UNIT – IV
Cryptographic Data Integrity Algorithms

Syllabus:

Message authentication Codes: Message authentication requirements, Message authentication functions, HMAC, Digital Signatures: Digital Signatures properties and requirements, Digital Signature Standard.

E-mail Security: PGP, S/MIME.

Message authentication Codes

One of the most fascinating and complex areas of cryptography are that of message authentication and the related area of digital signatures. It would be impossible, in anything less than book length, to exhaust all the cryptographic functions and protocols that have been proposed or implemented for message authentication and dig- ital signatures. Instead, the purpose of this chapter and the next is to provide a broad overview of the subject and to develop a systematic means of describing the various approaches.

Message authentication is a mechanism or service used to verify the integrity of a message. Message authentication assures that data received are exactly as sent by (i.e., contain no modification, insertion, deletion, or replay) and that the purported identity of the sender is valid.

- Symmetric encryption provides authentication among those who share the secret key.
- A message authentication code (MAC) is an algorithm that requires the use of a secret key. A MAC takes a variable-length message and a secret key as input and produces an authentication code. A recipient in posses- sion of the secret key can generate an authentication code to verify the integrity of the message.
- One means of forming a MAC is to combine a cryptographic hash function in some fashion with a secret key.
- Another approach to constructing a MAC is to use a symmetric block cipher in such a way that it produces a fixed-length output for a variable- length input.

4.1. MESSAGE AUTHENTICATION REQUIREMENTS:

In the context of communications across a network, the following attacks can be identified.

1. **Disclosure:** Release of message contents to any person or process not possessing the appropria te cryptographic key.
2. **Traffic analysis:** Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either a connection-oriented or connectionless environment, the number and length of messages between parties could be determined.
3. **Masquerade:** Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. Also included are fraudulent acknowledgments of message receipt or non receipt by someone other than the message recipient.
4. **Content modification:** Changes to the contents of a message, including insertion, deletion, transposition, and modification.
5. **Sequence modification:** Any modification to a sequence of messages between parties, including i nsertion, deletion, and reordering.
6. **Timing modification:** Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages could be a replay of some previous valid session, or

individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message (e.g., datagram) could be delayed or replayed.

7. **Source repudiation:** Denial of transmission of message by source.
8. **Destination repudiation:** Denial of receipt of message by destination.

Measures to deal with the first two attacks are in the realm of message confidentiality and are dealt with in Part One. Measures to deal with items (3) through (6) in the foregoing list are generally regarded as message authentication. Mechanisms for dealing specifically with item (7) come under the heading of digital signatures. Generally, a digital signature technique will also counter some or all of the attacks listed under items (3) through (6). Dealing with item (8) may require a combination of the use of digital signatures and a protocol designed to counter this attacks.

4.2. MESSAGE AUTHENTICATION FUNCTIONS:

Message authentication or digital signature mechanism has two levels of functionality. At the lower level, there must be some sort of function that produces an authenticator: a value to be used to authenticate a message. This lower-level function is then used as a primitive in a higher-level authentication protocol that enables a receiver to verify the authenticity of a message.

This section is concerned with the types of functions that may be used to produce an authenticator. These may be grouped into three classes.

- **Hash function:** A hash function is nothing but a mathematical function that can convert a numeric value into another numeric value that is compressed. The input to this hash function can be of any length but the output is always of fixed length. The values that a hash function returns are called the message digest or hash values. Message encryption: The cipher text of the entire message serves as its authenticator.
- **Message authentication code (MAC):** A message authentication code is a security code that the user of a computer has to type in order to access any account or portal. These codes are recognized by the system so that it can grant access to the right user. These codes help in maintaining information integrity. It also confirms the authenticity of the message
- **Message Encryption:** Message encryption by itself can provide a measure of authentication. The analysis differs for symmetric and public-key encryption schemes.

<u>Symmetric Encryption</u> Consider the straightforward use of symmetric encryption (Figure 4.1).

A message transmitted from source A to destination B is encrypted using a secret key shared by A and B. If no other party knows the key, then confidentiality is provided: No other party can recover the plaintext of the message.

In addition, B is assured that the message was generated by A. Why? The message must have come from A, because A is the only other party that possesses and therefore the only other party with the information necessary to construct cipher text that can be decrypted with. Furthermore, if is recovered, B knows that none of the bits of have been altered, because an opponent that does not know would not know how to alter bits in the ciphertext to produce the desired changes in the plaintext.

So we may say that symmetric encryption provides authentication as well as confidentiality. However, this flat statement needs to be qualified. Consider exactly what is happening at B. Given a decryption function D and a secret key, the destination will accept input and produce output. If is the ciphertext of a legitimate message produced by the corresponding encryption function, then is some plaintext message. Otherwise, will likely be a meaningless sequence of bits. There may need to be

some automated means of determining at B weather is legitimate plaintext and therefore must have come from A.

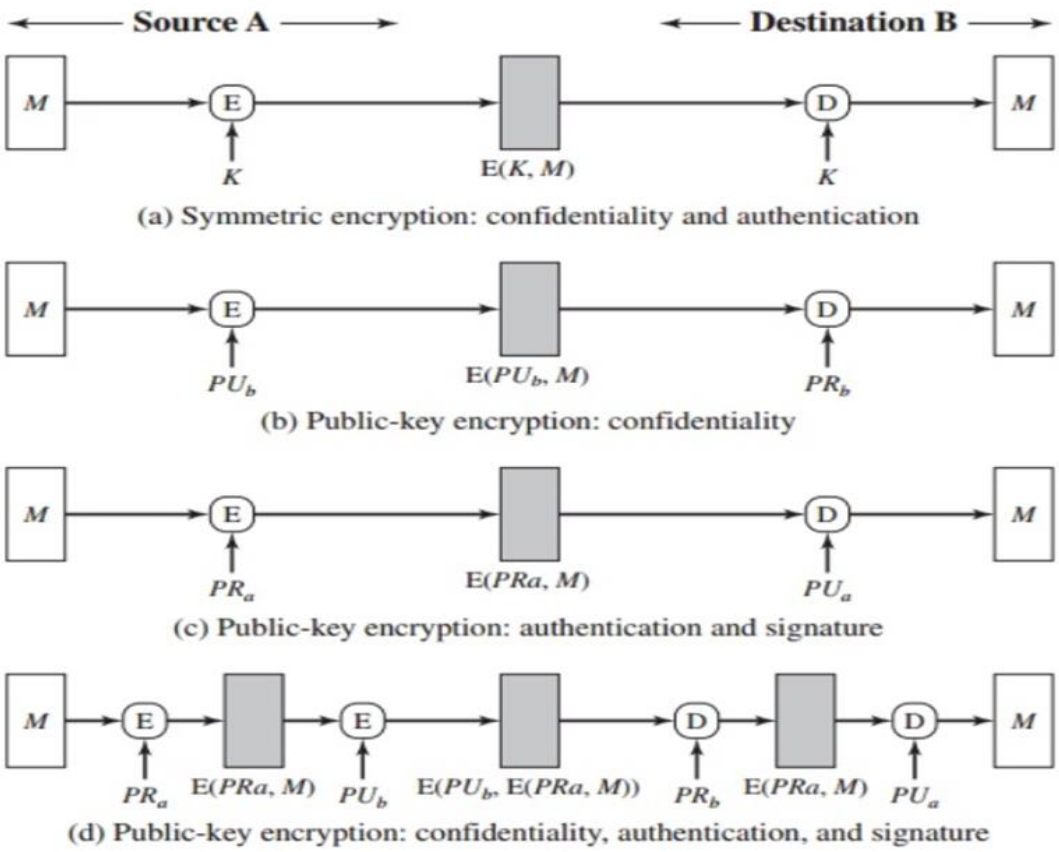

Figure 4.1: Basic use of Message Encryption

The implications of the line of reasoning in the preceding paragraph are profound from the point of view of authentication. Suppose the message can be any arbitrary bit pattern. In that case, there is no way to determine automatically, at the destination, whether an incoming message is the ciphertext of a legitimate message.

This conclusion is incontrovertible: If can be any bit pattern, then regardless of the value of, the value is bit pattern and therefore must be accepted as authentic plaintext.

For example, suppose that we are transmitting English language messages using a Caesar cipher with a shift of one (K 1).

A sends the following legitimate ciphertext:

```
nbsftfbupbutboeepftfbupbutboemjuumfmbnctfbuj
```

B decrypts to produce the following plaintext:

```
mareseatoatsanddoeseatoatsandlittlelambseativy
```

A simple frequency analysis confirms that this message has the profile of ordinary English. On the other hand, if an opponent generates the following random sequence of letters:

```
zuvrsoevgqxlzwigamdvnmhpmccxiuureosfbcebtqxsxq
```

this decrypts to

```
ytuqrndufpwkyvhfzlcumlgolbbwhttqdnreabdaspwrwp
```

which does not fit the profile of ordinary English.

It may be difficult to determine if incoming ciphertext decrypts to intelligible plaintext. If the plaintext is, say, a binary object file or digitized X-rays, determination of properly formed and therefore authentic plaintext may be difficult. Thus, an opponent could achieve a certain level of

disruption simply by issuing messages with random content purporting to come from a legitimate user.

One solution to this problem is to force the plaintext to have some structure that is easily recognized but that cannot be replicated without recourse to the encryption function. We could, for example, append an error-detecting code, also known as a frame check sequence (FCS) or checksum, to each message before encryption, as illustrated in Figure4.2, prepares a plaintext message and then provides this as input to a function F that produces an FCS.

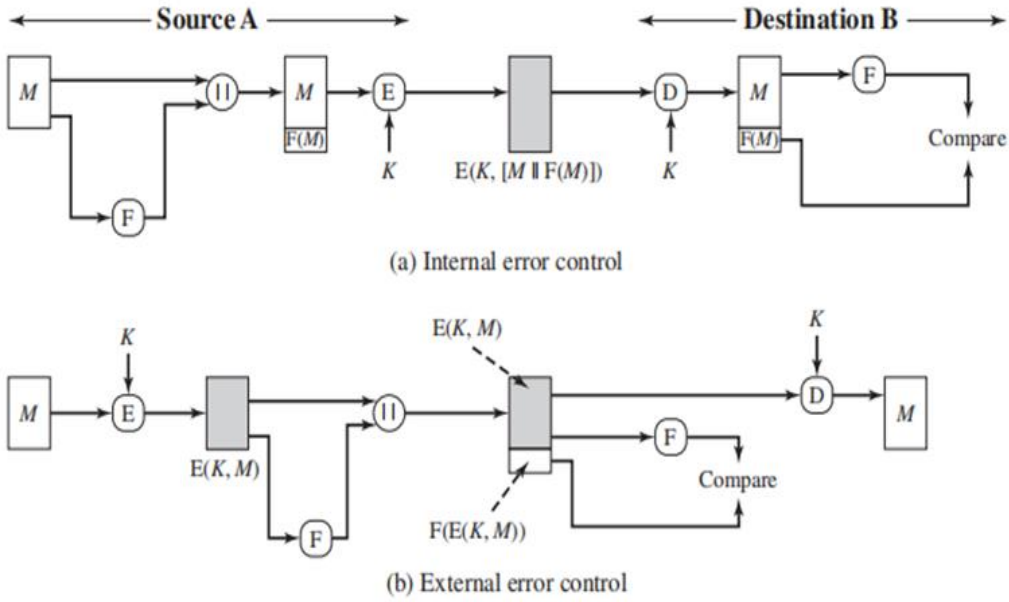

Figure 4.2: Internal and External Error Control

. The FCS is appended to and the entire block is then encrypted. At the destination, B M decrypts the incoming block and treats the results as a message with an appended FCS. B applies the same function F to attempt to reproduce the FCS. If the calculated FCS is equal to the incoming FCS, then the message is considered authentic. It is unlikely that any random sequence of bits would exhibit the desired relationship. Note that the order in which the FCS and encryption functions are performed is critical. The sequence illustrated in Figure4.2a is referred to in [DIFF79] as **internal error control**, which the authors contrast with **external error control** (Figure 4.2b). With internal error control, authentication is provided because an opponent would have difficulty generating ciphertext that, when decrypted, would have valid error control bits. If instead the FCS is the outer code, an opponent can construct messages with valid error-control codes. Although the opponent cannot know what the decrypted plaintext will be, he or she can still hope to create confusion and disrupt operations

<u>Public-Key Encryption</u> The straightforward use of public-key encryption (Figure 4.1.b) provides confidentiality but not authentication. The source (A) uses the public key PU_b ,of the destination (B) to encrypt. Because only B has the corresponding private key PR_b, only B can decrypt the message. This scheme provides no authentication, because any opponent could also use B's public key to encrypt a message and claim to be A.

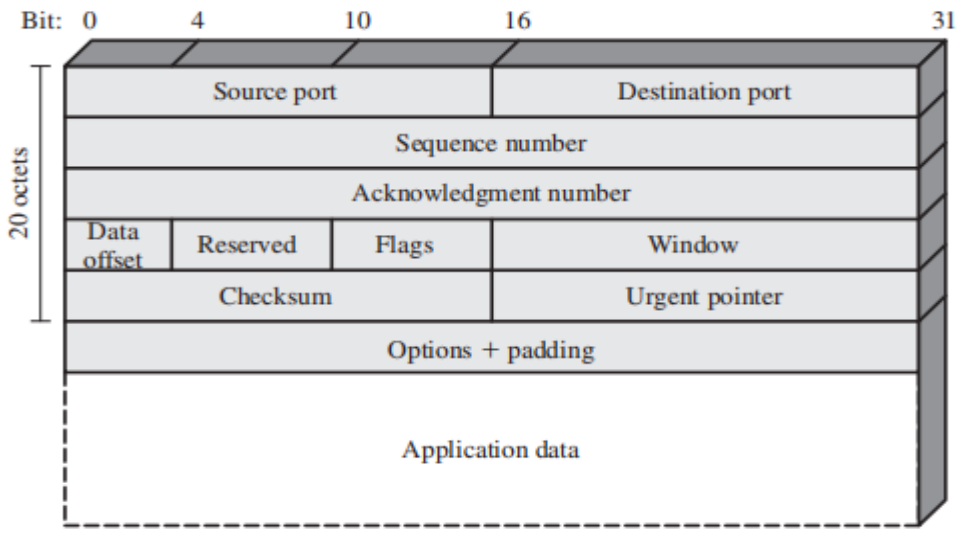

Figure 4.3: TCP segment

To provide authentication, A uses its private key to encrypt the message, and B uses A's public key to decrypt (Figure 4.1c). This provides authentication using the same type of reasoning as in the symmetric encryption case: The message must have come from A because A is the only party that possesses and therefore the only party with the information necessary to construct ciphertext that can be decrypted with. Again, the same reasoning as before applies: There must be some internal structure to the plaintext so that the receiver can distinguish between well-formed plaintext and random bits.

Assuming there is such structure, then the scheme of Figure 4.1c does provide authentication. It also provides what is known as digital signature.1 Only A could have constructed the ciphertext because only A possesses. Not even B, the recipient, could have constructed the ciphertext. Therefore, if B is in possession of the ciphertext, B has the means to prove that the message must have come from A. In effect, A has "signed" the message by using its private key to encrypt. Note that this scheme does not provide confidentiality. Anyone in possession of A's public key can decrypt the ciphertext.

To provide both confidentiality and authentication, A can encrypt first using its private key, which provides the digital signature, and then using B's public key, which provides confidentiality (Figure 4.1d). The is advantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

<u>Message Authentication Code:</u> An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as a cryptographic checksum or MAC, that is appended to the message. This technique assumes that two communicating parties, say A and B, share a common secret key. When A has a message to send to B, it calculates the MAC as a function of the message and the key:

$$MAC = MAC(K, M)$$

where

M= input message
C = MAC function
K= shared secret key
MAC = message authentication code

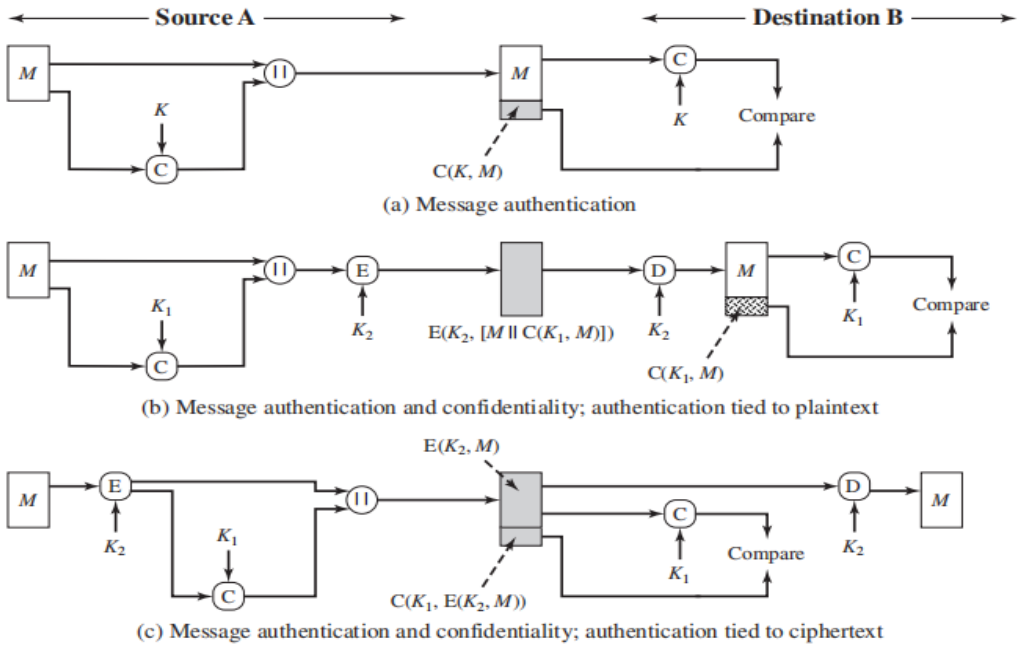

Figure 4.4: The basic use of Message Authentication Code

The message plus MAC are transmitted to the intended recipient. The recipient per forms the same calculation on the received message, using the same secret key, to generate a new MAC. The received MAC is compared to the calculated MAC (Figure 4.4a). If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

1. The receiver is assured that the message has not been altered. If an attacker alters the message but does not alter the MAC, then the receiver's calculation of the MAC will differ from the received MAC. Because the attacker is assumed not to know the secret key, the attacker cannot alter the MAC to correspond to the alterations in the message.
2. The receiver is assured that the message is from the alleged sender. Because no one else knows the secret key, no one else could prepare a message with a proper MAC.
3. If the message includes a sequence number (such as is used with HDLC, X.25, and TCP), then the receiver can be assured of the proper sequence because an attacker cannot successfully alter the sequence number.

A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption. In general, the MAC function is a many-to-one function. The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys. If an -bit MAC is used, then there are possible MACs, whereas there are possible messages with. Furthermore, with a -bit key, there are possible keys.

Because symmetric encryption will provide authentication and because it is widely used with readily available products, why not simply use this instead of a separate message authentication code? [DAVI89] suggests three situations in which a message authentication code is used.

1. There are a number of applications in which the same message is broadcast to a number of destinations.
   Examples are notification to users that the network is now unavailable or an alarm signal in a military control center. It is cheaper and more reliable to have only one destination responsible for monitoring authenticity. Thus, the message must be broadcast in plaintext with an associated message authentication code. The responsible system has the secret key

and performs authentication. If a violation occurs, the other destination systems are alerted by a general alarm.

2. Another possible scenario is an exchange in which one side has a heavy load and can't afford the time to decrypt all incoming messages. Authentication is carried out on a selective basis, messages being chosen at random for checking.
3. Authentication of a computer program in plaintext is an attractive service. The computer program can be executed without having to decrypt it every time, which would be wasteful of processor resources. However, if a message authentication code were attached to the program, it could be checked whenever assurance was required of the integrity of the program.

Three other rationales may be added.

4. For some applications, it may not be of concern to keep messages secret, but it is important to authenticate messages.
   An example is the Simple Network Management Protocol Version 3 (SNMPv3), which separates the functions of confidentiality and authentication. For this application, it is usually important for a managed system to authenticate incoming SNMP messages, particularly if the message contains a command to change parameters at the managed system. On the other hand, it may not be necessary to conceal the SNMP traffic.
5. Separation of authentication and confidentiality functions affords architectural flexibility. For example, it may be desired to perform authentication at the application level but to provide confidentiality at a lower level, such as the transport layer.
6. A user may wish to prolong the period of protection beyond the time of reception and yet allow processing of message contents. With message encryption, the protection is lost when the message is decrypted, so the message is protected against fraudulent modifications only in transit but not within the target system.

Finally, note that the MAC does not provide a digital signature, because both sender and receiver share the same key.

4.3. REQUIREMENTS FOR MESSAGE AUTHENTICATION CODES

A MAC, also known as a cryptographic checksum, is generated by a function C of the form

$$T = MAC(K, M)$$

Where

M is a variable-length message

K is a secret key shared only by sender and receiver

MAC(K, M) is the fixed-length authenticator, sometimes called a **tag**. The tag is appended to the message at the source at a time when the message is assumed or known to be correct. The receiver authenticates that message by re-computing the tag.

When an entire message is encrypted for confidentiality, using either symmetric or asymmetric encryption, the security of the scheme generally depends on the bit length of the key. Barring some weakness in the algorithm, the opponent must resort to a brute-force attack using all possible keys. On average, such an attack will require $2^{(k-1)}$attempts for a k-bit key. In particular, for a ciphertext only attack, the opponent, given ciphertext C, Pi=D(Ki,C) performs for all possible key K_i values until a P_i is produced that matches the form of acceptable plaintext.

In the case of a MAC, the considerations are entirely different. In general, the MAC function is a many-to-one function, due to the many-to-one nature of the function. Using brute-force methods, how

would an opponent attempt to discover a key? If confidentiality is not employed, the opponent has access to plaintext messages and their associated MACs. Suppose $k>n$; that is, suppose that the key size is greater than the MAC size.

Then, given a known M_1 and T_1 , with $T_1 = MAC(K, M_1)$,

The cryptanalyst can perform $T_i = MAC(K_i , M_1)$ for all possible key values .

At least one key is guaranteed to produce a match of $T_i = T_1$.

Note that a total of tags will be produced, but there are only $2^n<2^k$ different tag values. Thus, a number of keys will produce the correct tag and the opponent has no way of knowing which is the correct key. On average, a total of keys will produce a match. Thus, the opponent must iterate the attack way of knowing which the correct key is. On average, a total of keys $2^k /2^n = 2^{(k-n)}$ will produce a match. Thus, the opponent must iterate the attack.

- **Round 1**

  Given: $M_1, T_1 = \text{MAC}(K, M_1)$
  Compute $T_i = \text{MAC}(K_i, M_1)$ for all 2^k keys
  Number of matches $\approx 2^{(k-n)}$

- **Round 2**

  Given: $M_2, T_2 = \text{MAC}(K, M_2)$
  Compute $T_i = \text{MAC}(K_i, M_2)$ for the $2^{(k-n)}$ keys resulting from Round 1
  Number of matches $\approx 2^{(k-2\times n)}$

And so on. On average, α rounds will be needed if . For example, if an 80-bit key is used and the tab is 32 bits, then the first round will produce about 2^{48} possible keys. The second round will narrow the possible keys to about 2^{16} possibilities. The third round should produce only a single key, which must be the one used by the sender.

If the key length is less than or equal to the tag length, then it is likely that a first round will produce a single match. It is possible that more than one key will produce such a match, in which case the opponent would need to perform the same test on a new (message, tag) pair.

Thus, a brute-force attempt to discover the authentication key is no less effort and may be more effort than that required to discover a decryption key of the same length. However, other attacks that do not require the discovery of the key are possible.

A message authentication code (MAC) algorithm creates a message to be authenticated & a secret key that is popular only to the sender of the message and the receiver of the message & creates a MAC as an output. By using MAC, a receiver can test the integrity of the message & authenticity of the message i.e., whether it is appearing from the proper sender or not. MAC does not supports Non-Repudiation.

Three algorithms generally includes a MAC such as key generation algorithm, a signing algorithm and a verifying algorithm. The key generation algorithm selects a key at random.

The signing algorithm transmit a tag when likely the key and the message. The verifying algorithm can be used to check the authenticity of the message when given the key and tag. It will restore a message of accepted if the message and tag are authentic and same, but otherwise, it will restore a message of rejected.

There are some requirements for MACs are as follows −

- Message authentication codes (MACs) are generally used in digital funds transfers (EFTs) to support information integrity. They can validate that a message is authentic; that it actually does come, in other words, from the stated sender, and hasn't sustained some changes en-route.

- A verifier who also maintains the key can use it to recognize changes to the element of the message in question.
- Message authentication codes are generally needed to access any type of financial account. Banks, brokerage firms, trust organization, and some other deposit, investment, or insurance organization that provides online access can use these codes. They are an important component of financial cryptography.
    1. If an opponent observes M and C (K, M) and it must be computationally impossible for the opponent to make a message M' such that
    $$C(K, M') = C(K, M).$$
    2. C (K, M) should be consistently distributed in the sense that for randomly selected message, M and M', the probability that C (K, M) = C (K, M') is 2^{-n}, where n is the number of bits in the MAC.
    3. Let M' be same to some known transformation on M that is M'=f (M). For instance, f can include inverting one or more definite bits. In that case,
    $$Pr[C(K, M) = C(K, M')] = 2^{-n}.$$

SECURITY OF MACS

Just as with encryption algorithms and hash functions, we can group attacks on MACs into two categories: brute-force attacks and cryptanalysis.

Brute-Force Attacks:

A brute-force attack on a MAC is a more difficult undertaking than a brute-force attack on a hash function because it requires known message-tag pairs. Let us see why this is so. To attack a hash code, we can proceed in the following way. Given a fixed message x with n-bit hash code h = H(x), a brute-force method of finding a collision is to pick a random bit string and check if Hy) = H(x).The attacker can do this repeatedly off line. Whether an off-line attack can be used on a MAC algorithm depends on the relative size of the key and the tag.

Computation resistance: Given one or more text-MAC pairs, [xi, MAC(K, xi)] it is computationally infeasible to compute any text-MAC pair [x, MAC(K, x)] for any new input $x \neq x_i$.

In other words, the attacker would like to come up with the valid MAC code for a given message. There are two lines of attack possible: attack the key space and attack the MAC value. We examine each of these in turn.

Cryptanalysis:

As with encryption algorithms and hash functions, cryptanalytic attacks on MAC algorithms seek to exploit some property of the algorithm to perform some attack other than an exhaustive search. The way to measure the resistance of a MAC algorithm to cryptanalysis is to compare its strength to the effort required for a brute-force attack. That is, an ideal MAC algorithm will require a cryptanalytic effort greater than or equal to the brute-force effort.

There is much more variety in the structure of MACs than in hash functions, so it is difficult to generalize about the cryptanalysis of MACs.

4.4 **HMAC**

HMAC (**Hash-based Message Authentication Code**) is a type of message authentication code (MAC) that is acquired by executing a cryptographic hash function on the data that is to be authenticated and a secret shared key. Like any of the MACs, it is used for both data integrity and authentication.

- Unlike approaches based on signatures and asymmetric cryptography. Checking data integrity is necessary for the parties involved in communication. . The motivations for this interest are

  1. Cryptographic hash functions such as MD5 and SHA generally execute faster in software than symmetric block ciphers such as DES.

  2. Library code for cryptographic hash functions is widely available.

With the development of AES and the more widespread availability of code for encryption algorithms, these considerations are less significant, but hash-based MACs continue to be widely used.

- A hash function such as SHA HTTPS, SFTP, FTPS, and other transfer protocols use HMAC.
- The cryptographic hash function may be MD-5, SHA-1, or SHA-256.
- Digital signatures are nearly similar to HMACs i.e. they both employ a hash function and a shared key. The difference lies in the keys i.e. HMAC uses a symmetric key(same copy) while Signatures uses an asymmetric (two different keys).

HMAC Design Objectives RFC 2104 lists the following design objectives for HMAC.

- To use, without modifications, available hash functions. In particular, to use hash functions that perform well in software and for which code is freely and widely available.
- To allow for easy replace ability of the embedded hash function in case faster or more secure hash functions are found or required.
- To preserve the original performance of the hash function without incurring a significant degradation.
- To use and handle keys in a simple way.
- To have a well understood cryptographic analysis of the strength of the authentication mechanism based on reasonable assumptions about the embedded hash function.

HMAC Algorithm

Figure 4.5 illustrates the overall operation of HMAC. Define the following terms.

H = embedded hash function (e.g., MD5, SHA-1, RIPEMD-160)
IV = initial value input to hash function
M = message input to HMAC (including the padding specified in the embedded hash function)
Yi = *i* th block of *M*, 0 *i* (*L* – 1)

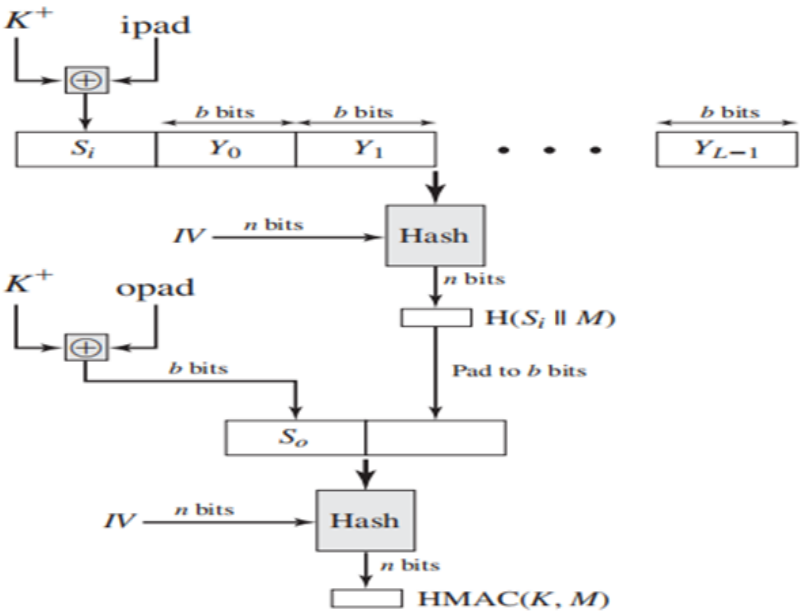

Figure 4.5: HMAC Structure

L = number of blocks in *M*
b = number of bits in a block
n = length of hash code produced by embedded hash function
K = secret key; recommended length is *n*; if key length is greater than *b*, the key is input to the hash function to produce an *n*-bit key.
K+ = K padded with zeros on the left so that the result is b bits in length
ipad = 00110110 (36 in hexadecimal) repeated b/8 times

opad = 01011100 (5C in hexadecimal) repeated b/8 times

Then HMAC can be expressed as

$$\text{HMAC}(K, M) = \text{H}[(K^+ \oplus \text{opad}) \| \text{H}[(K^+ \oplus \text{ipad}) \| M]]$$

- opad is the outer padding (a repeated 0x5c byte).
- ipad is the inner padding (a repeated 0x36 byte).
- ‖ denotes concatenation.
- ⊕ denotes XOR operation.

We can describe the algorithm as follows.

1. Append zeros to the left end of K to create a b-bit string K^+ (e.g., if K is of length 160 bits and $b = 512$, then K will be appended with 44 zeroes).
2. XOR (bitwise exclusive-OR) K^+ with ipad to produce the b-bit block S_i.
3. Append M to S_i.
4. Apply H to the stream generated in step 3.
5. XOR K^+ with opad to produce the b-bit block S_o.
6. Append the hash result from step 4 to S_o.
7. Apply H to the stream generated in step 6 and output the result.

Note that the XOR with ipad results in flipping one-half of the bits of K. Similarly, the XOR with opad results in flipping one-half of the bits of K, using a different set of bits. In effect, by passing S_i and S_0 through the compression function of the hash algorithm, we have pseudo randomly generated two keys from K.

HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for S_i, S_0, and the block produced from the inner hash).

A more efficient implementation is possible, as shown in Figure 4.6.

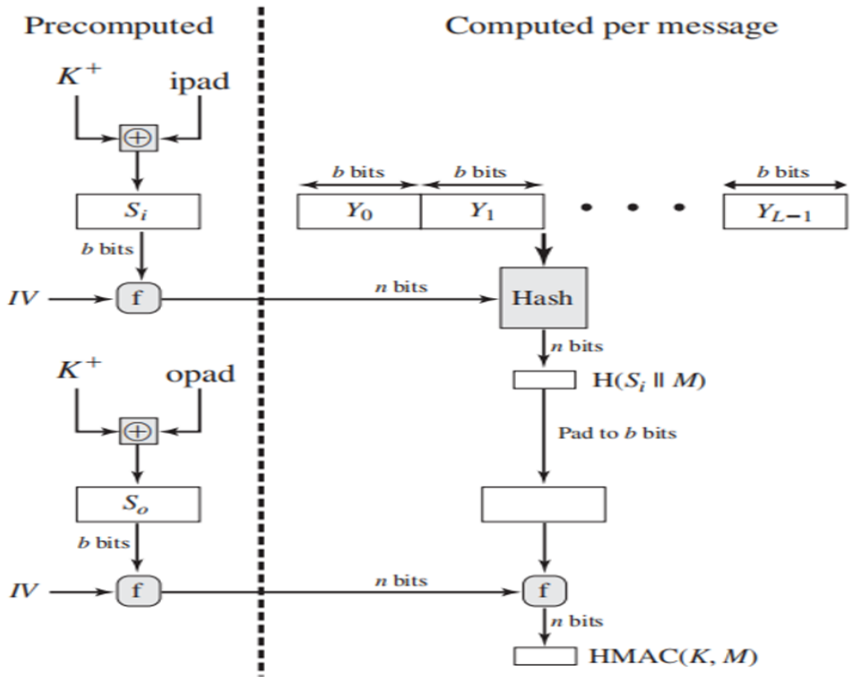

Figure 4.6 Efficient Implementation of HMAC

Two quantities are pre computed:

$$f(IV, (K^+ \oplus \text{ipad}))$$
$$f(IV, (K^+ \oplus \text{opad}))$$

Where is the compression function for the hash function, which takes as arguments a chaining variable of bits and a block of bits and produces a chaining variable of bits.These quantities only need to be computed initially and every time the key changes.In effect, the precomputed quantities substitute for the initial value (*IV*) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This more efficient implementation is especially worthwhile if most of the messages for which a MAC is computed are short.

Security of HMAC

The data is initially hashed by the client using a private key before being sent to the server as part of the request. The server then creates its own HMAC. This assures that the process is not vulnerable to attacks, which could result in crucial data being disclosed as subsequent MACs are generated. Additionally, once the procedure is completed, the delivered message becomes irreversible and resistant to hackers. Even if a malicious party attempts to steal the communication, they will be unable to determine its length or decrypt it because they do not have the decryption key.

The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message–tag pairs created with the same key. In essence, it is proved in [BELL96a] that for a given level of effort (time, message–tag pairs) on messages generated by a legitimate user and seen by the attacker, the probability of successful attack on HMAC is equivalent to one of the following attacks on the embedded hash function.

1. The attacker is able to compute an output of the compression function even with an *IV* that is random, secret, and unknown to the attacker.
2. The attacker finds collisions in the hash function even when the *IV* is random and secret.

.

Advantages of HMAC

- HMACs are ideal for high-performance systems like routers due to the use of hash functions which are calculated and verified quickly unlike the public key systems.
- Digital signatures are larger than HMACs, yet the HMACs provide comparably higher security.
- HMACs are used in administrations where public key systems are prohibited.

Disadvantages of HMAC

- HMACs uses shared key which may lead to non-repudiation. If either sender or receiver's key is compromised then it will be easy for attackers to create unauthorized messages.
- Securely managing and distributing secret keys can be challenging.
- Although unlikely, hash collisions (where two different messages produce the same hash) can occur.
- The security of HMAC depends on the length of the secret key. Short keys are more vulnerable to brute-force attacks.
- The security of HMAC relies on the strength of the chosen hash function (e.g., SHA-256). If the hash function is compromised, HMAC is also affected.

Applications of HMAC

- Verification of e-mail address during activation or creation of an account.
- Authentication of form data that is sent to the client browser and then submitted back.
- HMACs can be used for Internet of things (IoT) due to less cost.
- Whenever there is a need to reset the password, a link that can be used once is sent without adding a server state.
- It can take a message of any length and convert it into a fixed-length message digest. That is even if you got a long message, the message digest will be small and thus permits maximizing bandwidth.

4.6. <u>Digital Signatures</u>:

A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a signature. Typically the signature is formed by taking the hash of the message and encrypting the message with the creator's private key.
The main purposes of a digital signature are:

- source and integrity of the message.
- non-repudiation if the signer claims the signature is not authentic.

Digital signatures rely on asymmetric cryptography, also known as public key cryptography. An asymmetric key consists of a public/private key pair. The private key is used to create a signature, and the corresponding public key is used to verify the signature.

4.6.1. Digital Signatures properties:

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. Several forms of dispute between the two are possible.

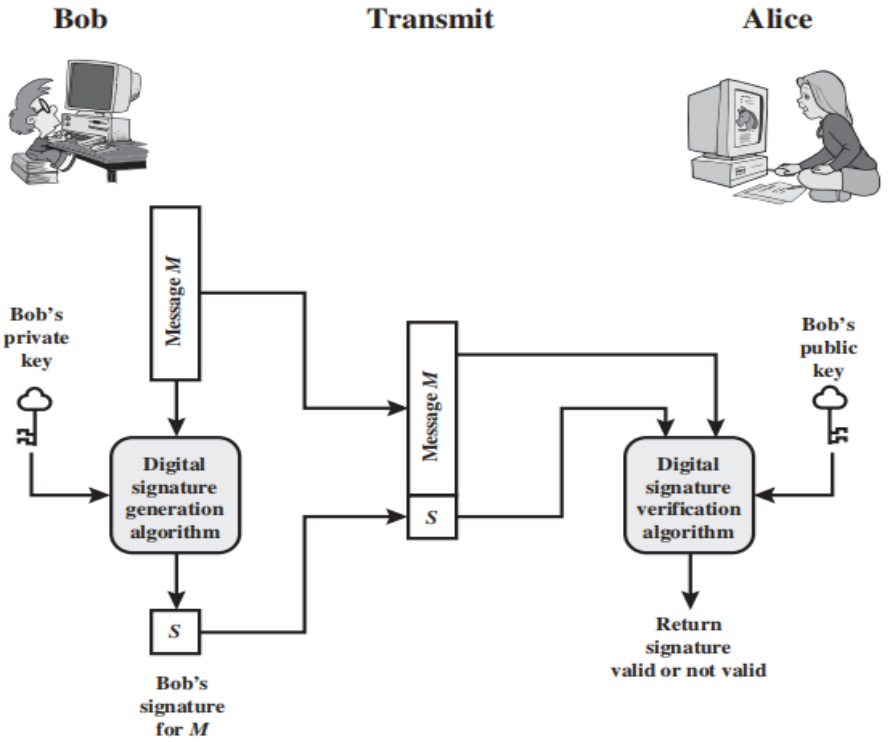

Figure 4.7: Generic Model of Digital Signature Process

For example, suppose that Bob sends an authenticated message to Alice, using one of the schemes of Figure 12.1. Consider the following disputes that could arise.

1. Alice may forge a different message and claim that it came from Bob. Alice would simply have to create a message and append an authentication code using the key that Bob and Alice share.
2. Bob can deny sending the message. Because it is possible for Alice to forge a message, there is no way to prove that Bob did in fact send the message.

Both scenarios are of legitimate concern. Here is an example of the first scenario: An electronic funds transfer takes place, and the receiver increases the amount of funds transferred and claims that the larger amount had arrived from the sender. An example of the second scenario is that an electronic mail message contains instructions to a stockbroker for a transaction that subsequently turns out badly. The sender pretends that the message was never sent.

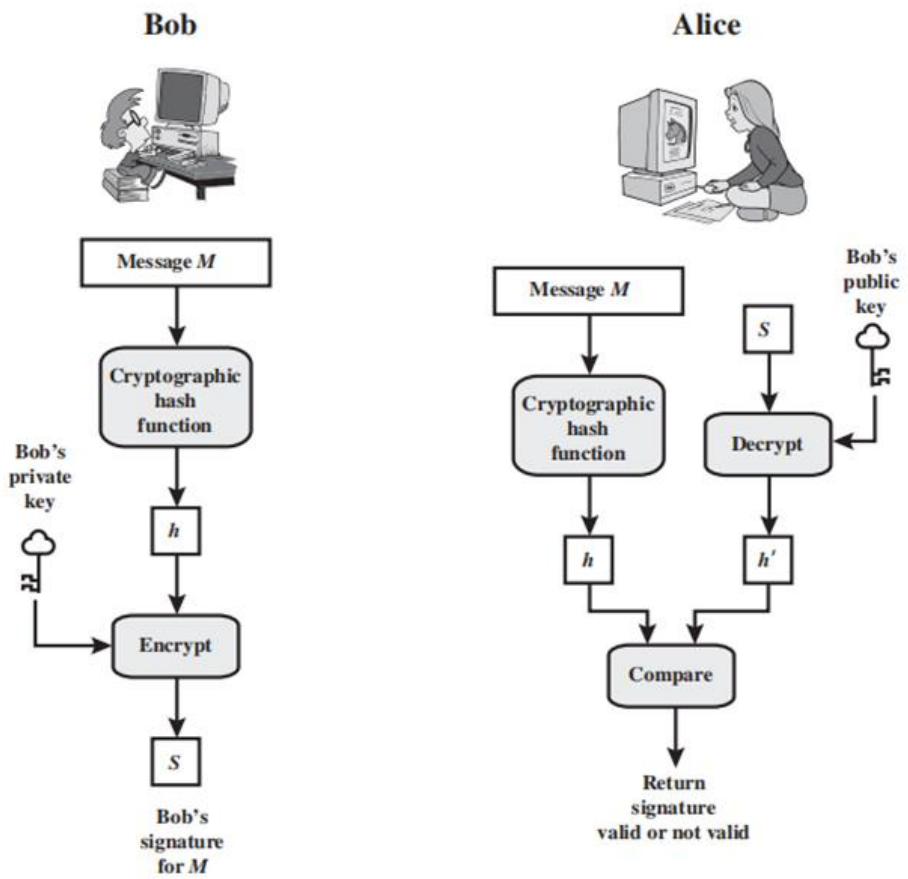

Figure 4.8: Simplified Depiction of Essential Elements of Digital Signature Process

In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature must have the following properties:

- It must verify the author and the date and time of the signature.
- It must authenticate the contents at the time of the signature.
- It must be verifiable by third parties, to resolve disputes.

Thus, the digital signature function includes the authentication function.

Attacks and Forgeries

[GOLD88] lists the following types of attacks, in order of increasing severity.
Here A denotes the user whose signature method is being attacked, and C denotes the attacker.

- **Key-only attack:** C only knows A's public key.
- **Known message attack:** C is given access to a set of messages and their signatures.
- **Generic chosen message attack:** C chooses a list of messages before attempting to breaks A's signature scheme, independent of A's public key. C then obtains from A valid signatures for the chosen messages. The attack is generic, because it does not depend on A's public key; the same attack is used against everyone.
- **Directed chosen message attack:** Similar to the generic attack, except that the list of messages to be signed is chosen after C knows A's public key but before any signatures are seen.
- **Adaptive chosen message attack:** C is allowed to use A as an "oracle." This means the A may request signatures of messages that depend on previously obtained message–signature pairs.

[GOLD88] then defines success at breaking a signature scheme as an outcome in which **C** can do any of the following with a non-negligible probability:

- **Total break:** C determines A's private key.
- **Universal forgery:** C finds an efficient signing algorithm that provides an equivalent way of constructing signatures on arbitrary messages.
- **Selective forgery:** C forges a signature for a particular message chosen by C.
- **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

4.6.2. Digital Signature Requirements

On the basis of the properties and attacks just discussed, we can formulate the following requirements for a digital signature.

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information unique to the sender to prevent both forgery and denial.
- It must be relatively easy to produce the digital signature.
- It must be relatively easy to recognize and verify the digital signature.
- It must be computationally infeasible to forge a digital signature, either by constructing a new message for an existing digital signature or by constructing a fraudulent digital signature for a given message.
- It must be practical to retain a copy of the digital signature in storage.

A secure hash function, embedded in a scheme such as that of *Figure 4.8* provides a basis for satisfying these requirements. However, care must be taken in the design of the details of the scheme.

Direct Digital Signature

The term **direct digital signature** refers to a digital signature scheme that involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source.

- Confidentiality can be provided by encrypting the entire message plus signature with a shared secret key (symmetric encryption). Note that it is important to perform the signature function first and then an outer confidentiality function.
- In case of dispute, some third party must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.
- The validity of the scheme just described depends on the security of the sender's private key. If a sender later wishes to deny sending a particular message, the sender can claim that the private key was lost or stolen and that someone else forged his or her signature.
- Administrative controls relating to the security of private keys can be employed to thwart or at least weaken this ploy, but the threat is still there, at least to some degree. One example is to require every signed message to include a timestamp (date and time) and to require prompt reporting of compromised keys to a central authority.

4.7. Digital Signature Standard.

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard FIPS 186, known as the Digital Signature Standard (DSS). The DSS makes use of the Secure Hash Algorithm (SHA) and presents a new digital signature technique, the **Digital Signature Algorithm (DSA)**.

The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback concerning the security of the scheme. There was a further minor revision in 1996. In 2000, an expanded version of the standard was issued as FIPS 186-2, subsequently updated to FIPS 186-3

in 2009. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm.

The DSS Approach

The DSS uses an algorithm that is designed to provide only the digital signature function. Unlike RSA, it cannot be used for encryption or key exchange. Nevertheless, it is a public-key technique. Figure 4.8 contrasts the DSS approach for generating digital signatures to that used with RSA.

- In the RSA approach, the message to be signed is input to a hash function that produces a secure hash code of fixed length. This hash code is then encrypted using the sender's private key to form the signature.
- Both the message and the signature are then transmitted. The recipient takes the message and produces a hash code. The recipient also decrypts the signature using the sender's public key.
- If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Because only the sender knows the private key, only the sender could have produced a valid signature.
- The DSS approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number generated ***k*** for this particular signature.
- The signature function also depends on the sender's private key PR_a and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key PU_G. The result is a signature consisting of two components, labeled s and r.

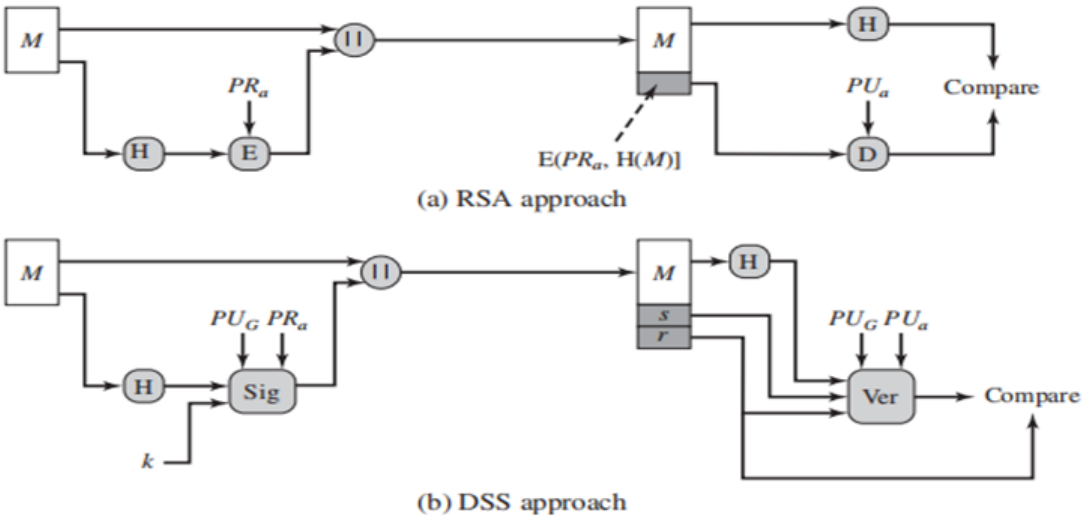

Figure 4.9: Two Approaches to Digital Signatures

Where

- M = Message or Plaintext
- H = Hash Function
- || = bundle the plantext and hash function (hash digest)
- E = Encryption Algorithm
- D = Decryption Algorithm
- PUa = Public key of sender
- PRa = Private key of sender
- Sig = Signature function
- Ver = Verification function
- PUG = Global public Key

At the receiving end, the hash code of the incoming message is generated. This plus the signature is input to a verification function. The verification function also depends on the global public key as well as the sender's public key , which is paired with the sender's private key. The output of the verification function is a value that is equal to the signature component if the signature is valid. The signature function is such that only the sender, with knowledge of the private key, could have produced the valid signature.

We turn now to the details of the algorithm.

The Digital Signature Algorithm:

The Digital Signature Algorithm (DSA) is a public-key technique (i.e., assymetric cryptography) and it is used to provide only the digital signature function, and it cannot be used for encryption or key exchange.

1. Global Public-Key Components

There are three parameters that are public and can be shared to a set of users.

- A prime number **p** is chosen with a length between 512 and 1024 bits such that q divides (p – 1). So, p is prime number where 2L-1 < p <2L for 512<= L<=1024 and L is a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits.
- Next, an N-bit prime number **q** is selected. So, q is prime divisor of (p – 1), where 2N-1 < q < 2N i.e., bit length of N bits.
- Finally, g is selected to be of the form h(p-1)/q mod p, where h is an integer between 1 and (p – 1) with the limitation that g must be greater than 1. So, g is = h(p – 1)/q mod p, where h is any integer with 1 < h < (p – 1) such that h(p-1)/q mod p > 1.

If a user has these numbers, then it can selects a private key and generates a public key.

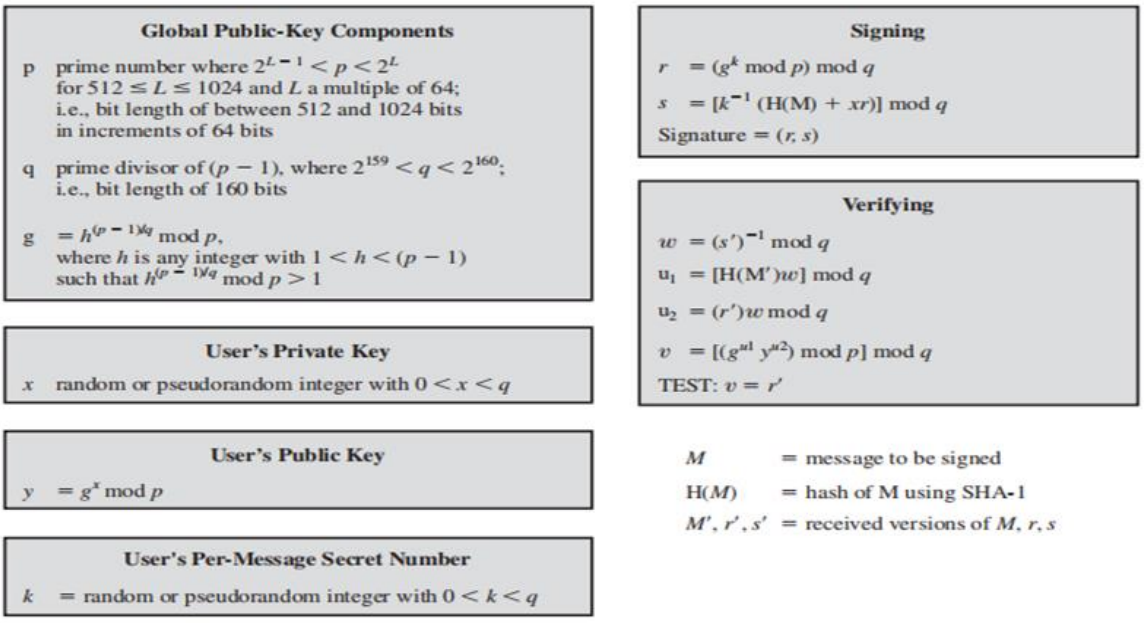

Figure 4.10: The Digital Signature Algorithm (DSA)

2. User's Private Key

The private key x should be chosen randomly or pseudorandomly and it must be a number from 1 to (q – 1), so x is random or pseudorandom integer with 0 < x < q.

3. User's Public Key

The public key is computed from the private key as y = gx mod p. The computation of y given x is simple. But, given the public key y, it is believed to be computationally infeasible to choose x, which is the discrete logarithm of y to the base g, mod p.

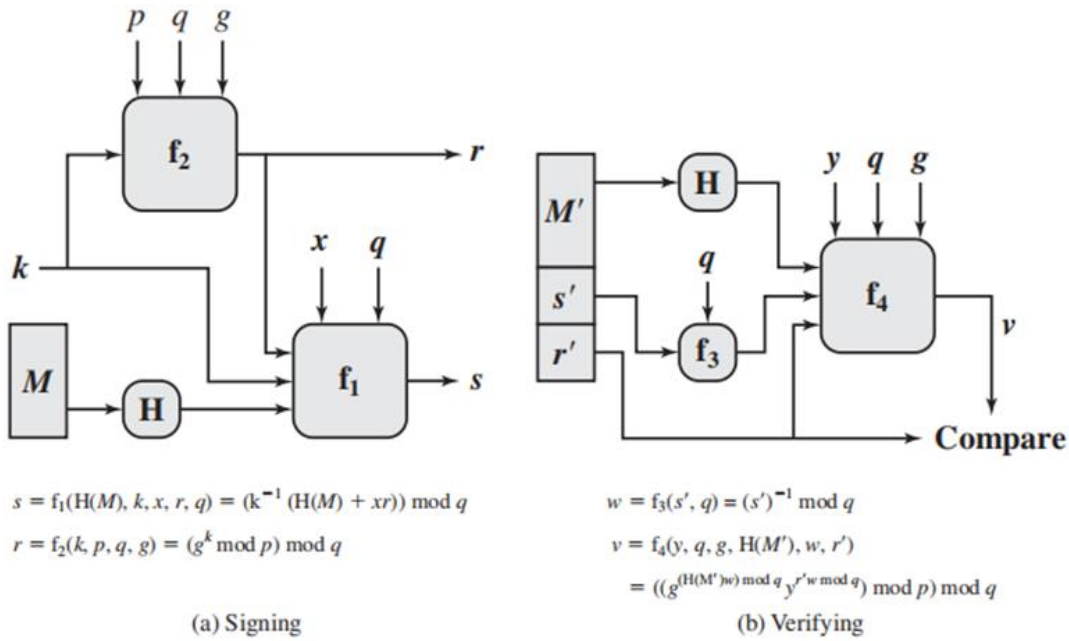

Figure 4.11 DSS Signing and Verifying

4. Signing

If a user want to develop a signature, a user needs to calculates two quantities, r and s, that are functions of the public key components (p, q, g), the hash code of the message H(M, the user's private key (x), and an integer k that must be generated randomly or pseudo randomly and be unique for each signing. k is generated randomly or pseudo randomly integer such that 0<k < q.

5. Verification

Let M, r′, and s′ be the received versions of M, r, and s, respectively.

Verification is performed using the formulas shown in below:

- w = (s′)-1 mod q
- u1 = [H(M′)w] mod q
- u2 = (r′)w mod q
- v = [(gu1 yu2) mod p] mod q

Services Provided By DSA

- **Message Authentication:** A secure digital signature scheme, like a secure conventional signature (one that cannot be easily copied) can provide message authentication (also referred to as data-origin authentication). Bob can easily confirm that the plaintext/message is sent by Alice as Alice's public key is used for verification and the Alice's public key would not verify the signature signed by Eve's private key
- **Message Integrity:** When we sign a whole message, its integrity remains intact because if the message changes, we won't get the same signature. Nowadays, digital signature methods use a special function called a hash function in both signing and verifying to ensure the message's integrity.
- **Nonrepudiation:** If Alice signs a message and later claims she didn't, can Bob provide evidence that she did? For example, if Alice instructs a bank (Bob) to transfer $10,000 to Ted's account and then denies sending the message, Bob needs to keep the signed message and use Alice's public key to recreate it. However, this approach may not work if Alice changes her keys or disputes the authenticity of the file. A solution is involving a trusted third party. This trusted party can authenticate messages and prevent Alice from denying them. In this setup, Alice sends her message along with her identity, Bob's identity, and her signature to the trusted center. The center verifies the message's authenticity and timestamps it before creating its own signature.

This process ensures that if Alice denies sending the message later, the center can provide evidence to settle the dispute. Encryption can also be added for confidentiality. Thus, nonrepudiation is achievable through a trusted party.

Advantages of DSA

- **Authentication:** At some point, digital signatures ensure strong identity authentication for the sender. The recipient can be sure that the message or document was signed by the purported signatory.
- **Integrity**: Digital signatures ensure the integrity of the content. If something is altered in the content after the signature is made, then it becomes invalid with respect to verifying the content.
- **Non-Repudiation:** A digital signature gives non-repudiation, meaning the sender cannot disclaim his creation of that document post factum. Most relevant in legal and contractual issues.
- **Efficiency:** Digital signatures make the process of signing electronic and automate it, giving way to fast online transactions free from the need of manual verification, paperwork, and a physical signature.
- **Security:** As long as the whole digital signing process is well organized, digital signatures may prove to be secure. Cryptographic public key cryptography and hashing algorithms prevent unauthorized parties from forging digital signatures.
- **World Acceptance:** Such a mechanism (digital signatures) to represent the conclusion of the related transaction in case of legal or contractual terms is known and widely accepted all over the world.
- **Timestamping:** Timestamping would also make another secure layer against replay attacks and against the freshness of the signature.
- **Cost Savings:** The digital signing process discontinues the need for transporting documents, thereby saving on costs to be done with printing, courier services, and manual handling.

4.8 E-mail Security:

Cryptography plays a vital role in securing email communications by ensuring confidentiality, integrity, authenticity, and non-repudiation. Techniques such as encryption, digital signatures, and secure transmission protocols are essential for protecting emails from unauthorized access, tampering, and spoofing. Effective key management and user education are critical for the successful implementation of email security in cryptographic systems.

1. Confidentiality:

- **Encryption of Email Content:**
  - **S/MIME (Secure/Multipurpose Internet Mail Extensions):** Uses X.509 certificates for public-key encryption. The email content is encrypted with a symmetric key, which is then encrypted with the recipient's public key. This ensures that only the intended recipient can decrypt and read the email.
  - **PGP (Pretty Good Privacy):** Utilizes a combination of symmetric key encryption and public-key cryptography to secure email content. The message is encrypted using a symmetric key, and this key is then encrypted with the recipient's public key.
  - **End-to-End Encryption:** In end-to-end encrypted emails, the email content is encrypted on the sender's device and only decrypted on the recipient's device. No intermediate server can read the email, enhancing confidentiality.

2. **Integrity:**

- **Hash Functions:**
  - **Message Digest (MD5, SHA-256):** Hash functions are used to create a fixed-size hash value (digest) from the email content. This hash value is unique to the message content, meaning that even a small change in the message will result in a significantly different hash.

- **HMAC (Hash-based Message Authentication Code):** An additional layer of security that uses a secret key along with the hash function to ensure that the message has not been altered during transit.

3. **Authentication:**

- **Digital Signatures:**
  - **S/MIME:** Allows the sender to sign an email using their private key. The recipient can verify the sender's identity by checking the signature against the sender's public key. This ensures that the email genuinely comes from the purported sender.
  - **PGP:** Similar to S/MIME, PGP uses digital signatures to authenticate the sender. The sender signs the email with their private key, and the recipient verifies the signature using the sender's public key.

4.8.1: PGP: PRETTY GOOD PRIVACY

PGP is a remarkable phenomenon. Largely the effort of a single person, Phil Zimmermann, PGP provides a confidentiality and authentication service that can be used for electronic mail and file storage applications. In essence, Zimmermann has done the following:

1. Selected the best available cryptographic algorithms as building blocks.
2. Integrated these algorithms into a general-purpose application that is independent of operating system and processor and that is based on a small set of easy-to-use commands.
3. Made the package and its documentation, including the source code, freely available via the Internet, bulletin boards, and commercial networks such as AOL (America On Line).
4. Entered into an agreement with a company (Viacrypt, now Network Associates) to provide a fully compatible, low-cost commercial version of PGP.

PGP has grown explosively and is now widely used. A number of reasons can be cited for this growth.

1. It is available free worldwide in versions that run on a variety of platforms, including Windows, UNIX, Macintosh, and many more. In addition, the commercial version satisfies users who want a product that comes with vendor support.
2. It is based on algorithms that have survived extensive public review and are considered extremely secure. Specifically, the package includes RSA, DSS, and Diffie-Hellman for public-key encryption; CAST-128, IDEA, and 3DES for symmetric encryption; and SHA-1 for hash coding.
3. It has a wide range of applicability, from corporations that wish to select and enforce a standardized scheme for encrypting files and messages to individuals who wish to communicate securely with others worldwide over the Internet and other networks.
4. It was not developed by, nor is it controlled by, any governmental or standards organization. For those with an instinctive distrust of "the establishment," this makes PGP attractive.
5. PGP is now on an Internet standards track (RFC 3156; MIME Security with OpenPGP). Nevertheless, PGP still has an aura of an antiestablishment endeavor.

We begin with an overall look at the operation of PGP. Next, we examine how cryptographic keys are created and stored. Then, we address the vital issue of public-key management.

<u>PGP Notation</u>

Most of the notation used in this chapter has been used before, but a few terms are new. It is perhaps best to summarize those at the beginning. The following symbols are used.

K_s = session key used in symmetric encryption scheme

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

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

EP = public-key encryption

DP = public-key decryption

EC = symmetric encryption

DC = symmetric decryption
H = hash function
|| = concatenation
Z = compression using ZIP algorithm
R64 = conversion to radix 64 ASCII format

The PGP documentation often uses the term secret key to refer to a key paired with a public key in a public-key encryption scheme. As was mentioned earlier, this practice risks confusion with a secret key used for symmetric encryption. Hence, we use the term private key instead.

Operational Description

The actual operation of PGP, as opposed to the management of keys, consists of four services: authentication, confidentiality, compression, and e-mail compatibility (Table) We examine each of these in turn.

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

Table: PGP Services

i. AUTHENTICATION Figure 18.1a illustrates the digital signature service provided by PGP. This is the digital signature scheme discussed in and illustrated in Figure 4.12. The sequence is as follows.

1. The sender creates a message.
2. SHA-1 is used to generate a 160-bit hash code of the message.
3. The hash code is encrypted with RSA using the sender's private key, and the result is prepended to the message.
4. The receiver uses RSA with the sender's public key to decrypt and recover the hash code.
5. The receiver generates a new hash code for the message and compares it with the decrypted hash code. If the two match, the message is accepted as authentic.

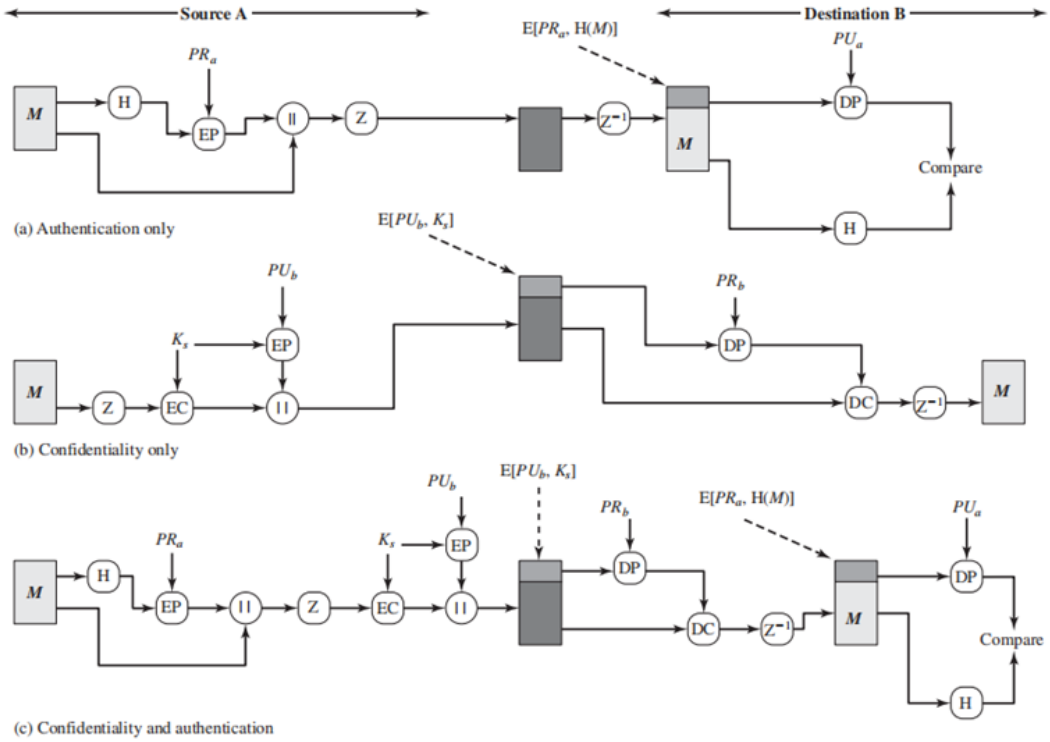

Figure 4.12: PGP Cryptographic Functions

ii. Confidentiality:

Another basic service provided by PGP is confidentiality, which is provided by encrypting messages to be transmitted or to be stored locally as files. In both cases, the symmetric encryption algorithm CAST-128 may be used. Alternatively, IDEA or 3DES may be used.The 64-bit cipher feedback (CFB) mode is used.

Figure 4.12(b) illustrates the sequence, which can be described as follows.
The sender generates a message and a random 128-bit number to be used as a session key for this message only.

1. The message is encrypted using CAST-128 (or IDEA or 3DES) with the session key.
2. The session key is encrypted with RSA using the recipient's public key and is prepended to the message.
3. The receiver uses RSA with its private key to decrypt and recover the session key.
4. The session key is used to decrypt the message.

Confidentiality and Authentication: As Figure 1c illustrates, both services may be used for the same message. First, a signature is generated for the plaintext message and prepended to the message. hen the plaintext message plus signature is encrypted using CAST-128 (or IDEA or 3DES), and the session key is encrypted using RSA.
This sequence is preferable to the opposite: encrypting the message and then generating a signature for the encrypted message. It is generally more convenient to store a signature with a plaintext version of a message.
Furthermore, for purposes of third-party verification, if the signature is performed first, a third party need not be concerned with the symmetric key when verifying the signature. In summary, when both services are used, the sender first signs the message with its own private key, then encrypts the message with a session key, and finally encrypts the session key with the recipient's public key.
iii. Compression: PGP compresses the message after applying the signature but before encryption. This has the benefit of saving space both for e-mail transmission and for file storage.

The placement of the compression algorithm, indicated by Z for compression and Z–1 for decompression in Figure 4.12, is critical.

1. The signature is generated before compression for two reasons:

a. It is preferable to sign an uncompressed message so that one can store only the uncompressed message together with the signature for future verification. If one signed a compressed document, then it would be necessary either to store a compressed version of the message for later verification or to recompress the message when verification is required.

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

2. Message encryption is applied after compression to strengthen cryptographic security.

Because the compressed message has less redundancy than the original plaintext, cryptanalysis is more difficult.

iv.E-mail compatibility :

- When PGP is used, at least part of the block to be transmitted is encrypted.
- If only the signature service is used, then the message digest is encrypted (with the sender's private key).
- If the confidentiality service is used, the message plus signature (if present) are encrypted (with a one-time symmetric key). Thus, part or all of the resulting block consists of a stream of arbitrary 8-bit octets.
- However, many electronic mail systems only permit the use of blocks consisting of ASCII text. To accommodate this restriction, PGP provides the service of converting the raw 8-bit binary stream to a stream of printable ASCII characters.
- The scheme used for this purpose is radix-64 conversion. Each group of three octets of binary data is mapped into four ASCII characters. This format also appends a CRC to detect transmission errors. The use of radix 64 expands a message by 33%. Fortunately, the session key and signature portions of the message are relatively compact, and the plaintext message has been compressed. In fact, the compression should be more than enough to compensate for the radix-64 expansion.
- Figure 2 shows the relationship among the four services so far discussed. On transmission (if it is required), a signature is generated using a hash code of the uncompressed plaintext.
- Then the plaintext (plus signature if present) is compressed. Next, if confidentiality is required, the block (compressed plaintext or compressed signature plus plaintext) is encrypted and prepended with the public-keyencrypted symmetric encryption key.
- Finally, the entire block is converted to radix-64 format. On reception, the incoming block is first converted back from radix-64 format to binary. Then, if the message is encrypted, the recipient recovers the session key and decrypts the message. The resulting block is then decompressed.
- If the message is signed, the recipient recovers the transmitted hash code and compares it to its own calculation of the hash code.

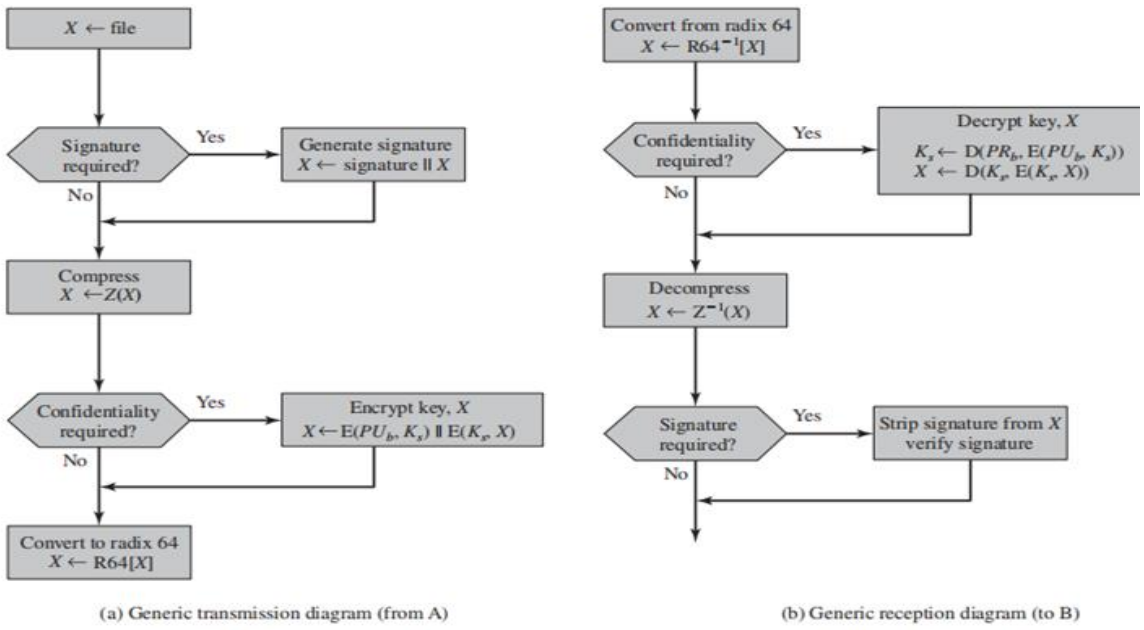

Figure 4.13: Transmission and Reception of PGP Messages

v. Segmentation and Reassembly:

- E-mail facilities often are restricted to a maximum message length. For example, many of the facilities accessible through the Internet impose a maximum length of 50,000 octets. Any message longer than that must be broken up into smaller segments, each of which is mailed separately.
- To accommodate this restriction, PGP automatically subdivides a message that is too large into segments that are small enough to send via e-mail. The segmentation is done after all of the other processing, including the radix-64 conversion.
- Thus, the session key component and signature component appear only once, at the beginning of the first segment.

The public-key ring can be indexed by either User ID or Key ID; we will see the need for both means of indexing later.

We are now in a position to show how these key rings are used in message transmission and reception. First consider message transmission and assume that the message is to be both signed and encrypted. The sending PGP entity performs the following steps.

1. Signing the message:

a. PGP retrieves the sender's private key from the private-key ring using your_userid as an index. If your_userid was not provided in the command, the first private key on the ring is retrieved.
b. PGP prompts the user for the passphrase to recover the unencrypted private key.
c. The signature component of the message is constructed.

2. Encrypting the message:

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

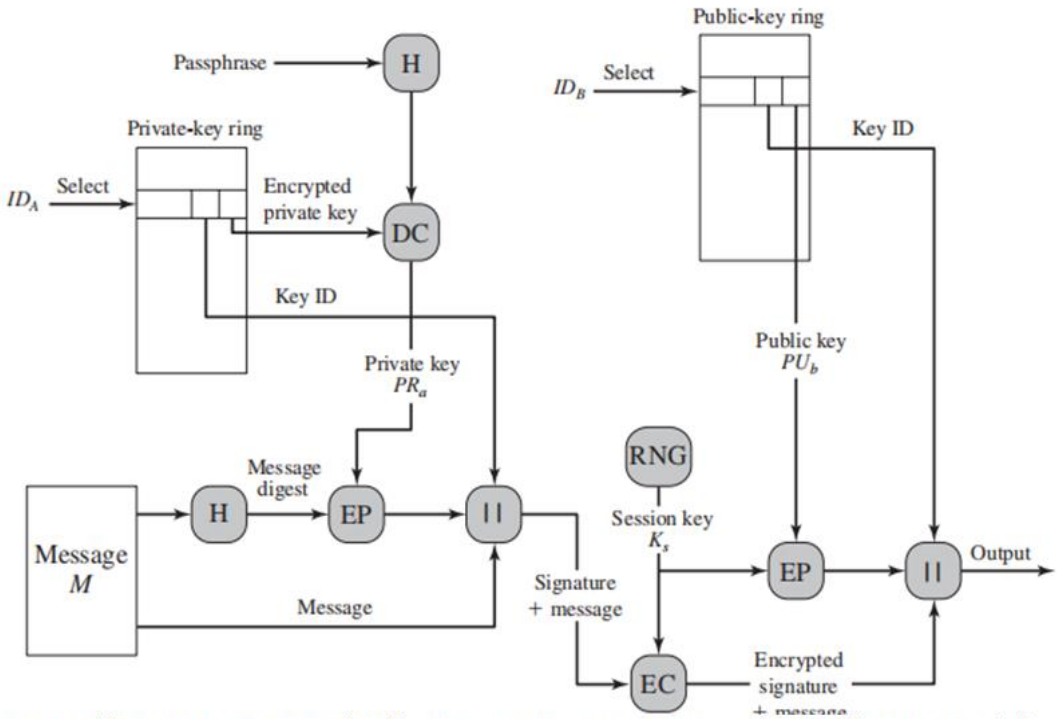

Figure 4.13(a) PGP Message Generation (from User A to User B: no compression or radix-64 conversion)

1. Decrypting the message:

a. PGP retrieves the receiver's private key from the private-key ring using the Key ID field in the session key component of the message as an index.
b. PGP prompts the user for the passphrase to recover the unencrypted private key.
c. PGP then recovers the session key and decrypts the message.

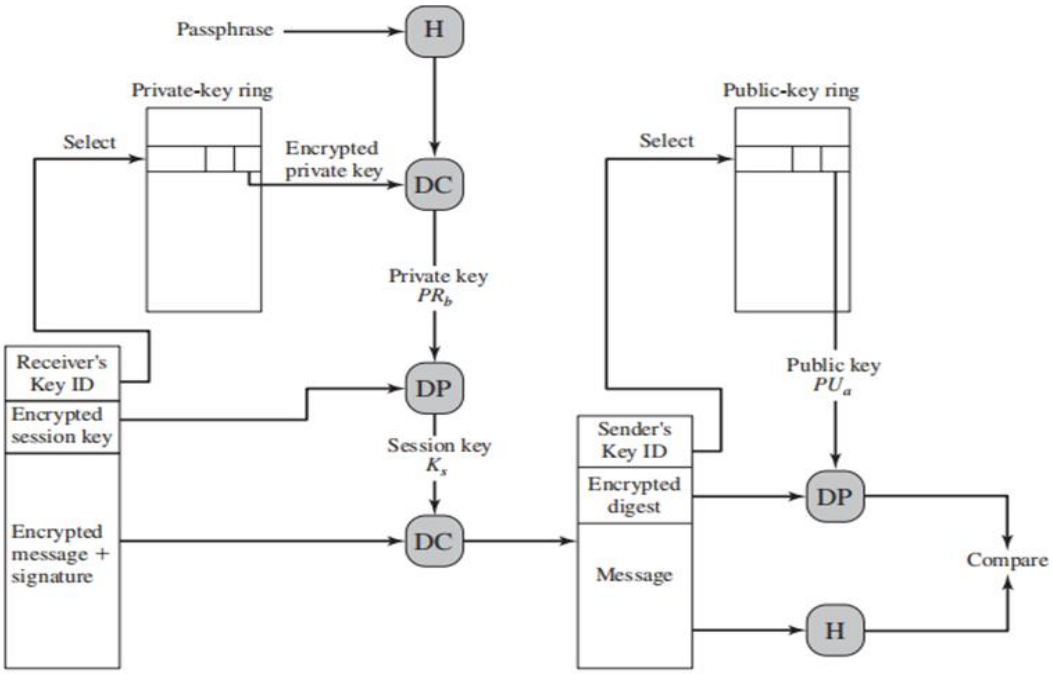

Figure 4.13(b) PGP Message Reception (from User A to User B: no compression or radix-64 conversion)

2. Authenticating the message:

a. PGP retrieves the sender's public key from the public-key ring using the Key ID field in the signature key component of the message as an index.
b. PGP recovers the transmitted message digest.

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

Advantages and Disadvantages of PGP:

PGP offers robust security and a decentralized trust model, making it a powerful tool for protecting the confidentiality, integrity, and authenticity of email communications. However, its complexity, key management challenges, and compatibility issues can hinder its usability, particularly for less technical users. Despite these challenges, PGP remains a valuable tool for those who require strong privacy and security in their communications.

Advantages of PGP:

1. **Strong Security**
    - Encryption Strength
    - End-to-End Encryption
2. **Digital Signatures**
    - Data Integrity and Confidentiality
3. **Cross-Platform Compatibility**
4. **Web of Trust**
    - Decentralized Trust Model
    - Resilience Against Centralized Attacks
5. **No Central Authority Required**
6. **User Control**
    - Key Management
    - Self-Management
7. **Versatility**
    - Email Security
    - File Encryption

***Disadvantages of PGP*:**

1. **Complexity**
    - User Difficulty
    - Potential for User Errors
2. **Key Management Issues**
    - Key Distribution
    - Key Revocation
3. **No Centralized Trust**
    - Web of Trust Complications
    - Trust Scalability
4. **Compatibility Issues**
    - Inconsistent Support
    - Lack of Uniform Standards
5. **Security Risks**
    - Key Mismanagement
    - Vulnerabilities
6. **No Perfect Forward Secrecy (PFS)**
    - Exposure of Past Messages
7. **Lack of Support for Some Modern Features**
    - No Native Support for Mobile Devices
    - Limited Metadata Protection

4.8.2: **S/MIME:**

Secure/Multipurpose Internet Mail Extension (S/MIME) is a security enhancement to the MIME Internet e-mail format standard based on technology from RSA Data Security. Although

both PGP and S/MIME are on an IETF standards track, it appears likely that S/MIME will emerge as the industry standard for commercial and organizational use, while PGP will remain the choice for personal e-mail security for many users. S/MIME is defined in a number of documents—most importantly RFCs 3370, 3850, 3851, and 3852.

To understand S/MIME, we need first to have a general understanding of the underlying e-mail format that it uses, namely MIME. But to understand the significance of MIME, we need to go back to the traditional e-mail format standard, RFC 822, which is still in common use. The most recent version of this format specification is RFC 5322 (*Internet Message Format*). Accordingly, this section first provides an introduction to these two earlier standards and then moves on to a discussion of S/MIME.

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

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

Another field that is commonly found in RFC 5322 headers is *Message-ID*. This field contains a unique identifier associated with this message

Multipurpose Internet Mail Extensions

Multipurpose Internet Mail Extension (MIME) is an extension to the RFC 5322 framework that is intended to address some of the problems and limitations of the use of Simple Mail Transfer Protocol (SMTP), defined in RFC 821, or some other mail transfer protocol and RFC 5322 for electronic mail. [PARZ06] lists the following limitations of the SMTP/5322 scheme.

1. SMTP cannot transmit executable files or other binary objects. A number of schemes are in use for converting binary files into a text form that can be used by SMTP mail systems, including the popular UNIX UUencode/UUdecode scheme. However, none of these is a standard or even a *de facto* standard.
2. SMTP cannot transmit text data that includes national language characters, because these are represented by 8-bit codes with values of 128 decimal or higher, and SMTP is limited to 7-bit ASCII.
3. SMTP servers may reject mail message over a certain size.
4. SMTP gateways that translate between ASCII and the character code EBCDIC do not use a consistent set of mappings, resulting in translation problems.
5. SMTP gateways to X.400 electronic mail networks cannot handle non textual data included in X.400 messages.
6. Some SMTP implementations do not adhere completely to the SMTP standards defined in RFC 821. Common problems include:
    - Deletion, addition, or reordering of carriage return and linefeed
    - Truncating or wrapping lines longer than 76 characters
    - Removal of trailing white space (tab and space characters)
    - Padding of lines in a message to the same length

- Conversion of tab characters into multiple space characters

MIME is intended to resolve these problems in a manner that is compatible with existing RFC 5322 implementations. The specification is provided in RFCs 2045 through 2049.

OVERVIEW The MIME specification includes the following elements.

1. Five new message header fields are defined, which may be included in an RFC 5322 header. These fields provide information about the body of the message.
2. A number of content formats are defined, thus standardizing representations that support multimedia electronic mail.
3. Transfer encodings are defined that enable the conversion of any content format into a form that is protected from alteration by the mail system.

The five header fields defined in MIME are

a. **MIME-Version:** Must have the parameter value 1.0. This field indicates that the message conforms to RFCs 2045 and 2046.
b. **Content-Type:** Describes the data contained in the body with sufficient detail that the receiving user agent can pick an appropriate agent or mechanism to represent the data to the user or otherwise deal with the data in an appropriate manner.
c. **Content-Transfer-Encoding:** Indicates the type of transformation that has been used to represent the body of the message in a way that is acceptable for mail transport.
d. **Content-ID:** Used to identify MIME entities uniquely in multiple contexts.
e. **Content-Description:** A text description of the object with the body; this is useful when the object is not readable (e.g., audio data).

MIME CONTENT TYPES: The bulk of the MIME specification is concerned with the definition of a variety of content types. This reflects the need to provide standardized ways of dealing with a wide variety of information representations in a multimedia environment.

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

Table2.MIME Content Types

MIME TRANSFER ENCODINGS:

The other major component of the MIME specification, in addition to content type specification, is a definition of transfer encodings for message bodies. The objective is to provide reliable delivery across the largest range of environments.

The MIME standard defines two methods of encoding data. The Content Transfer-Encoding field can actually take on six values, as listed in Table 3. However, three of these values (7bit, 8bit, and binary) indicate that no encoding has been done but provide some information about the nature of the data. For SMTP transfer, it is safe to use the 7bit form. The 8bit and binary forms may be usable in other mail transport contexts. Another Content-Transfer-Encoding value is x-token, which indicates that some other encoding scheme is used for which a name is to be supplied. This could be a vendor-specific or application-specific scheme. The two actual encoding schemes defined are quoted-printable and base64. Two schemes are defined to provide a choice between a transfer technique that is essentially human readable and one that is safe for all types of data in a way that is reasonably compact.

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

Table3: MIME Transfer Encodings

The **quoted-printable** transfer encoding is useful when the data consists largely of octets that correspond to printable ASCII characters. In essence, it represents nonsafe characters by the hexadecimal representation of their code and introduces reversible (soft) line breaks to limit message lines to 76 characters.

The **base64 transfer encoding,** also known as radix-64 encoding, is a common one for encoding arbitrary binary data in such a way as to be invulnerable to the processing by mail-transport programs.

CANONICAL FORM: An important concept in MIME and S/MIME is that of canonical form. Canonical form is a format, appropriate to the content type, that is standardized for use between systems. This is in contrast to native form, which is a format that may be peculiar to a particular system. Table 18.5, from RFC 2049, should help clarify this matter.

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

Table4: Native and Canonical Form

S/MIME Functionality

In terms of general functionality, S/MIME is very similar to PGP. Both offer the ability to sign and/or encrypt messages. In this subsection, we briefly summarize S/MIME capability.We then look in more detail at this capability by examining message formats and message preparation.

FUNCTIONS S/MIME provides the following functions:

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

CRYPTOGRAPHIC ALGORITHMS Table 18.6 summarizes the cryptographic algorithms used in S/MIME. S/MIME uses the following terminology taken from RFC 2119 (*Key Words for use in RFCs to Indicate Requirement Levels*) to specify the requirement level:

- **MUST:** The definition is an absolute requirement of the specification. An implementation must include this feature or function to be in conformance with the specification.
- **SHOULD:** There may exists valid reasons in particular circumstances to ignore this feature or function, but it is recommended that an implementation include the feature or function.

S/MIME incorporates three public-key algorithms. The Digital Signature Standard (DSS) described in Chapter 13 is the preferred algorithm for digital signature. S/MIME lists Diffie-Hellman as the preferred algorithm for encrypting session keys; in fact, S/MIME uses a variant of Diffie-Hellman that does provide encryption/decryption, known as ElGamal.

Function	Requirement
Create a message digest to be used in forming a digital signature.	MUST support SHA-1. Receiver SHOULD support MD5 for backward compatibility.
Encrypt message digest to form a digital signature.	Sending and receiving agents MUST support DSS. Sending agents SHOULD support RSA encryption. Receiving agents SHOULD support verification of RSA signatures with key sizes 512 bits to 1024 bits.
Encrypt session key for transmission with a message.	Sending and receiving agents SHOULD support Diffie-Hellman. Sending and receiving agents MUST support RSA encryption with key sizes 512 bits to 1024 bits.
Encrypt message for transmission with a one-time session key.	Sending and receiving agents MUST support encryption with tripleDES. Sending agents SHOULD support encryption with AES. Sending agents SHOULD support encryption with RC2/40.
Create a message authentication code.	Receiving agents MUST support HMAC with SHA-1. Sending agents SHOULD support HMAC with SHA-1.

Table5: Cryptographic Algorithms Used in S/MIME

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

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

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

If a message is to be sent to multiple recipients and a common encryption algorithm cannot be selected for all, then the sending agent will need to send two messages. However, in that case, it is important to note that the security of the message is made vulnerable by the transmission of one copy with lower security.

S/MIME Messages

S/MIME makes use of a number of new MIME content types, which are shown in Table 6. All of the new application types use the designation PKCS. This refers to a set of public-key

cryptography specifications issued by RSA Laboratories and made available for the S/MIME effort.

Type	Subtype	smime Parameter	Description
Multipart	Signed		A clear-signed message in two parts: one is the message and the other is the signature.
Application	pkcs7-mime	signedData	A signed S/MIME entity.
	pkcs7-mime	envelopedData	An encrypted S/MIME entity.
	pkcs7-mime	degenerate signedData	An entity containing only public-key certificates.
	pkcs7-mime	CompressedData	A compressed S/MIME entity.
	pkcs7-signature	signedData	The content type of the signature subpart of a multipart/signed message.

Table 6: S/MIME Content Types

We examine each of these in turn after first looking at the general procedures for S/MIME message preparation.

The steps for preparing an envelopedData MIME entity are

1. Generate a pseudorandom session key for a particular symmetric encryption algorithm (RC2/40 or triple DES).
2. For each recipient, encrypt the session key with the recipient's public RSA key.
3. For each recipient, prepare a block known as RecipientInfo that contains an identifier of the recipient's public-key certificate,4 an identifier of the algorithm used to encrypt the session key, and the encrypted session key.
4. Encrypt the message content with the session key.

S/MIME Certificate Processing

S/MIME uses public-key certificates that conform to version 3 of X.509. The key-management scheme used by S/MIME is in some ways a hybrid between a strict X.509 certification hierarchy and PGP's web of trust. As with the PGP model, S/MIME managers and/or users must configure each client with a list of trusted keys and with certificate revocation lists. That is, the responsibility is local for maintaining the certificates needed to verify incoming signatures and to encrypt outgoing messages. On the other hand, the certificates are signed by certification authorities.

USER AGENT ROLE

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

- **Key generation:** The user of some related administrative utility (e.g., one associated with LAN management) MUST be capable of generating separate Diffie-Hellman and DSS key pairs and SHOULD be capable of generating RSA key pairs. Each key pair MUST be generated from a good source of non-deterministic random input and be protected in a secure fashion. A user agent SHOULD generate RSA key pairs with a length in the range of 768 to 1024 bits and MUST NOT generate a length of less than 512 bits.
- **Registration:** A user's public key must be registered with a certification authority in order to receive an X.509 public-key certificate.
- **Certificate storage and retrieval:** A user requires access to a local list of certificates in order to verify incoming signatures and to encrypt outgoing messages. Such a list could be maintained by the user or by some local administrative entity on behalf of a number of users.

An example process flow of sending an S/MIME email includes the following steps:

Step 1: Creating a Message: The sender composes an email in their S/MIME-compatible email client.

Step 2: Signing the Message: The sender's email client signs the message using the sender's private key.

Step 3: Encrypting the Message: The message is encrypted using a symmetric key. This symmetric key is then encrypted with the recipient's public key.

Step 4: Sending the Message: The signed and encrypted message is sent to the recipient.

Step 5: Receiving the Message: The recipient's email client decrypts the message using the recipient's private key and verifies the signature using the sender's public key.

Advantages of S/MIME

- **Security:** S/MIME offers strong encryption and authentication, making email communications secure against eavesdropping and tampering.
- **Interoperability:** As a standard, S/MIME is supported by most major email clients and can be used across different platforms.
- **Ease of Use:** Once set up, S/MIME operates transparently, encrypting and signing emails automatically.

Disadvantages of S/MIME

- **Certificate Management Complexity:** Managing certificates can be complex and requires careful attention to security practices.
- **Cost:** Certificates from trusted CAs can be costly, especially for organizations requiring multiple certificates.
- **Compatibility Issues:** Not all email clients fully support S/MIME, and there can be challenges with compatibility, especially when interacting with non-S/MIME users.

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. What is the primary purpose of a Message Authentication Code (MAC)?
    a) Encrypt the message content
    b) Provide message integrity and authentication
    c) Compress the message
    d) Generate a unique key for encryption

2. Which of the following is NOT a requirement for a secure Message Authentication Code (MAC)?
    a) It should be computationally infeasible to find two different messages with the same MAC
    b) It should be computationally infeasible to deduce the key from the MAC
    c) It should be reversible to recover the original message
    d) The MAC should be uniformly distributed for different messages

3. Which of the following functions is commonly used to create a MAC?
    a) Hash function
    b) Symmetric encryption
    c) Public key encryption
    d) Compression function

4. In HMAC, what is the key mixed with before applying the hash function?
    a) Public key
    b) Secret key
    c) Digital signature
    d) Message digest

5. What makes HMAC more secure than a simple MAC?
    a) HMAC uses symmetric key encryption
    b) HMAC combines the hash function with a secret key
    c) HMAC uses public key encryption
    d) HMAC compresses the message before hashing

6. Which of the following is a key characteristic of a secure MAC function?
    a) The output length is variable
    b) The MAC is indistinguishable from random data
    c) The MAC is easy to reverse engineer
    d) The MAC is independent of the message

7. Which of the following is NOT a typical requirement of message authentication?
    a) Message integrity
    b) Non-repudiation
    c) Confidentiality
    d) Message authentication

8. Which of the following is an advantage of using HMAC over other MAC methods?
    a) HMAC uses public key encryption
    b) HMAC provides built-in key management
    c) HMAC can be based on any cryptographic hash function
    d) HMAC is faster than all other MACs
9. Which property does a digital signature NOT provide?
    a) Confidentiality
    b) Authentication
    c) Integrity
    d) Non-repudiation
10. What is the main difference between a MAC and a digital signature?
    a) MAC uses symmetric key, while digital signature uses asymmetric key
    b) MAC provides non-repudiation, while digital signature does not
    c) MAC provides encryption, while digital signature does not
    d) MAC is used for email security, while digital signature is not
11. Which algorithm is specified in the Digital Signature Standard (DSS)?
    a) RSA
    b) DSA
    c) SHA-256
    d) AES
12. Which of the following is a requirement for a secure digital signature?
    a) It should be computable only by the receiver
    b) It should be easy to forge by anyone
    c) It should be verifiable by the receiver using the sender's public key
    d) It should remain the same even if the message changes
13. What is the purpose of the hash function in the digital signature process?
    a) To encrypt the message
    b) To generate a unique value that represents the message content
    c) To compress the message
    d) To generate the public/private key pair
14. Which of the following is true regarding digital signatures?
    a) The signature is generated using the receiver's private key
    b) The signature is verified using the sender's public key
    c) The signature is always the same size as the message
    d) The signature must be kept confidential from the receiver
15. In the Digital Signature Algorithm (DSA), which component is used to generate the signature?
    a) Public key
    b) Private key
    c) Hash value
    d) Symmetric key
16. Which of the following is NOT a property of a digital signature?
    a) It is encrypted with the signer's public key
    b) It is easy to verify by anyone with the correct public key

c) It ensures that the signer cannot deny signing the document
d) It is unique to both the message and the signer

17. What does PGP stand for in the context of email security?
    a) Pretty Good Privacy
    b) Public Good Protection
    c) Personal Guard Protocol
    d) Privacy Guaranteed Protection
18. Which of the following is a key feature of PGP?
    a) It relies solely on symmetric key cryptography
    b) It uses a combination of symmetric and asymmetric cryptography
    c) It does not provide message authentication
    d) It requires a Certificate Authority to function
19. What is the main difference between PGP and S/MIME?
    a) PGP uses a Web of Trust model, while S/MIME uses a hierarchical trust model
    b) PGP requires a Certificate Authority, while S/MIME does not
    c) PGP is used only for file encryption, while S/MIME is used for emails
    d) PGP does not support digital signatures, while S/MIME does
20. Which of the following is NOT a function provided by S/MIME?
    a) Confidentiality
    b) Authentication
    c) Compression
    d) Message integrity
21. In PGP, what is used to encrypt the message?
    a) The recipient's private key
    b) A symmetric session key
    c) The sender’s public key
    d) The recipient’s public key
22. What role does the Certificate Authority (CA) play in S/MIME?
    a) To encrypt email messages
    b) To manage public keys and issue digital certificates
    c) To decrypt messages on behalf of the recipient
    d) To compress email messages for faster transmission
23. Which of the following is a feature of PGP that differs from S/MIME?
    a) Use of symmetric encryption
    b) Web of Trust model for key verification
    c) Reliance on Certificate Authorities
    d) Support for multimedia content
24. What is the purpose of a digital certificate in S/MIME?
    a) To encrypt email content
    b) To bind a public key to an identity
    c) To generate a symmetric key
    d) To verify the email content
25. Which of the following is a common use case for S/MIME?
    a) Encrypting files on a disk
    b) Securing email communications
    c) Encrypting network traffic
    d) Generating random keys for cryptographic algorithms

SECTION-B

B. Descriptive Questions

1. What is PGP? Examine how authentication, confidentiality and compatibility can be maintained in PGP? Justify why compression is required before encryption and after digital signature?
2. What are the functionalities provided by Secure MIME (S/MIME)? Explain.
3. Give the structure of PGP message generation. Explain with a neat diagram.
4. Briefly explain about the different message authentication functions with neat diagrams.
5. Describe the overall operation of HMAC with algorithm with neat diagram.
6. What is Message Authentication Code? Explain its functions and basic uses.
7. Illustrate the steps involved in Signature generation and verification functions of DSS.
8. Explain in detail about Elementary SHA-512 Operation (single round) with neat diagram.
9. How a hash functions can be used for “intrusion detection”, “virus detection” and creating a “one-way password file”?
10. Illustrate how a hash code is used to provide digital signature.
11. Describe the steps in message digest generation in Secure Hash Algorithm.
12. Explain the difference between a MAC and a digital signature.
13. Describe how HMAC enhances the security of a basic MAC.
14. Summarize the role of a hash function in the process of creating a digital signature.
15. What is the primary purpose of the Web of Trust model in PGP?
16. Explain how S/MIME ensures the integrity of an email message.
17. Analyze the security advantages and disadvantages of using MACs compared to digital signatures.
18. Compare and contrast the trust models used in PGP and S/MIME.
19. Examine how the choice of hash function impacts the security of HMAC.
20. Differentiate between the encryption mechanisms used in PGP and S/MIME.
21. Critique the effectiveness of digital certificates in ensuring email security in S/MIME.

****