d. 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 will see in the next 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

If e-mail is a one-time activity, how can the sender and receiver agree on a cryptographic algorithm to use for e-mail security? If there is no session and no handshaking to negotiate the algorithms for encryption/decryption and hashing, how can the receiver know which algorithm the sender has chosen for each purpose?

One solution is for the underlying protocol to select one algorithm for each cryptographic operation and to force Alice to use only those algorithms. This solution is very restrictive and limits the capabilities of the two parties.

A better solution is for the underlying protocol to define a set of algorithms for each operation that the user used in his/her system. Alice includes the name (or identifiers) of the algorithms she has used in the e-mail. For example, Alice can choose triple DES for encryption/decryption and MD5 for hashing. When Alice sends a message to Bob, she includes the corresponding identifiers for triple DES and MD5 in her message. Bob receives the message and extracts the identifiers first. He then knows which algorithm to use for decryption and which one for hashing.

In e-mail security, the sender of the message needs to include the name or identifiers of the algorithms used in the message.

Cryptographic Secrets

The same problem for the cryptographic algorithms applies to the cryptographic secrets (keys). If there is no negotiation, how can the two parties establish secrets between themselves? Alice and Bob could use asymmetric-key algorithms for authentication and encryption, which do not require the establishment of a symmetric key. However, as we have discussed, the use of asymmetric-key algorithms is very inefficient for the encryption/decryption of a long message.

Most e-mail security protocols today require that encryption/decryption be done using a symmetric-key algorithm and a one-time secret key sent with the message. Alice can create a secret key and send it with the message she sends to Bob. To protect

the secret key from interception by Eve, the secret key is encrypted with Bob's public key. In other words, the secret key itself is encrypted.

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.

Certificates

One more issue needs to be considered before we discuss any e-mail security protocol in particular. It is obvious that some public-key algorithms must be used for e-mail security. For example, we need to encrypt the secret key or sign the message. To encrypt the secret key, Alice needs Bob's public key; to verify a signed message, Bob needs Alice's public key. So, for sending a small authenticated and confidential message, two public keys are needed. How can Alice be assured of Bob's public key, and how can Bob be assured of Alice's public key? Each e-mail security protocol has a different method of certifying keys.

16.2 PGP

The first protocol discussed in this chapter is called **Pretty Good Privacy (PGP).** PGP was invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication. PGP can be used to create a secure e-mail message or to store a file securely for future retrieval.

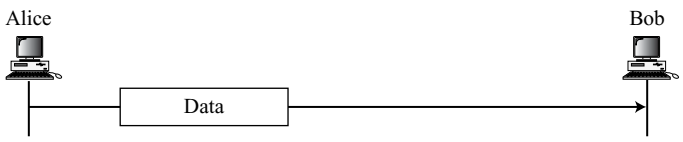
Scenarios

Let us first discuss the general idea of PGP, moving from a simple scenario to a complex one. We use the term "Data" to show the message or file prior to processing.

Plaintext

The simplest scenario is to send the e-mail message (or store the file) in plaintext as shown in Figure 16.2. There is no message integrity or confidentiality in this scenario. Alice, the sender, composes a message and sends it to Bob, the receiver. The message is stored in Bob's mailbox until it is retrieved by him.

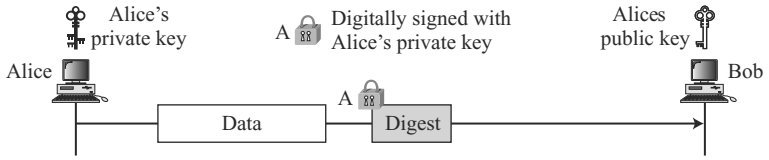
Figure 16.2 *A plaintext message*



Message Integrity

Probably the next improvement is to let Alice sign the message. Alice creates a digest of the message and signs it with her private key. When Bob receives the message, he verifies the message by using Alice's public key. Two keys are needed for this scenario. Alice needs to know her private key; Bob needs to know Alice's public key. Figure 16.3 shows the situation.

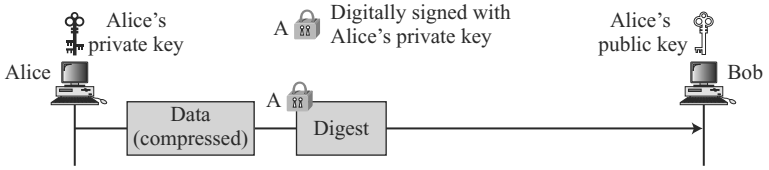
Figure 16.3 *An authenticated message*



Compression

A further improvement is to compress the message to make the packet more compact. This improvement has no security benefit, but it eases the traffic. Figure 16.4 shows the new scenario.

Figure 16.4 *A compressed message*

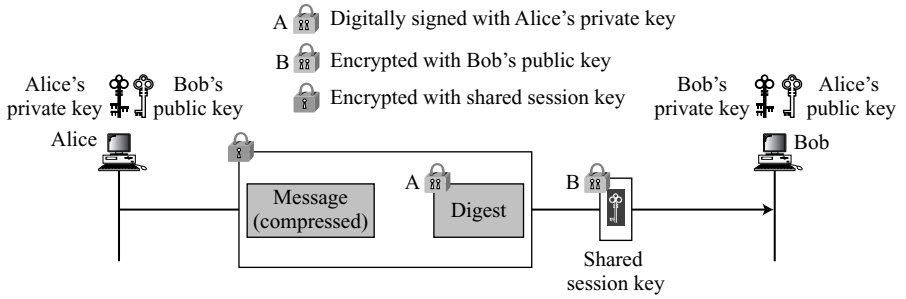


Confidentiality with One-Time Session Key

As we discussed before, confidentiality in an e-mail system can be achieved using conventional encryption with a one-time session key. Alice can create a session key, use the session key to encrypt the message and the digest, and send the key itself with the message. However, to protect the session key, Alice encrypts it with Bob's public key. Figure 16.5 shows the situation.

When Bob receives the packet, he first decrypts the key, using his private key to remove the key. He then uses the session key to decrypt the rest of the message. After decompressing the rest of the message, Bob creates a digest of the message and checks to see if it is equal to the digest sent by Alice. If it is, then the message is authentic.

Figure 16.5 *A confidential message*



Code Conversion

Another service provided by PGP is code conversion. Most e-mail systems allow the message to consist of only ASCII characters. To translate other characters not in the ASCII set, PGP uses Radix-64 conversion. Each character to be sent (after encryption) is converted to Radix-64 code, which is discussed later in the chapter.

Segmentation

PGP allows segmentation of the message after it has been converted to Radix-64 to make each transmitted unit the uniform size as allowed by the underlying e-mail protocol.

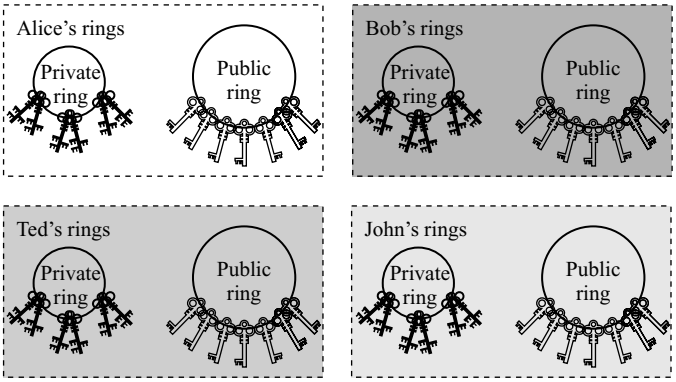
Key Rings

In all previous scenarios, we assumed that Alice needs to send a message only to Bob. That is not always the case. 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. Figure 16.6 shows a community of four people, each having a ring of pairs of private/public keys and, at the same time, a ring of public keys belonging to other people in the community.

Alice, for example, 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.
   a. She uses her private key to sign the digest.
   b. She uses the receiver's public key to encrypt a newly created session key.
   c. She encrypts the message and signed digest with the session key created.

Figure 16.6 *Key rings in PGP*



2. Alice receives a message from another person in the community.
   a. She uses her private key to decrypt the session key.
   b. She uses the session key to decrypt the message and digest.
   c. She uses her public key to verify the digest.

PGP Algorithms

The following algorithms are used in PGP.

Public-Key Algorithms The public-key algorithms that are used for signing the digests or encrypting the messages are listed in Table 16.1.

Table 16.1 *Public-key algorithms*

ID	*Description*
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 The symmetric-key algorithms that are used for conventional encrypting are shown in Table 16.2.

Table 16.2 *Symmetric-key algorithms*

ID	*Description*
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

Hash Algorithms The hash algorithms that are used for creating hashes in PGP are shown in Table 16.3.

Table 16.3 *Hash Algorithms*

ID	*Description*
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 The compression algorithms that are used for compressing text are shown in Table 16.4.

Table 16.4 *Compression methods*

ID	*Description*
0	Uncompressed
1	ZIP
2	ZLIP
100–110	Private methods

PGP Certificates

PGP, like other protocols we have seen so far, uses certificates to authenticate public keys. However, the process is totally different.

X.509 Certificates

Protocols that use X.509 certificates depend on the hierarchical structure of the trust. There is a predefined chain of trust from the root to any certificate. Every user fully trusts the authority of the CA at the root level (prerequisite). The root issues certificates for the CAs at the second level, a second level CA issues a certificate for the third level, and so on. Every party that needs to be trusted presents a certificate from some CA in the tree. If Alice does not trust the certificate issuer for Bob, she can appeal to a higher-level authority up to the root (which must be trusted for the system to work). In other words, there is one single path from a fully trusted CA to a certificate.

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

PGP Certificates

In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring. Bob can sign a certificate for Ted, John, Anne, and so on. There is no hierarchy of trust in PGP; there is no tree. The lack of hierarchical structure may result in the fact that Ted may have one certificate from Bob and another certificate from Liz. If Alice wants to follow the line of certificates for Ted, there are two paths: one starts from Bob and one starts from Liz. An interesting point is that Alice may fully trust Bob, but only partially trust Liz. There can be multiple paths in the line of trust from a fully or partially trusted authority to a certificate. In PGP, the issuer of a certificate is usually called an *introducer.*

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

Trusts and Legitimacy

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. (Even in real life we cannot fully trust everyone that we know.) To solve this problem, 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. The following scenarios can happen.

1. 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.
2. 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.
3. 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.
4. 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

You might have realized a problem with the above discussion. What if nobody sends a certificate for a fully or partially trusted entity? For example, how can the legitimacy of Bob's public key be determined if no one has sent a certificate for Bob? In PGP, the key legitimacy of a trusted or partially trusted entity can be also determined by other methods.

1. 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.

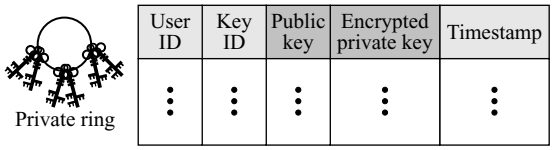
2. If Bob's voice is recognizable to Alice, Alice can call him and obtain his public key on the phone.
3. A better solution proposed by PGP is for Bob to send his public key to Alice by e-mail. Both Alice and Bob make a 16-byte MD5 (or 20-byte SHA-1) digest from the key. The digest is normally displayed 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. If the key is altered or changed during the e-mail transmission, the two fingerprints do not match. To make it even more convenient, PGP has created a list of words, each representing a 4-digit combination. When Alice calls Bob, Bob can pronounce the eight words (or ten words) for Alice. The words are carefully chosen by PGP to avoid those similar in pronunciation; for example, if *sword* is in the list, *word* is not.
4. 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

Each user, such as Alice, keeps track of two key rings: one private-key ring and one public key ring. PGP defines a structure for each of these key rings in the form of a table.

Private Key Ring Table Figure 16.7 shows the format of a private key ring table.

Figure 16.7 *Format of private key ring table*



- **User ID.** The user ID is usually the e-mail address of the user. However, the user may designate a unique e-mail address or alias for each key pair. The table lists the user ID associated with each pair.
- **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 (key mod 2^{64}). The key ID is needed for the operation of PGP because Bob may have several public keys belonging to Alice in his public key ring. When he receives a message from Alice, Bob must know which key ID to use to verify the message. The key ID, which is sent with the message, as we will see shortly, enables Bob to use a specific public key for Alice from his public ring. You might ask why the entire public key is not sent. The answer is that in public-key cryptography, the size of the public key may be very long. Sending just 8 bytes reduces the size of the message.
- **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 16.1

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.

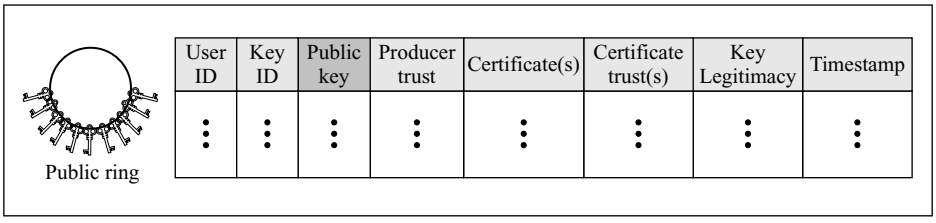
Table 16.5 *Private key ring table for Example 1*

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

Note that although the values of key ID, public key, and private key are shown in hexadecimal, and *ddmmyy-time* format is used for the timestamp, these formats are only for presentation and may be different in an actual implementation.

Public Key Ring Table Figure 16.8 shows the format of a public key ring table.

Figure 16.8 *Format of a public key ring table*



- **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 16.2

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.

Table 16.6 *Example 2, starting table*

User ID	*Key ID*	*Public key*	*Prod. trust*	*Certificate*	*Cert. trust*	*Key legit.*	*Time-stamp*
Alice...	AB...	AB.....	F			F	.....

Note that, based on this table, we assume that Alice has issued a certificate for herself (implicitly). Alice of course trusts herself fully. The producer level of trust is also *full* and so is the key legitimacy. Although Alice never uses this first row, it is needed for the operation of PGP.

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. Table 16.7 shows this new event.

Table 16.7 *Example 2, after Bob is added to the table*

User ID	*Key ID*	*Public key*	*Prod. trust*	*Certificate*	*Cert. trust*	*Key legit.*	*Time-stamp*
Alice...	AB...	AB.....	F			F	.....
Bob...	12...	12.....	F			F	.....

Note that the value of the producer trust is *full* for Bob because Alice fully trusts Bob. The value of the certificate field is empty, which shows that this key has been received indirectly, and not by a certificate.

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, as shown in Table 16.8.

Table 16.8 *Example 2, after Ted is added to the table*

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

Note that the value of certificate field shows that the certificate was received from Bob. The value of the certificate trust is copied by PGP from Bob's producer trust field. The value of the key legitimacy field is the value of the certificate trust multiplied by 1 (the weight).

4. Now Alice adds Anne to the list. Alice partially trusts Anne, but Bob, who is fully trusted, sends a certificate for Anne. Table 16.9 shows the new event.

Table 16.9 *Example 2, after Anne is added to the table*

User ID	*Key ID*	*Public key*	*Prod. trust*	*Certificate*	*Cert. trust*	*Key legit.*	*Time-stamp*
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	.....

Note that the producer trust value for Anne is partial, but the certificate trust and key legitimacy is full.

5. Now Anne introduces John, who is not trusted by Alice. Table 16.10 shows the new event.

Table 16.10 *Example 2, after John is added to the table*

User ID	*Key ID*	*Public key*	*Prod. Trust*	*Certificate*	*Cert. trust*	*Key legit.*	*Time-stamp*
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	.....

Note that PGP has copied the value of Anne's producer trust (P) to the certificate trust field for John. The value of the key legitimacy field for John is 1/2 (P) at this moment, which means that Alice must not use John's key until it changes to 1 (F).

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 as shown in Table 16.11.

   Because John has two certificates in Alice's table and his key legitimacy value is 1, Alice can use his key. But John is still untrustworthy. Note that Alice can continue to add entries to the table.

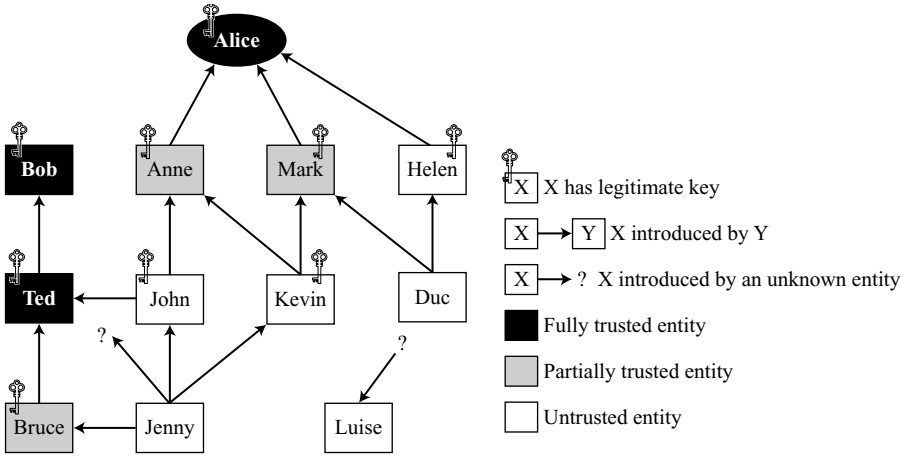
Table 16.11 *Example 2, after one more certificate received for John*

User ID	*Key ID*	*Public key*	*Prod. trust*	*Certificate*	*Cert. trust*	*Key legit.*	*Time-stamp*
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

As Zimmermann has proposed, we can create a trust model for any user in a ring with the user as the center of activity. Such a model can look like the one shown in Figure 16.9. The figure shows the trust model for Alice at some moment. The diagram may change with any changes in the public key ring table.

Figure 16.9 *Trust model*



Let us elaborate on the figure. Figure 16.9 shows that there are three entities in Alice's ring with full trust (Alice herself, Bob, and Ted). The figure also shows three entities with partial trust (Anne, Mark, and Bruce). There are also six entities with no trust. Nine entities have a legitimate key. Alice can encrypt a message to any one of these entities or verify a signature received from one of these entities (Alice's key is never used in this model). There are also three entities that do not have any legitimate keys with Alice.

Bob, Anne, and Mark have made their keys legitimate by sending their keys by e-mail and verifying their fingerprints by phone. Helen, on the other hand, has sent a certificate from a CA because she is not trusted by Alice and verification on the phone is not possible. Although Ted is fully trusted, he has given Alice a certificate signed by Bob. John has sent Alice two certificates, one signed by Ted and one by Anne. Kevin has sent two certificates to Alice, one signed by Anne and one by Mark. Each of these certificates gives Kevin half a point of legitimacy; therefore, Kevin's key is legitimate. Duc has sent two certificates to Alice, one signed by Mark and the other by Helen. Since Mark is half-trusted and Helen is not trusted, Duc does not have a legitimate key. Jenny has sent four certificates, one signed by a half-trusted entity, two by untrusted entities, and one by an unknown entity. Jenny does not have enough points to make her key legitimate. Luise has sent one certificate signed by an unknown entity. Note that Alice may keep Luise's name in the table in case future certificates for Luise arrive.

Web of Trust

PGP can eventually make a **web of trust** between a group of people. If each entity introduces more entities to other entities, the public key ring for each entity gets larger and larger and entities in the ring can send secure e-mail to each other.

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

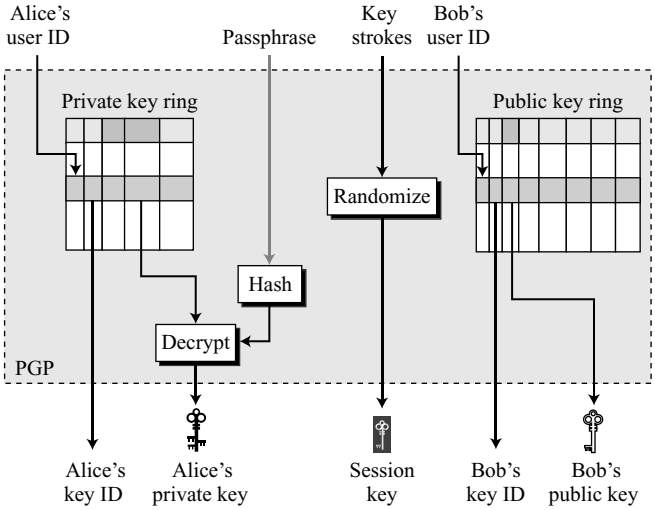
As we have seen, the sender and receiver each have two key rings, one private and one public. Let us see how information needed for sending and receiving a message is extracted from these rings.

Sender Site

Assume that Alice is sending an e-mail to Bob. Alice needs five pieces of information: the key ID of the public key she is using, her private key, the session key, Bob's public-key ID, and Bob's public key. To obtain these five pieces of information, Alice needs to feed four pieces of information to PGP: her user ID (for this e-mail), her passphrase, a sequence of key strokes with possible pauses, and Bob's user ID. See Figure 16.10.

Alice's public-key ID (to be sent with the message) and her private key (to sign the message) are stored in the private key ring table. Alice selects the user ID (her e-mail address) that she wants to use as an index to this ring. PGP extracts the key ID and the encrypted private key. PGP uses the predefined decryption algorithm and her hashed passphrase (as the key) to decrypt this private key.

Figure 16.10 *Extracting information at the sender site*



Alice also needs a secret session key. The session key in PGP is a random number with a size defined in the encryption/decryption algorithm. PGP uses a random number generator to create a random session key; the seed is a set of arbitrary keystrokes typed by Alice on her keyboard. Each key stroke is converted to 8 bits and each pause between the keystrokes is converted to 32 bits. The combination goes through a complex random number generator to create a very reliable random number as the session key. Note that the session key in PGP is a one-time random key (see Appendix K) and used only once.

Alice also needs Bob's key ID (to be sent with the message) and Bob's public key (to encrypt the session key). These two pieces of information are extracted from the public key ring table using Bob's user ID (his e-mail address).

Receiver Site

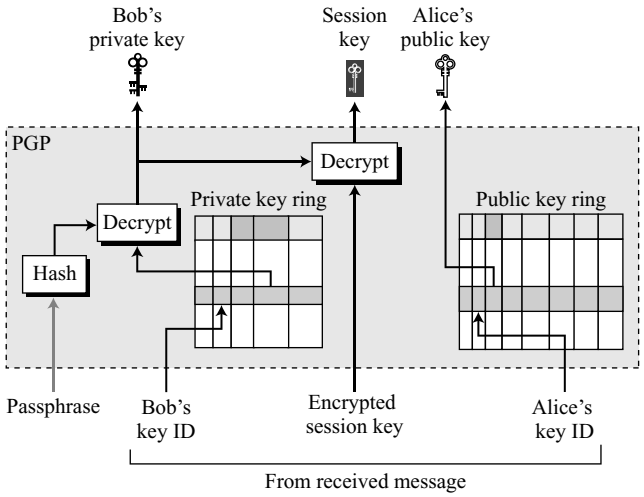
At the receiver site, Bob needs three pieces of information: Bob's private key (to decrypt the session key), the session key (to decrypt the data), and Alice's public key (to verify the signature). See Figure 16.11.

Bob uses the key ID of his public key sent by Alice to find his corresponding private key needed to decrypt the session key. This piece of information can be extracted from Bob's private key ring table. The private key, however, is encrypted when stored. Bob needs to use his passphrase and the hash function to decrypt it.

The encrypted session key is sent with the message; Bob uses his decrypted private key to decrypt the session key.

Bob uses Alice's key ID sent with the message to extract Alice's public key, which is stored in Bob's public key ring table.

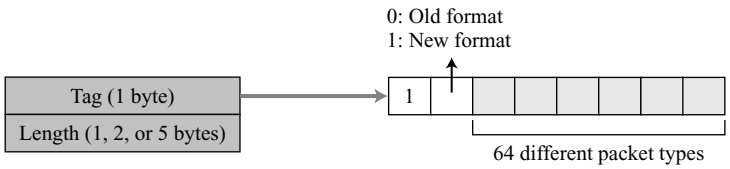
Figure 16.11 *Extracting information at the receiver site*



PGP Packets

A message in PGP consists of one or more packets. During the evolution of PGP, the format and the number of packet types have changed. Like other protocols we have seen so far, PGP has a generic header that applies to every packet. The generic header, in the most recent version, has only two fields, as shown in Figure 16.12.

Figure 16.12 *Format of packet header*



- **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. The receiver can determine

Table 16.12 *Some commonly used packet types*

Value	*Packet type*
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

the number of bytes of the length field by looking at the value of the byte immediately following the tag field.

a. 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
```

b. 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
```

c. 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)
```

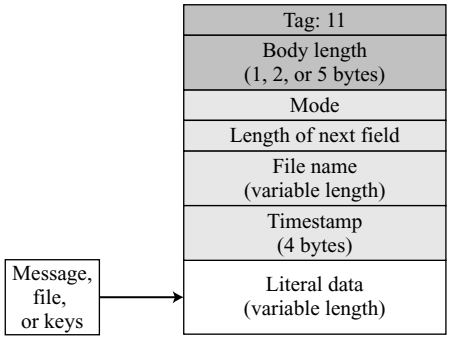
Note that the formula means $1 \times 2^{(\text{first byte \& 0x1F})}$. The power is actually the value of the five rightmost bits. Because the field is between 224 and 254, inclusive, the value of the five rightmost bits is between 0 and 30, inclusive. In other words, the partial body length can be between one (2^0) and 1,073,741,824 (2^{30}). When a packet becomes several partial bodies, the partial body length is applicable. Each partial body length defines one part of the length. The last length field cannot be a partial body length definer. For example, if a packet has four parts, it can have three partial length fields and one length field of another type.

d. 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 The literal data packet is the packet that carries or holds the actual data that is being transmitted or stored. This packet is the most elementary type of message; that is, it cannot carry any other packet. The format of the packet is shown in Figure 16.13.

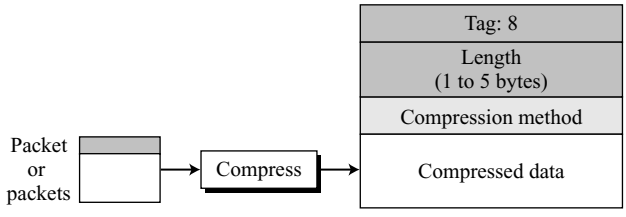
Figure 16.13 *Literal data packet*



- **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 This packet carries compressed data packets. Figure 16.14 shows the format of a compressed data packet.

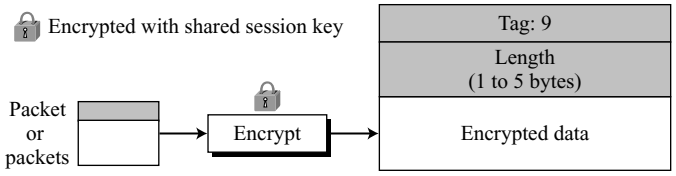
Figure 16.14 *Compressed data packet*



- **Compression method.** This one-byte field defines the compression method used to compress the data (next field). The values defined for this field so far are 1 (ZIP) and 2 (ZLIP). Also, an implementation can use other experimental compression methods. ZIP is discussed in Appendix M.
- **Compressed data.** This variable-length field carries the data after compression. Note that the data in this field can be one packet or the concatenation of two or more packets. The common situation is a single literal data packet or a combination of a signature packet followed by a literal data packet.

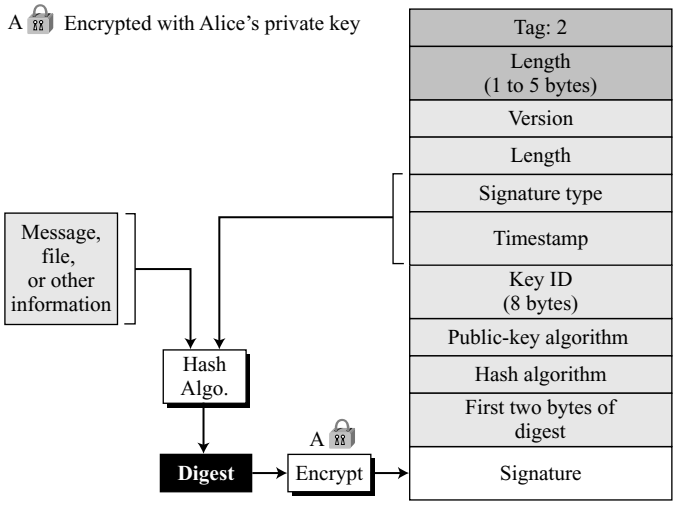
Data Packet Encrypted with Secret Key This packet carries data from one packet or a combination of packets that have been encrypted using a conventional symmetric-key algorithm. Note that a packet carrying the one-time session key must be sent before this packet. Figure 16.15 shows the format of the encrypted data packet.

Figure 16.15 *Encrypted data packet*



Signature Packet A signature packet, as we discussed before, protects the integrity of the data. Figure 16.16 shows the format of the signature packet.

Figure 16.16 *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. Table 16.13 shows some signature types.

Table 16.13 *Some signature values*

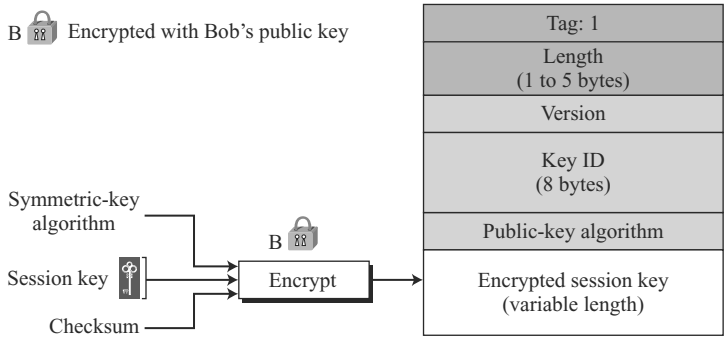
Value	*Signature*
0x00	Signature of a binary document (message or file).
0x01	Signature of a text document (message or file).
0x10	Generic certificate of a user ID and public-key packet. The signer does not make any particular assertion about the owner of the key.
0x11	Personal certificate of a user ID and public-key packet. No verification is done on the owner of the key.
0x12	Casual certificate of a User ID and public-key packet. Some casual verification done on the owner of the key.
0x13	Positive certificate of a user ID and public-key packet. Substantial verification done.
0x30	Certificate revocation signature. This removes an earlier certificate (0*x*10 through 0*x*13).

- **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 This packet is used to send the session key encrypted with the receiver public key. The format of the packet is shown in Figure 16.17.

- **Version.** This one-byte field defines the PGP version being used.
- **Key ID.** This eight-byte field defines the public-key ID of the sender. It indicates to the receiver which sender public key should be used to decrypt the session key.
- **Public-key algorithm.** This one-byte field gives the code for the public-key algorithm used to encrypt the session key. The receiver uses the same algorithm to decrypt the session key.

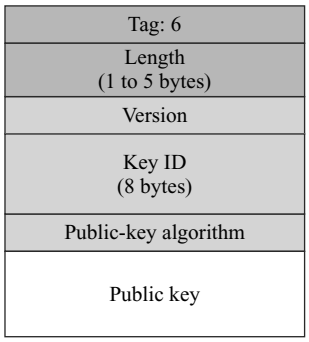
Figure 16.17 *Session-key packet*



- **Encrypted session.** This variable-length field is the encrypted value of the session key created by the sender and sent to the receiver. The encryption is done on the following:
  a. One-octet symmetric encryption algorithm
  b. The session key
  c. A two-octet checksum equal to the sum of the preceding session-key octets

Public-Key Packet This packet contains the public key of the sender. The format of the packet is shown in Figure 16.18.

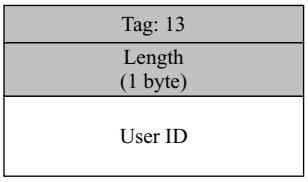
Figure 16.18 *Public-key packet*



- **Version.** This one-byte field defines the PGP version of the PGP being used.
- **Timestamp.** This four-byte field defines the time the key was created.
- **Validity.** This two-byte field shows the number of days the key is valid. If the value is 0, it means the key does not expire.
- **Public-key algorithm.** This one-byte field gives the code for the public-key algorithm.
- **Public key.** This variable-length field holds the public key itself. Its contents depend on the public-key algorithm used.

User ID Packet This packet identifies a user and can normally associate the user ID contents with a public key of the sender. Figure 16.19 shows the format of the user ID packet. Note that the length field of the general header is only one byte.

Figure 16.19 *User ID packet*



- **User ID.** This variable-length string defines the user ID of the sender. It is normally the name of the user followed by an e-mail address.

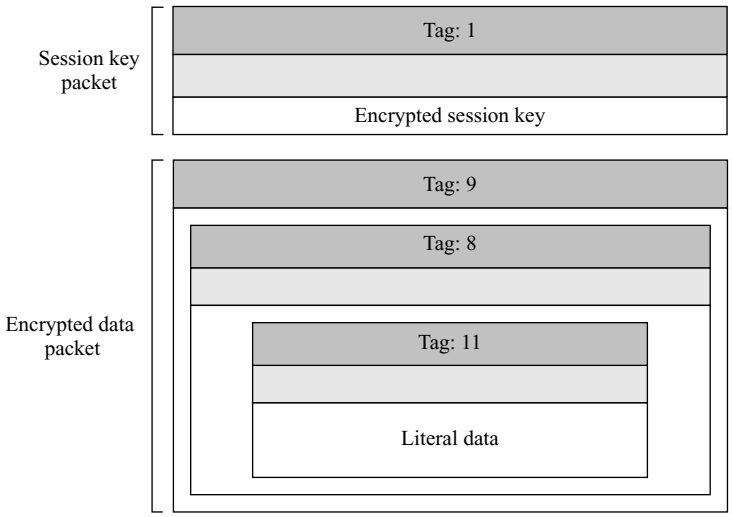
PGP Messages

A message in PGP is a combination of sequenced and/or nested packets. Even though not all combinations of packets can make a message, the list of combinations is still long. In this section, we give a few examples to show the idea.

Encrypted Message

An encrypted message can be a sequence of two packets, a session-key packet and a symmetrically encrypted packet. The latter is normally a nested packet. Figure 16.20 shows this combination.

Figure 16.20 *Encrypted message*

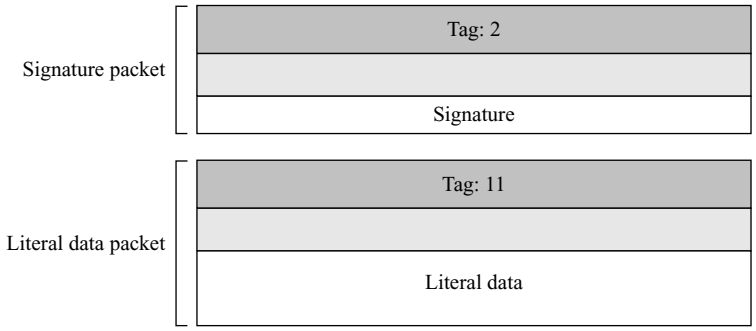


Note that the session-key packet is just a single packet. The encrypted data packet, however, is made of a compressed packet. The compressed packet is made of a literal data packet. The last one holds the literal data.

Signed Message

A signed message can be the combination of a signature packet and a literal packet, as shown in Figure 16.21.

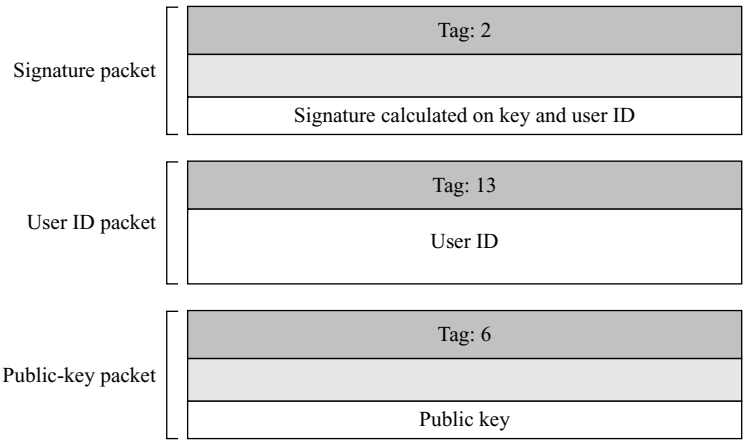
Figure 16.21 *Signed message*



Certificate Message

Although a certificate can take many forms, one simple example is the combination of a user ID packet and a public-key packet as shown in Figure 16.22. The signature is then calculated on the concatenation of the key and user ID.

Figure 16.22 *Certificate message*



Applications of PGP

PGP has been extensively used for personal e-mails. It will probably continue to be.

16.3 S/MIME

Another security service designed for electronic mail is **Secure/Multipurpose Internet Mail Extension (S/MIME).** The protocol is an enhancement of the **Multipurpose Internet Mail Extension (MIME)** protocol. To better understand S/MIME, first we briefly describe MIME. Next, S/MIME is discussed as the extension to MIME.

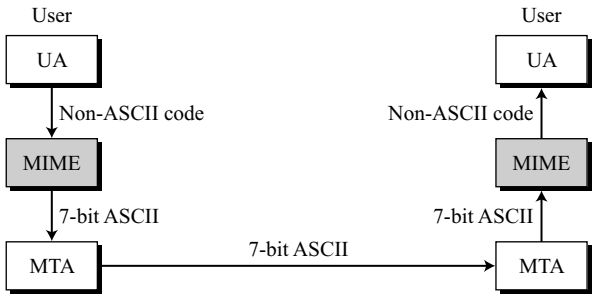
MIME

Electronic mail has a simple structure. Its simplicity, however, comes with a price. It can send messages only in NVT 7-bit ASCII format. In other words, it has some limitations. For example, it cannot be used for languages that are not supported by 7-bit ASCII characters (such as Arabic, Chinese, French, German, Hebrew, Japanese, and Russian). Also, it cannot be used to send binary files or video or audio data.

Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving side is transformed back to the original data.

We can think of MIME as a set of software functions that transform non-ASCII data to ASCII data, and vice versa, as shown in Figure 16.23.

Figure 16.23 *MIME*



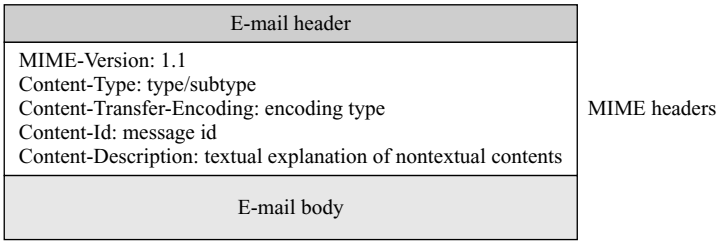
MIME defines five headers that can be added to the original e-mail header section to define the transformation parameters:

1. MIME-Version
2. Content-Type

3. Content-Transfer-Encoding
4. Content-Id
5. Content-Description

Figure 16.24 shows the MIME headers. We will describe each header in detail.

Figure 16.24 *MIME header*



MIME-Version

This header defines the version of MIME used. The current version is 1.1.

MIME-Version: 1.1

Content-Type

This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters.

Content-Type: <type / subtype; parameters>

MIME allows seven different types of data. These are listed in Table 16.14 and described in more detail below.

- **Text.** The original message is in 7-bit ASCII format and no transformation by MIME is needed. There are two subtypes currently used, *plain* and *HTML*.
- **Multipart.** The body contains multiple, independent parts. The multipart header needs to define the boundary between each part. A parameter is used for this purpose. The parameter is a string token that comes before each part; it is on a separate line by itself and is preceded by two hyphens. The body is terminated using the boundary token, again preceded by two hyphens, and then terminated with two hyphens.

  Four subtypes are defined for this type: *mixed, parallel, digest,* and *alternative*. In the mixed subtype, the parts must be presented to the recipient in the exact order

Table 16.14 *Data types and subtypes in MIME*

Type	*Subtype*	*Description*
	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).

as in the message. Each part has a different type and is defined at the boundary. The parallel subtype is similar to the mixed subtype, except that the order of the parts is unimportant. The digest subtype is also similar to the mixed subtype except that the default type/subtype is message/RFC822, as defined below. In the alternative subtype, the same message is repeated using different formats. The following is an example of a multipart message using a mixed subtype:

```
Content-Type: multipart/mixed; boundary=xxxx

--xxxx
Content-Type: text/plain;
.....
--xxxx
Content-Type: image/gif;
.....
--xxxx--
```

- **Message.** In the message type, the body is itself an entire mail message, a part of a mail message, or a pointer to a message. Three subtypes are currently used: *RFC822, partial,* and *external-body.* The subtype RFC822 is used if the body is encapsulating another message (including header and the body). The partial subtype

is used if the original message has been fragmented into different mail messages and this mail message is one of the fragments. The fragments must be reassembled at the destination by MIME. Three parameters must be added: *id, number,* and the *total*. The id identifies the message and is present in all the fragments. The number defines the sequence order of the fragment. The total defines the number of fragments that comprise the original message. The following is an example of a message with three fragments:

```
Content-Type: message/partial;
id="forouzan@challenger.atc.fhda.edu";
number=1;
total=3;

.....
.....
```

The subtype external-body indicates that the body does not contain the actual message but is only a reference (pointer) to the original message. The parameters following the subtype define how to access the original message. The following is an example:

```
Content-Type: message/external-body;
name="report.txt";
site="fhda.edu";
access-type="ftp";

.....
.....
```

- **Image.** The original message is a stationary image, indicating that there is no animation. The two currently used subtypes are *Joint Photographic Experts Group (JPEG),* which uses image compression, and *Graphics Interchange Format (GIF)*.
- **Video.** The original message is a time-varying image (animation). The only subtype is Moving Picture Experts Group (*MPEG*). If the animated image contains sounds, it must be sent separately using the audio content type.
- **Audio.** The original message is sound. The only subtype is basic, which uses 8 kHz standard audio data.
- **Application.** The original message is a type of data not previously defined. There are only two subtypes used currently: *PostScript* and *octet-stream.* PostScript is used when the data are in Adobe PostScript format. Octet-stream is used when the data must be interpreted as a sequence of 8-bit bytes (binary file).

Content-Transfer-Encoding

This header defines the method used to encode the messages into 0s and 1s for transport:

Content-Transfer-Encoding: <type>

The five types of encoding methods are listed in Table 16.15.

Table 16.15 *Content-transfer-encoding*

Type	*Description*
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.

- **7bit.** This is 7-bit NVT ASCII encoding. Although no special transformation is needed, the length of the line should not exceed 1,000 characters.
- **8bit.** This is 8-bit encoding. Non-ASCII characters can be sent, but the length of the line still should not exceed 1,000 characters. MIME does not do any encoding here; the underlying SMTP protocol must be able to transfer 8-bit non-ASCII characters. It is, therefore, not recommended. Radix-64 and quoted-printable types are preferable.
- **Binary.** This is 8-bit encoding. Non-ASCII characters can be sent, and the length of the line can exceed 1,000 characters. MIME does not do any encoding here; the underlying SMTP protocol must be able to transfer binary data. It is, therefore, not recommended. Radix-64 and quoted-printable types are preferable.
- **Radix-64.** This is a solution for sending data made of bytes when the highest bit is not necessarily zero. Radix-64 transforms this type of data to printable characters, which can then be sent as ASCII characters or any type of character set supported by the underlying mail transfer mechanism.

  Radix-64 divides the binary data (made of streams of bits) into 24-bit blocks. Each block is then divided into four sections, each made of 6 bits (see Figure 16.25).

  Each 6-bit section is interpreted as one character according to Table 16.16.
- **Quoted-printable.** Radix-64 is a redundant encoding scheme; that is, 24 bits become four characters, and eventually are sent as 32 bits. We have an overhead of 25 percent. If the data consist mostly of ASCII characters with a small non-ASCII portion, we can use **quoted-printable** encoding. If a character is ASCII, it is sent as is. If a character is not ASCII, it is sent as three characters. The first character is the equal sign (=). The next two characters are the hexadecimal representations of the byte. Figure 16.26 shows an example.

Figure 16.25 *Radix-64 conversion*

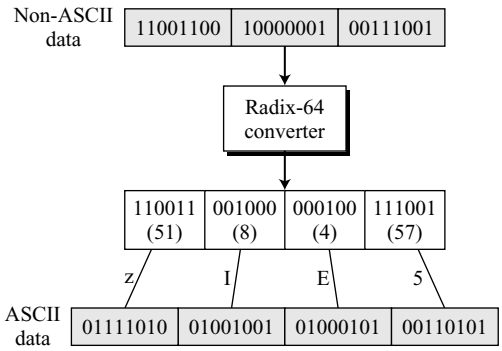
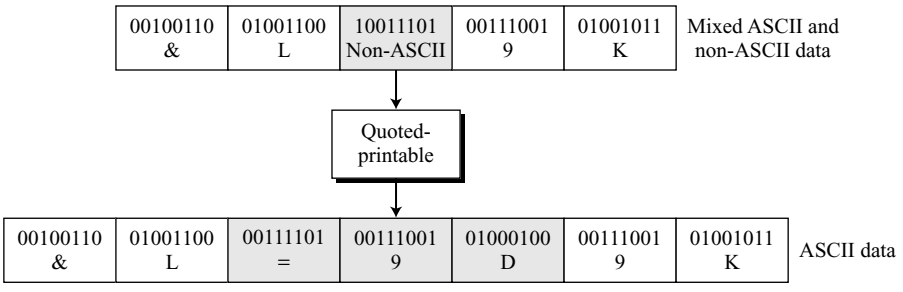


Table 16.16 *Radix-64 encoding table*

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

Figure 16.26 *Quoted-printable*



Content-Id

This header uniquely identifies the whole message in a multiple message environment.

Content-Id: id=<content-id>

Content-Description

This header defines whether the body is image, audio, or video.

Content-Description: <description>

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."

Cryptographic Message Syntax (CMS)

To define how security services, such as confidentiality or integrity, can be added to MIME content types, S/MIME has defined **Cryptographic Message Syntax (CMS).** The syntax in each case defines the exact encoding scheme for each content type. The following describe the type of message and different subtypes that are created from these messages. For details, the reader is referred to RFC 3369 and 3370.

Data Content Type This is an arbitrary string. The object created is called *Data*.

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*. Figure 16.27 shows the process of creating an object of this type. 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.

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.* Figure 16.28 shows the process of creating an object of this type.

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.

Figure 16.27 *Signed-data content type*

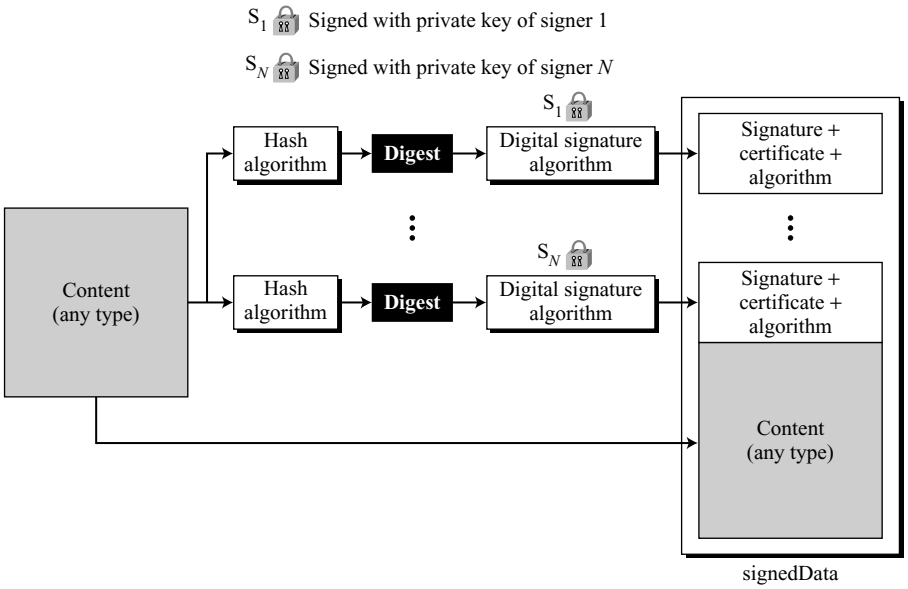
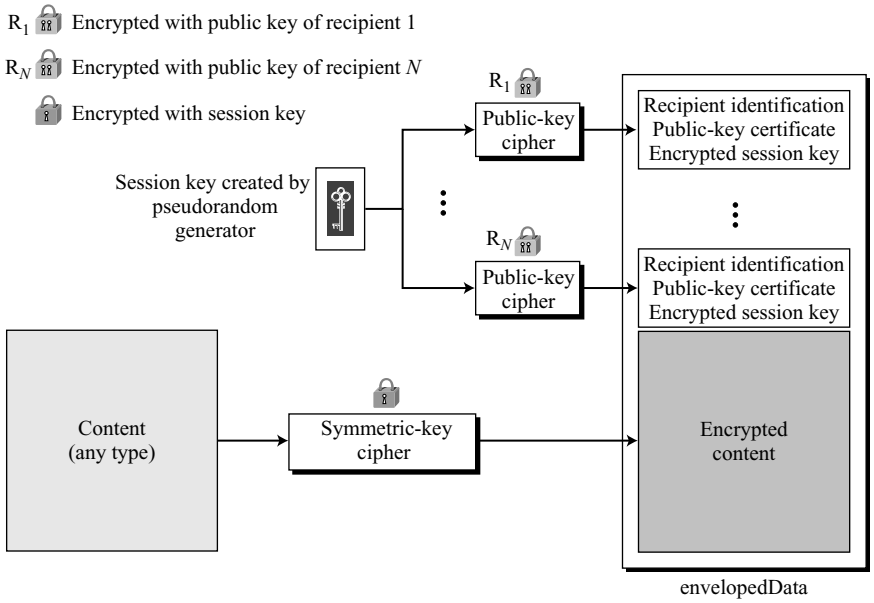
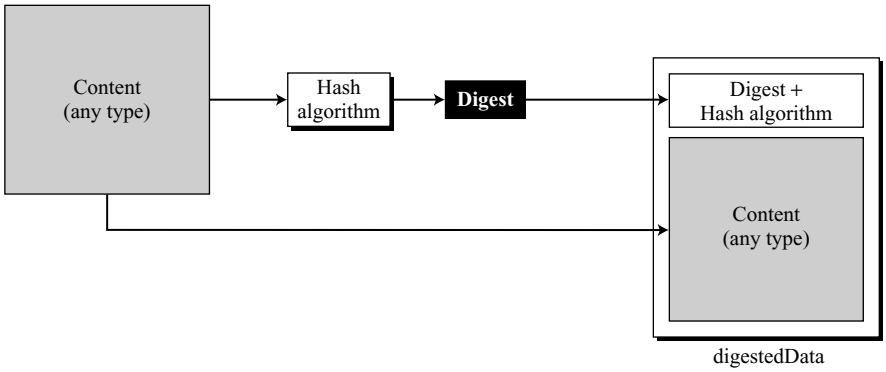


Figure 16.28 *Enveloped-data content type*



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.

Figure 16.29 *Digest-data content type*



1. A message digest is calculated from the content.
2. The message digest, the algorithm, and the content are added together to create the *digestedData* object.

Encrypted-Data Content Type This type is used to create an encrypted version of any content type. Although this looks like the enveloped-data content type, the encrypted-data content type has no recipient. It can be used to store the encrypted data instead of transmitting it. The process is very simple, the user employs any key (normally driven from the password) and any algorithm to encrypt the content. The encrypted content is stored without including the key or the algorithm. The object created is called *encryptedData*.

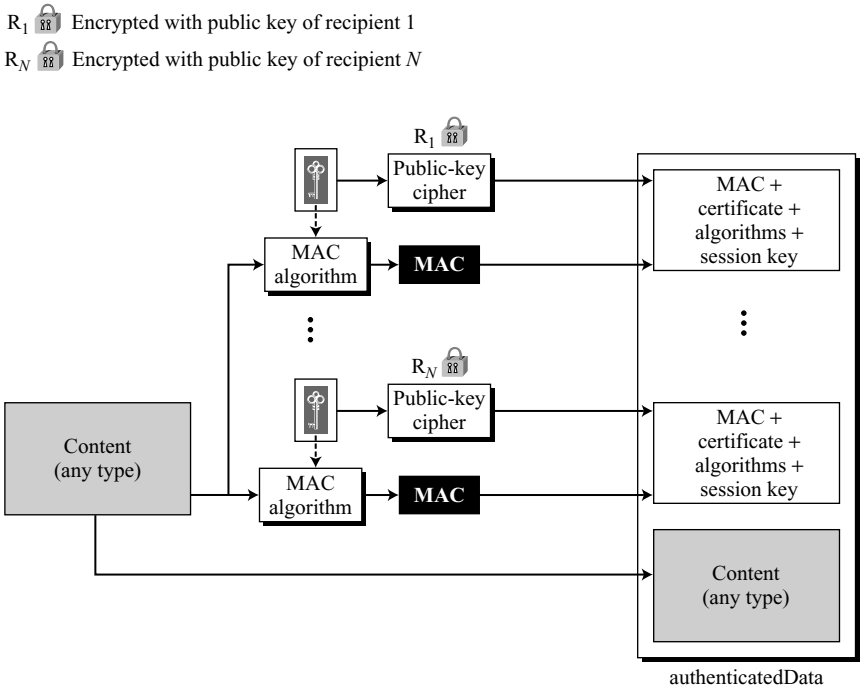
Authenticated-Data Content Type This type is used to provide authentication of the data. The object is called *authenticatedData*. Figure 16.30 shows the process.

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 informations are collected together to form the authenticatedData object.

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.

Figure 16.30 *Authenticated-data content type*



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.

Table 16.17 *Cryptographic algorithm for S/MIME*

Algorithm	*Sender must support*	*Receiver must support*	*Sender should support*	*Receiver should support*
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		

Example 16.3

The following shows an example of an enveloped-data in which a small message is encrypted using triple DES.

```
Content-Type: application/pkcs7-mime; mime-type=enveloped-data
Content-Transfer-Encoding: Radix-64
Content-Description: attachment
name="report.txt";
cb32ut67f4bhijHU21oi87eryb0287hmnklsgFDoY8bc659GhIGfH6543mhjkdsaH23YjBnmN
ybmlkzjhgfdyhGe23Kjk34XiuD678Es16se09jy76jHuytTMDcbnmlkjgfFdiuyu678543m0n3h
G34un12P2454Hoi87e2ryb0H2MjN6KuyrlsgFDoY897fk923jljk1301XiuD6gh78EsUyT23y
```

Applications of S/MIME

It is predicted that S/MIME will become the industry choice to provide security for commercial e-mail.

16.4 RECOMMENDED READING

The following books and websites give more details about subjects discussed in this chapter. The items in brackets refer to the reference list at the end of the text.

Books

Electronic mail is discussed in [For06] and [For07]. PGP is discussed in [Sta06], [KPS02], and [Rhe03]. S/MIME is discussed in [Sta06] and [Rhe03].

WebSites

The following websites give more information about topics discussed in this chapter.

http://axion.physics.ubc.ca/pgp-begin.html
csrc.nist.gov/publications/nistpubs/800-49/sp800-49.pdf
www.faqs.org/rfcs/rfc2632.html

16.5 KEY TERMS

Cryptographic Message Syntax (CMS)
electronic mail (e-mail)
key ring
message access agent (MAA)
message transfer agent (MTA)
Multipurpose Internet Mail Extension (MIME)
Pretty Good Privacy (PGP)
quoted-printable
Radix-64 encoding
Secure/Multipurpose Internet Mail Extension (S/MIME)
user agent (UA)
web of trust

16.6 SUMMARY

- Because there is no session in e-mail communication, the sender of the message needs to include the name or identifiers of the algorithms used in the message. In e-mail communication, 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.
- The first protocol discussed in this chapter is called Pretty Good Privacy (PGP), which was invented by Phil Zimmermann to provide e-mail with privacy, integrity, and authentication. PGP can be used to create a secure e-mail message or to store a file securely for future retrieval.
- In PGP, Alice needs a ring of public keys for each person with whom Alice needs to correspond. She also needs a ring of private/public keys belonging to her.
- In PGP, there is no need for CAs; anyone in the ring can sign a certificate for anyone else in the ring. There is no hierarchy of trust in PGP; there is no tree. There can be multiple paths from fully or partially trusted authorities to any subject.
- The entire operation of PGP is based on introducer trust, levels of trust, and the legitimacy of the public keys. PGP makes a web of trust between a group of people.
- PGP has defined several packet types: literal data packet, compressed data packet, data packet encrypted with secret key, signature packet, session-key packet encrypted with public key, public-key packet, and user ID packet.
- In PGP, we can have several types of messages: encrypted message, signed message, and certificate message.
- Another security service designed for electronic mail is Secure/Multipurpose Internet Mail Extension (S/MIME). The protocol is an enhancement of the Multipurpose Internet Mail Extension (MIME) protocol, which is a supplementary protocol that allows non-ASCII data to be sent through e-mail. S/MIME adds some new content types to MIME to provide security services.
- Cryptographic Message Syntax (CMS) has defined several message types that produce new content types to be added to MIME. This chapter mentioned several message types, including data content type, signed-data content type, enveloped-data content type, digested-data content type, encrypted-data content type, and authenticated-data content type.
- 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.

16.7 EXERCISES

Review Questions

1. Explain how Bob finds out what cryptographic algorithms Alice has used when he receives a PGP message from her.

2. Explain how Bob finds out what cryptographic algorithms Alice has used when he receives an S/MIME message from her.
3. In PGP, explain how Bob and Alice exchange the secret key for encrypting messages.
4. In S/MIME, explain how Bob and Alice exchange the secret key for encrypting messages.
5. Compare and contrast the nature of certificates in PGP and S/MIME. Explain the web of trust made from certificates in PGP and in S/MIME.
6. Name seven types of packets used in PGP and explain their purposes.
7. Name three types of messages in PGP and explain their purposes.
8. Name all content types defined by CMS and their purposes.
9. Compare and contrast key management in PGP and S/MIME.

Exercises

10. Bob receives a PGP message. How can he find out the type of the packet if the tag value is
    a. 8
    b. 9
    c. 2
11. In PGP, can an e-mail message use two different public-key algorithms for encryption and signing? How is this defined in a message sent from Alice to Bob?
12. Answer the following questions about tag values in PGP:
    a. Can a packet with a tag value of 1 contain another packet?
    b. Can a packet with a tag value of 6 contain another packet?
13. What types of a packet should be sent in PGP to provide the following security services:
    a. Confidentiality
    b. Message integrity
    c. Authentication
    d. Nonrepudiation
    e. Combination of a and b
    f. Combination of a and c
    g. Combination of a, b, and c
    h. Combination of a, b, c, and d.
14. What content type in S/MIME provides the following security services:
    a. confidentiality
    b. message integrity
    c. authentication
    d. nonrepudiation
    e. combination of a and b