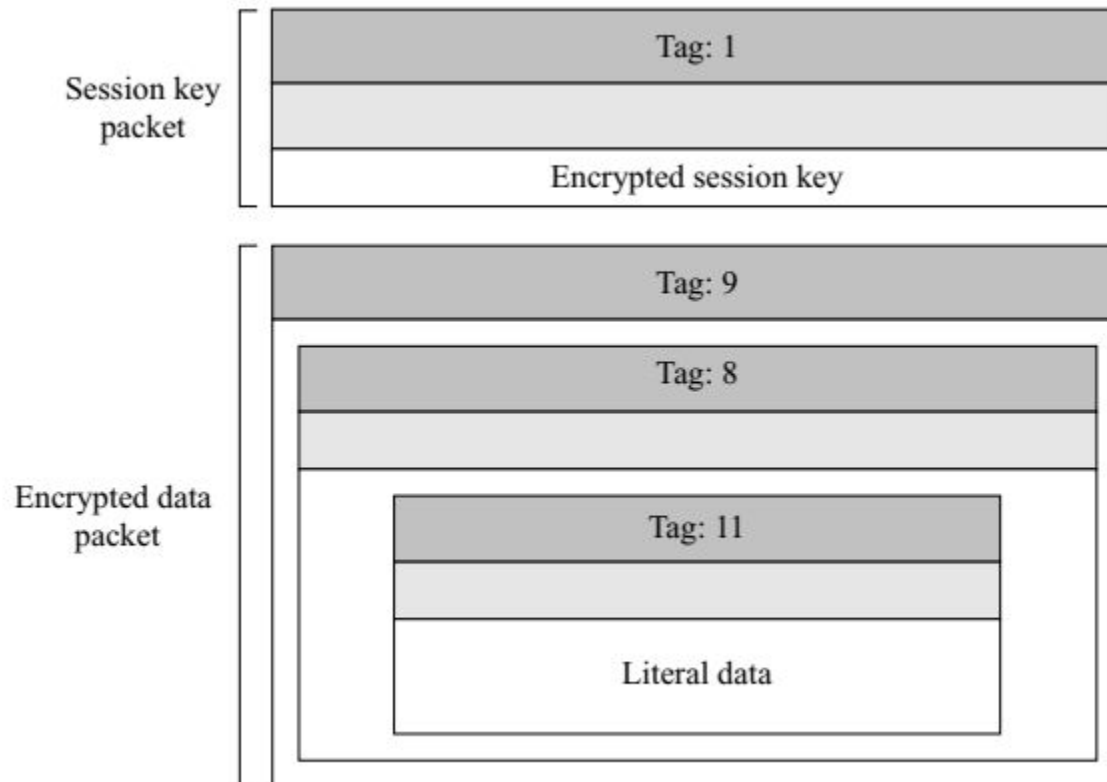
---

Tag: 6
Length (1 to 5 bytes)
Version
Key ID (8 bytes)
Public-key algorithm
Public key

# PGP Messages

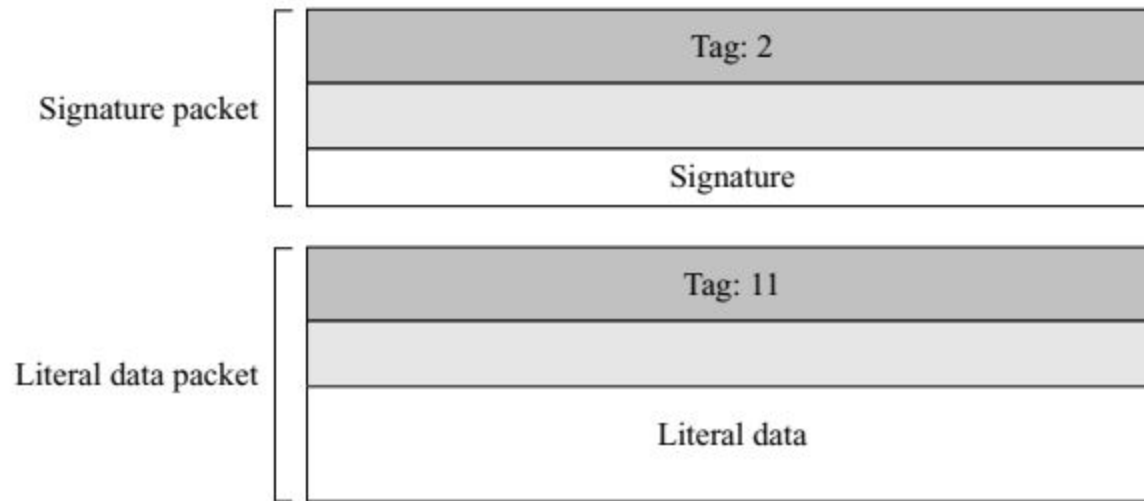
---

## Encrypted Message



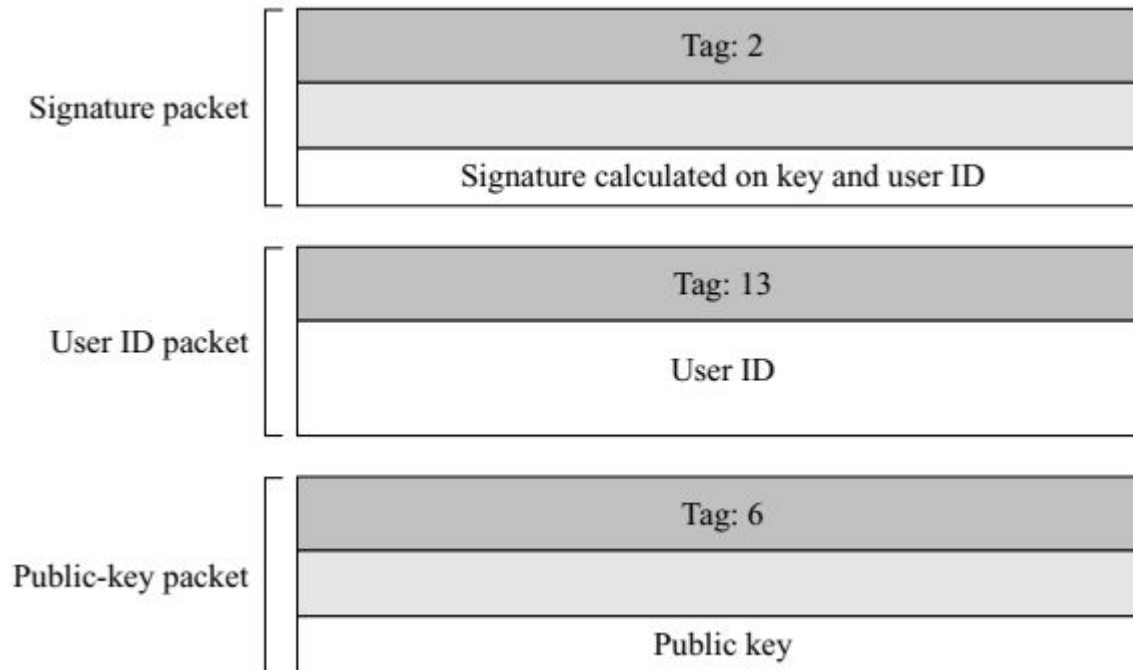
# Signed Message

---



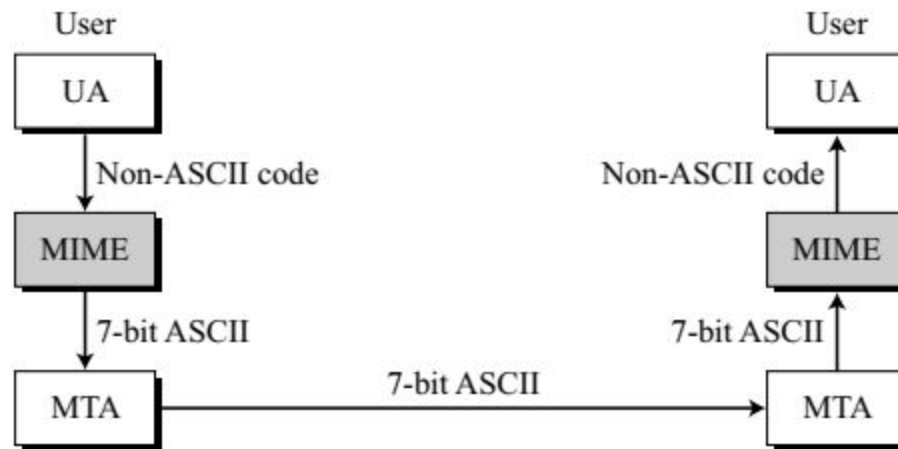
# Certificate Message

---



# MIME

---



# MIME headers

---

Added to the original e-mail header section to define the transformation parameters:

1. MIME-Version
2. Content-Type
3. Content-Transfer-Encoding
4. Content-Id
5. Content-Description

---

E-mail header
MIME-Version: 1.1 Content-Type: type/subtype Content-Transfer-Encoding: encoding type Content-Id: message id Content-Description: textual explanation of nontextual contents
E-mail body

MIME headers

---

MIME-Version: 1.1

Content-Type: <type / subtype; parameters>

Content-Transfer-Encoding: <type>

Content-Id: id=<content-id>

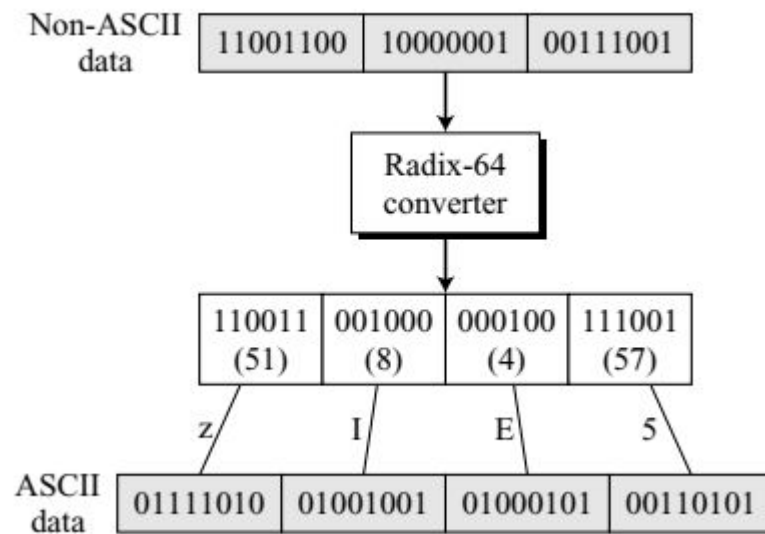
Content-Description: <description>



<i>Type</i>	<i>Subtype</i>	<i>Description</i>
	Plain	Unformatted.
	HTML	HTML format.
Multipart	Mixed	Body contains ordered parts of different data types.
	Parallel	Same as above, but no order.
	Digest	Similar to Mixed, but the default is message/RFC822.
	Alternative	Parts are different versions of the same message.
Message	RFC822	Body is an encapsulated message.
	Partial	Body is a fragment of a bigger message.
	External-Body	Body is a reference to another message.
Image	JPEG	Image is in JPEG format.
	GIF	Image is in GIF format.
Video	MPEG	Video is in MPEG format.
Audio	Basic	Single channel encoding of voice at 8 KHz.
Application	PostScript	Adobe PostScript.
	Octet-stream	General binary data (eight-bit bytes).

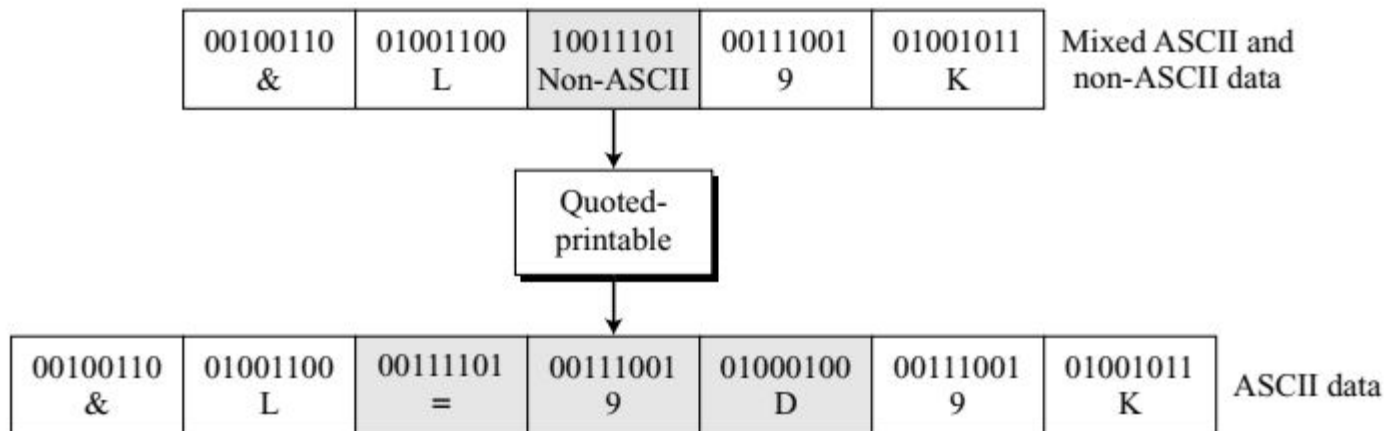
---

<i>Type</i>	<i>Description</i>
7bit	NVT ASCII characters and short lines.
8bit	Non-ASCII characters and short lines.
Binary	Non-ASCII characters with unlimited-length lines.
Radix-64	6-bit blocks of data are encoded into 8-bit ASCII characters using Radix-64 conversion.
Quoted-printable	Non-ASCII characters are encoded as an equal sign followed by an ASCII code.



**Table 16.16** *Radix-64 encoding table*

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



# S/MIME

---


- ❑ S/MIME adds some new content types to include security services to the MIME.
- ❑ All of these new types include the parameter “application/pkcs7-mime,” in which “pkcs” defines “Public Key Cryptography Specification.”


# Signed-Data Content Type

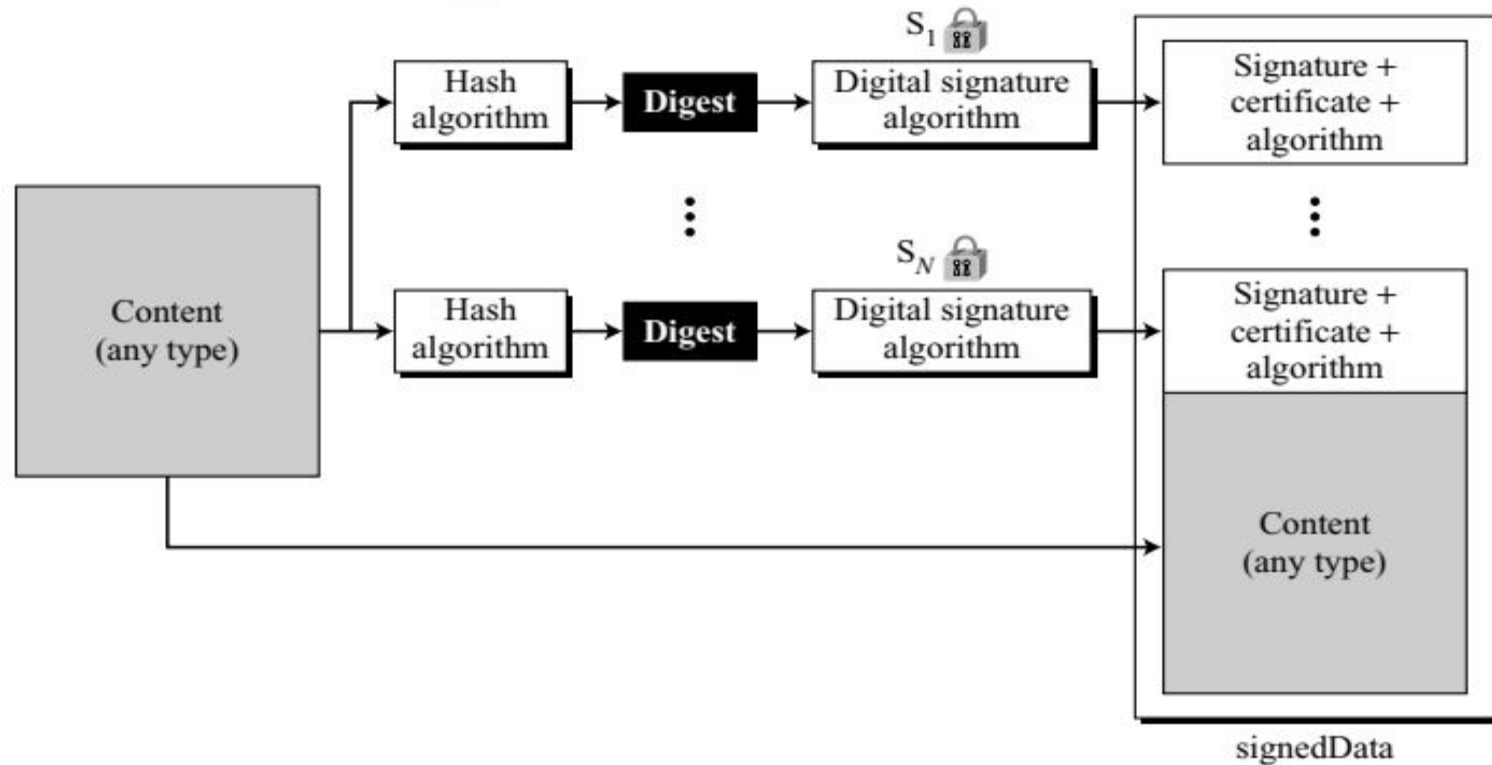
---

This type provides only integrity of data. It contains any type and zero or more signature values. The encoded result is an object called `signedData`. The following are the steps in the process:

1. For each signer, a message digest is created from the content using the specific hash algorithm chosen by that signer.
2. Each message digest is signed with the private key of the signer.
3. The content, signature values, certificates, and algorithms are then collected to create the `signedData` object.

$S_1$   Signed with private key of signer 1

$S_N$   Signed with private key of signer  $N$







# Enveloped-Data Content Type


---

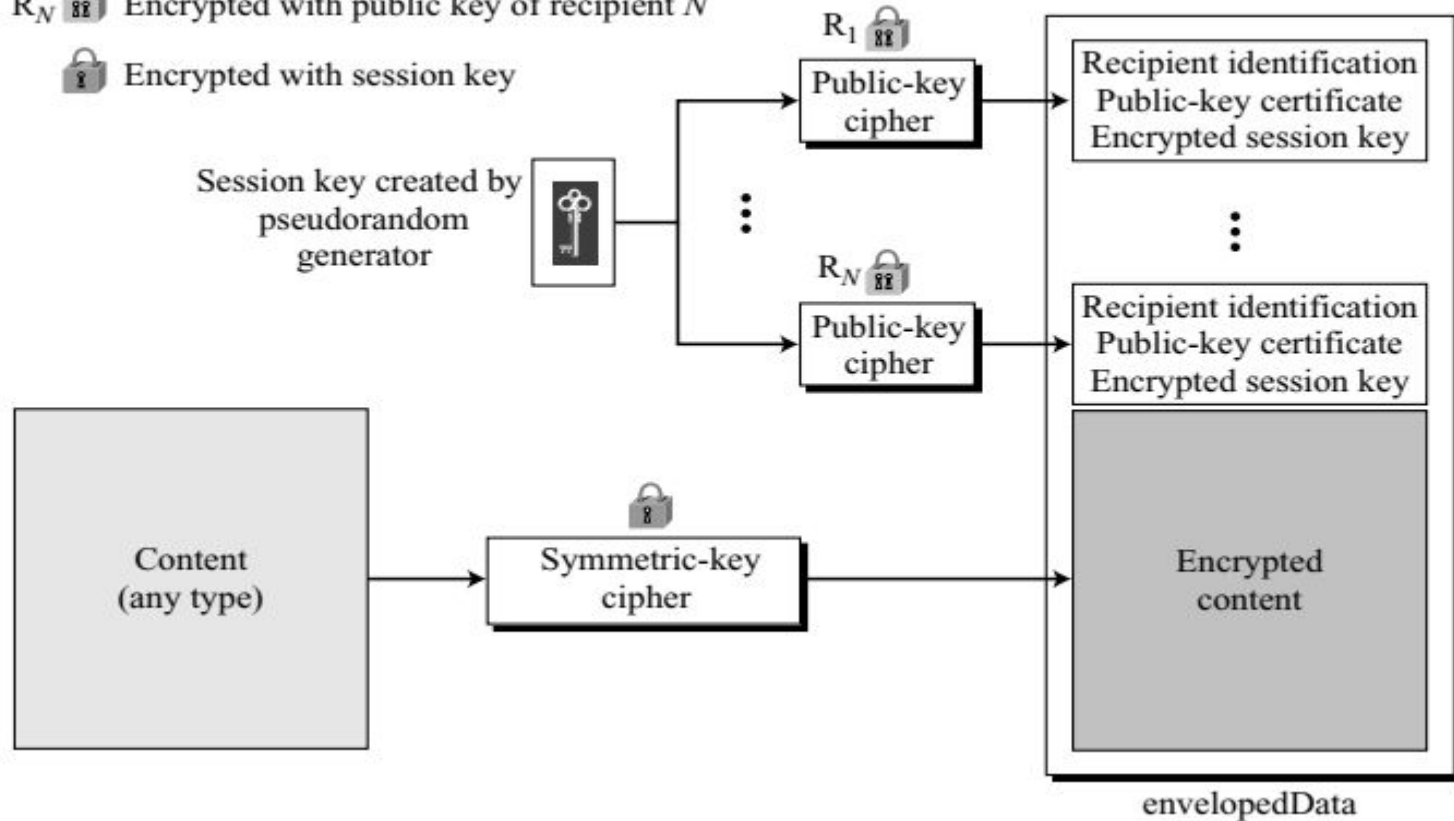
This type is used to provide privacy for the message. It contains any type and zero or more encrypted keys and certificates. The encoded result is an object called `envelopedData`.

1. A pseudorandom session key is created for the symmetric-key algorithms to be used.
2. For each recipient, a copy of the session key is encrypted with the public key of each recipient.
3. The content is encrypted using the defined algorithm and created session key.
4. The encrypted contents, encrypted session keys, algorithm used, and certificates are encoded using Radix-64.

$R_1$   Encrypted with public key of recipient 1

$R_N$   Encrypted with public key of recipient  $N$

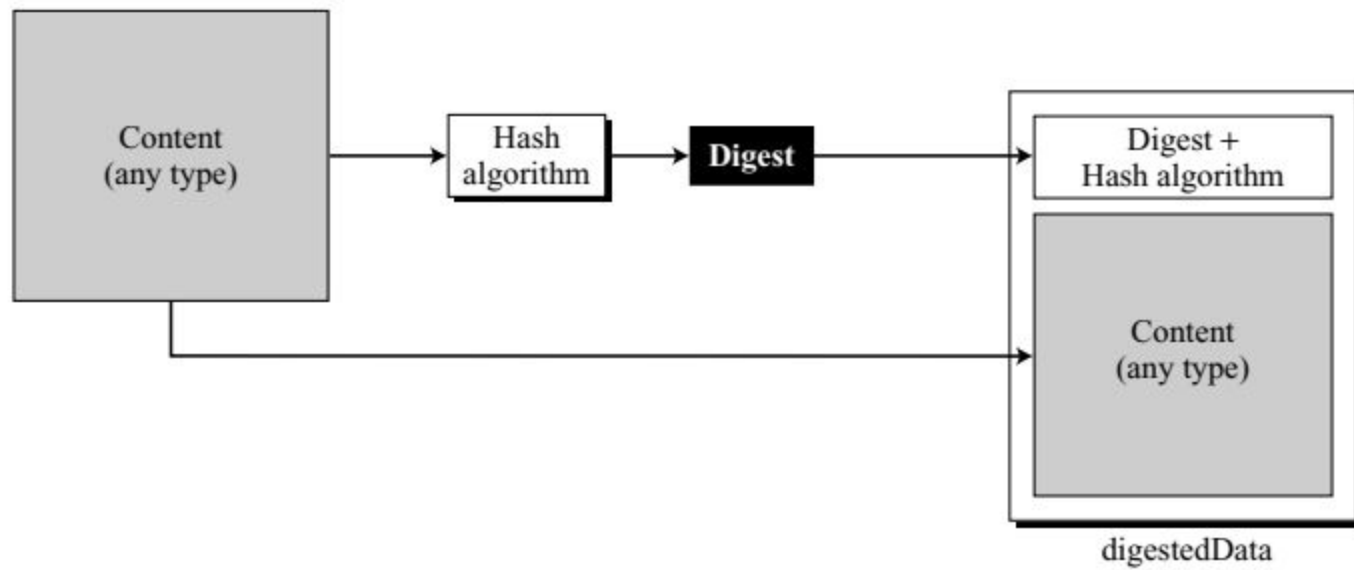
 Encrypted with session key



# Digested-Data Content Type

---

- ❑ This type is used to provide integrity for the message.
- ❑ The result is normally used as the content for the enveloped-data content type. The encoded result is an object called `digestedData`.
- ❑ Figure 16.29 shows the process of creating an object of this type.





# Authenticated-Data Content Type

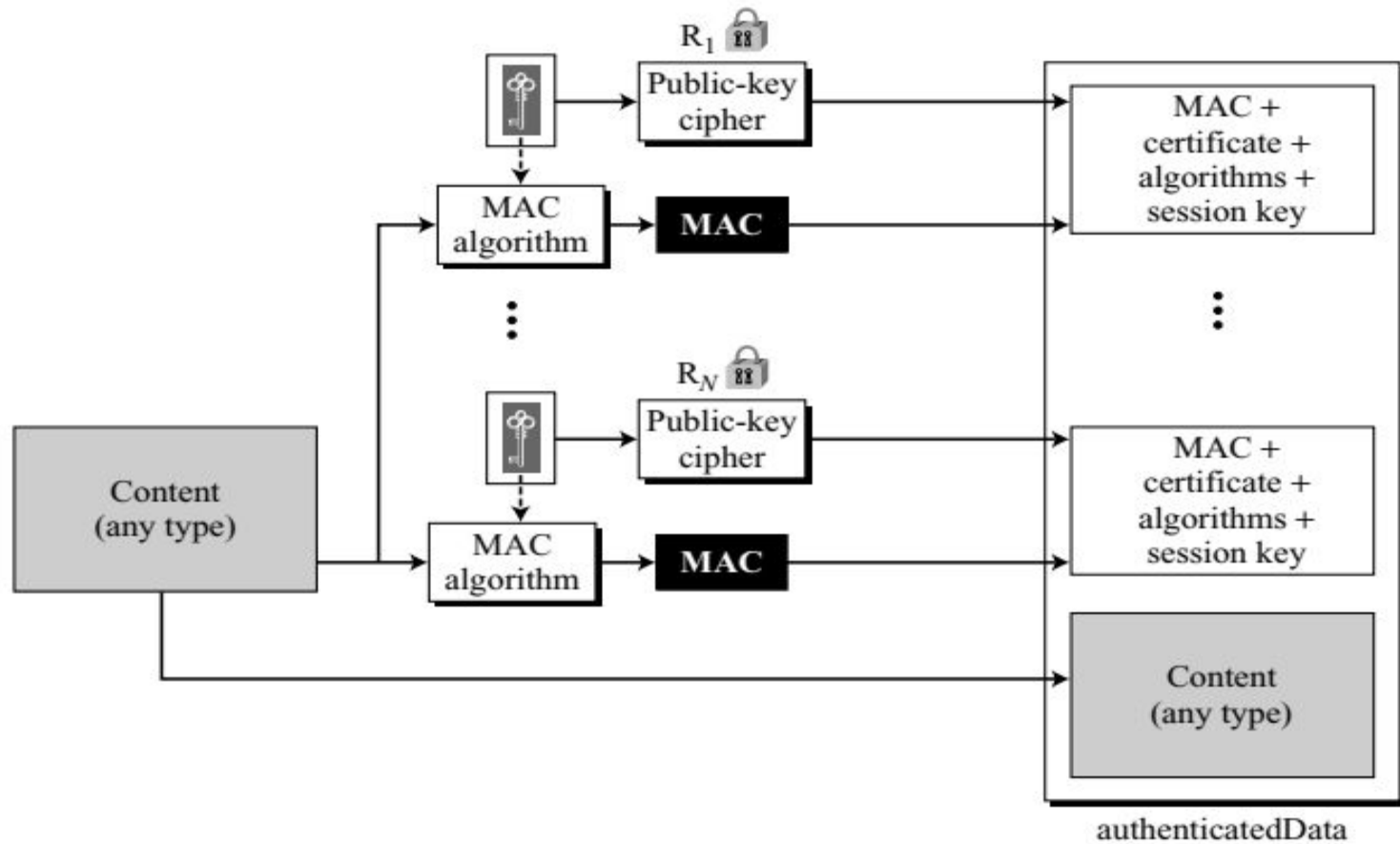
---

This type is used to provide authentication of the data. The object is called `authenticatedData`

1. Using a pseudorandom generator, a MAC key is generated for each recipient.
2. The MAC key is encrypted with the public key of the recipient.
3. A MAC is created for the content.
4. The content, MAC, algorithms, and other information are collected together to form the `authenticatedData` object.

$R_1$   Encrypted with public key of recipient 1

$R_N$   Encrypted with public key of recipient  $N$



---

## **Key Management**

The key management in S/MIME is a combination of key management used by X.509 and PGP. S/MIME uses public-key certificates signed by the certificate authorities defined by X.509. However, the user is responsible to maintain the web of trust to verify signatures as defined by PGP.

## **Cryptographic Algorithms**

S/MIME defines several cryptographic algorithms as shown in Table 16.17. The term “must” means an absolute requirement; the term “should” means recommendation.

# Cryptographic Algorithms

---

<i>Algorithm</i>	<i>Sender must support</i>	<i>Receiver must support</i>	<i>Sender should support</i>	<i>Receiver should support</i>
Content-encryption algorithm	Triple DES	Triple DES		1. AES 2. RC2/40
Session-key encryption algorithm	RSA	RSA	Diffie-Hellman	Diffie-Hellman
Hash algorithm	SHA-1	SHA-1		MD5
Digest-encryption algorithm	DSS	DSS	RSA	RSA
Message-authentication algorithm		HMAC with SHA-1		