

## UNIT IV NETWORK SECURITY

OSI Security Architecture – Attacks – Security Services and Mechanisms – Encryption – Advanced Encryption Standard – Public Key Cryptosystems – RSA Algorithm – Hash Functions – Secure Hash Algorithm – Digital Signature Algorithm.

### Network Security

Information is an asset that has a value like any other asset. Information needs to be secured from attacks. To be secured, information needs to be hidden from unauthorized access (confidentiality), protected from unauthorized change (integrity), and available to an authorized entity when it is needed (availability).

### Security Goals

#### 1) Confidentiality

Confidentiality is probably the most common aspect of information security. We need to protect our confidential information. An organization needs to guard against those malicious actions that endanger the confidentiality of its information. Confidentiality not only applies to the storage of information; it also applies to the transmission of information.

#### 2) Integrity

Information needs to be changed constantly. In a bank, when a customer deposits or withdraws money, the balance of her account needs to be changed. Integrity means that changes need to be done only by authorized entities and through authorized mechanisms.

#### 3) Availability

The third component of information security is availability. The information created and stored by an organization needs to be available to authorized entities. Information is useless if it is not available.

### OSI Security Architecture

Security Architecture for OSI, defines a systematic approach of organizing the task of providing security. The OSI security architecture focuses on security attacks, mechanisms, and services.

**Security attack:** Any action that compromises the security of information owned by an organization.

**Security mechanism:** A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack.

**Security service:** A processing or communication service that enhances the security of the data processing systems and the information transfers of an organization. The

services are intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.

## 1. Security Attacks:

Security attacks are passive attacks and active attacks. A passive attack attempts to learn or make use of information from the system but does not affect system resources. An active attack attempts to alter system resources or affect their operation.

### a) Passive attacks:

Passive attacks are in the nature of eavesdropping on, or monitoring of, transmissions. The goal of the opponent is to obtain information that is being transmitted. Two types of passive attacks are the release of message contents and traffic analysis.

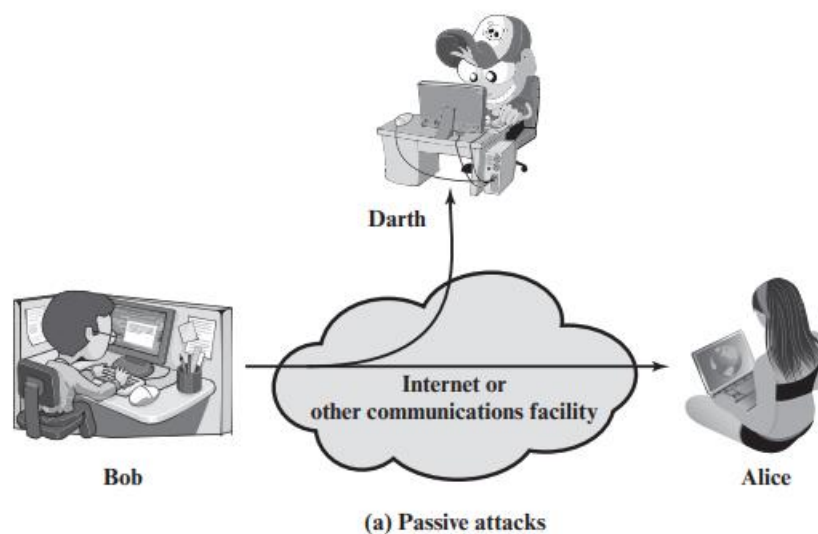
#### The release of message contents.

A telephone conversation, an electronic mail message, and a transferred file may contain sensitive or confidential information. We would like to prevent an opponent from learning the contents of these transmissions.

#### Traffic Analysis

If we had encryption protection in place, an opponent might still be able to observe the pattern of these messages. The opponent could determine the location and identity of communicating hosts and could observe the frequency and length of messages being exchanged. This information might be useful in guessing the nature of the communication that was taking place.

Passive attacks are very difficult to detect, because they do not involve any alteration of the data. Typically, the message traffic is sent and received in an apparently normal fashion, and neither the sender nor receiver is aware that a third party has read the messages or observed the traffic pattern. Thus, the emphasis in dealing with passive attacks is on prevention rather than detection.



**b) Active attacks:**

Active attacks involve some modification of the data stream or the creation of a false stream and can be subdivided into four categories: masquerade, replay, modification of messages, and denial of service.

**Masquerading**

Masquerading, or spoofing, happens when the attacker impersonates somebody else. For example, an attacker might steal the bank card and PIN of a bank customer and pretend that she is that customer. Sometimes the attacker pretends instead to be the receiver entity. For example, a user tries to contact a bank, but another site pretends that it is the bank and obtains some information from the user.

**Replaying**

In replaying, the attacker obtains a copy of a message sent by a user and later tries to replay it. For example, a person sends a request to her bank to ask for payment to the attacker, who has done a job for her. The attacker intercepts the message and sends it again to receive another payment from the bank.

**Modification**

After intercepting or accessing information, the attacker modifies the information to make it beneficial to herself. For example, a customer sends a message to a bank to initiate some transaction. The attacker intercepts the message and changes the type of transaction to benefit herself.

**Denial of Service**

Denial of service (DoS) is a very common attack. It may slow down or totally interrupt the service of a system. The attacker can use several strategies to achieve this. She might send so many bogus requests to a server that the server crashes because of the heavy load. The attacker might intercept and delete a server's response to a client, making the client believe that the server is not responding. The attacker may also intercept requests from the clients, causing the clients to send requests many times and overload the system.

Active attacks present the opposite characteristics of passive attacks. Whereas passive attacks are difficult to detect, measures are available to prevent their success. On the other hand, it is quite difficult to prevent active attacks absolutely because of the wide variety of potential physical, software, and network vulnerabilities. Instead, the goal is to detect active attacks and to recover from any disruption or delays caused by them.

**2. Security Services**

These services are divided into five categories and fourteen specific services.

### **a) Authentication**

The authentication service is concerned with assuring that a communication is authentic. In the case of a single message, such as a warning or alarm signal, the function of the authentication service is to assure the recipient that the message is from the source that it claims to be from.

#### **Peer entity authentication:**

Used in association with a logical connection to provide confidence in the identity of the entities connected.

#### **Data-Origin Authentication**

In a connectionless transfer, provides assurance that the source of received data is as claimed.

### **b) Access Control**

In the context of network security, access control is the ability to limit and control the access to host systems and applications via communications links. To achieve this, each entity trying to gain access must first be identified, or authenticated, so that access rights can be tailored to the individual.

### **c) Data Confidentiality**

The protection of data from unauthorized disclosure.

**Connection Confidentiality** The protection of all user data on a connection.

**Connectionless Confidentiality** The protection of all user data in a single data block.

**Selective-Field Confidentiality** The confidentiality of selected fields within the user data on a connection or in a single data block.

**Traffic-Flow Confidentiality** The protection of the information that might be derived from observation of traffic flows.

### **d) Data Integrity**

The assurance that data received are exactly as sent by an authorized entity (i.e., contain no modification, insertion, deletion, or replay).

**Connection Integrity with Recovery** Provides for the integrity of all user data on a connection and detects any modification, insertion, deletion, or replay of any data within an entire data sequence, with recovery attempted.

**Connection Integrity without Recovery** As above but provides only detection without recovery.

**Selective-Field Connection Integrity** Provides for the integrity of selected fields within the user data of a data block transferred over a connection and takes the form of determination of whether the selected fields have been modified, inserted, deleted, or replayed.

**Connectionless Integrity** Provides for the integrity of a single connectionless data block and may take the form of detection of data modification. Additionally, a limited form of replay detection may be provided.

**Selective-Field Connectionless Integrity** Provides for the integrity of selected fields within a single connectionless data block; takes the form of determination of whether the selected fields have been modified.

### e) **Nonrepudiation**

Provides protection against denial by one of the entities involved in a communication of having participated in all or part of the communication.  
**Nonrepudiation, Origin** Proof that the message was sent by the specified party.

**Nonrepudiation, Destination** Proof that the message was received by the specified party.

### 3. **Security Mechanisms:**

The mechanisms are divided into those that are implemented in a specific protocol layer, such as TCP or an application-layer protocol, and those that are not specific to any particular protocol layer or security service.

**1. Specific Security Mechanisms** May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.

**Encipherment** The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.

**Digital Signature** Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).

**Access Control** A variety of mechanisms that enforce access rights to resources.

**Data Integrity** A variety of mechanisms used to assure the integrity of a data unit or stream of data units.

**Authentication Exchange** A mechanism intended to ensure the identity of an entity by means of information exchange.

**Traffic Padding** The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.

**Routing Control** Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.

**Notarization** The use of a trusted third party to assure certain properties of a data exchange.

#### **2. Pervasive Security Mechanisms**

Mechanisms that are not specific to any particular OSI security service or protocol layer.

**Trusted Functionality** That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).

**Security Label** The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.

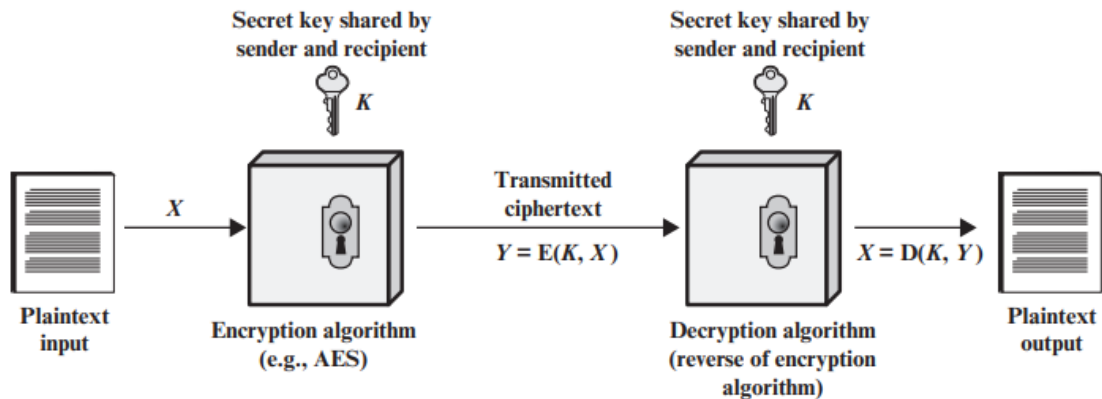
**Event Detection** Detection of security-relevant events.

**Security Audit Trail** Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.

**Security Recovery** Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

## Encryption

### Symmetric Cipher model



A symmetric encryption scheme has five ingredients:

**Plaintext:** This is the original intelligible message or data that is fed into the algorithm as input. **Encryption algorithm:** The encryption algorithm performs various substitutions and transformations on the plaintext.

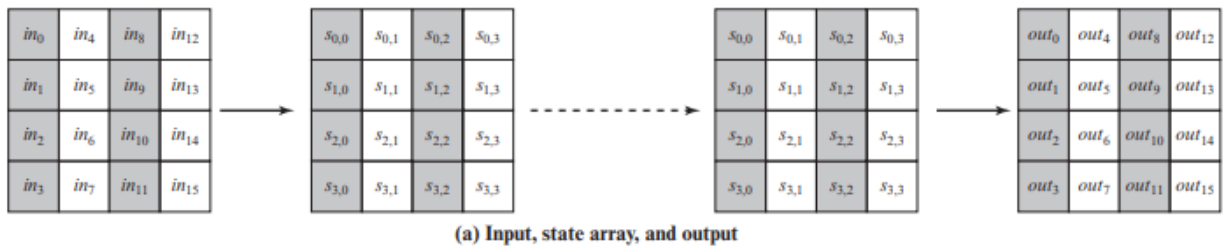
**Secret key:** The secret key is also input to the encryption algorithm. The key is a value independent of the plaintext and of the algorithm. The algorithm will produce a different output depending on the specific key being used at the time.

**Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the secret key.

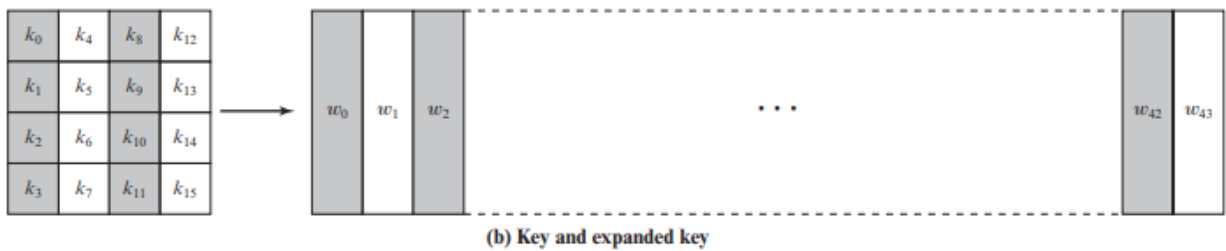
**Decryption algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

### Advanced Encryption Standard

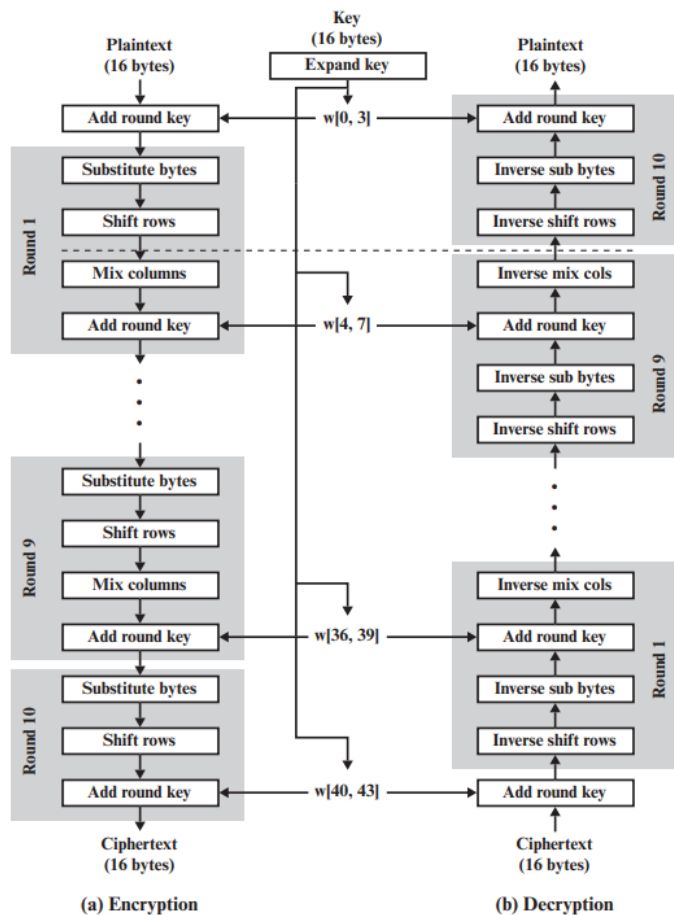
- The Advanced Encryption Standard (AES) was published by the National Institute of Standards and Technology (NIST) in 2001. AES is a symmetric block cipher.
- The cipher takes a plaintext block size of 128 bits, or 16 bytes. The key length can be 16, 24, or 32 bytes (128, 192, or 256 bits). The algorithm is referred to as AES-128, AES-192, or AES-256, depending on the key length.
- The input to the encryption and decryption algorithms is a single 128-bit block. This block is depicted as a  $4 \times 4$  square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix.
- The ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on.



- Similarly, the key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words. The key that is provided as input is expanded into an array of forty-four 32-bit words,  $w[i]$ . The first four bytes of the expanded key, which form a word, occupy the first column of the  $w$  matrix.



## AES Structure



Four different stages are used, one of permutation and three of substitution:

- Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block.
- ShiftRows: A simple permutation.
- MixColumns: A substitution that makes use of arithmetic over  $GF(2^8)$ .
- AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key.

- For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages.
- Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
- Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that  $A \oplus B \oplus B = A$ .

## Substitute Bytes Transformation

### FORWARD AND INVERSE TRANSFORMATIONS

- The forward substitute byte transformation, called Sub Bytes, is a simple table lookup. AES defines a  $16 * 16$  matrix of byte values, called an S-box that contains a permutation of all possible 256 8-bit values.

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76	
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0	
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15	
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75	
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84	
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF	
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8	
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2	
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73	
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB	
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79	
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08	
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A	
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E	
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF	
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16	

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB	
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB	
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E	
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25	
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92	
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84	
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06	
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B	
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73	
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E	
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B	
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4	
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F	
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF	
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61	
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D	

(b) Inverse S-box

- Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used

as a column value. Similarly for reverse substitute byte transformation inverse s box is used.

- For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

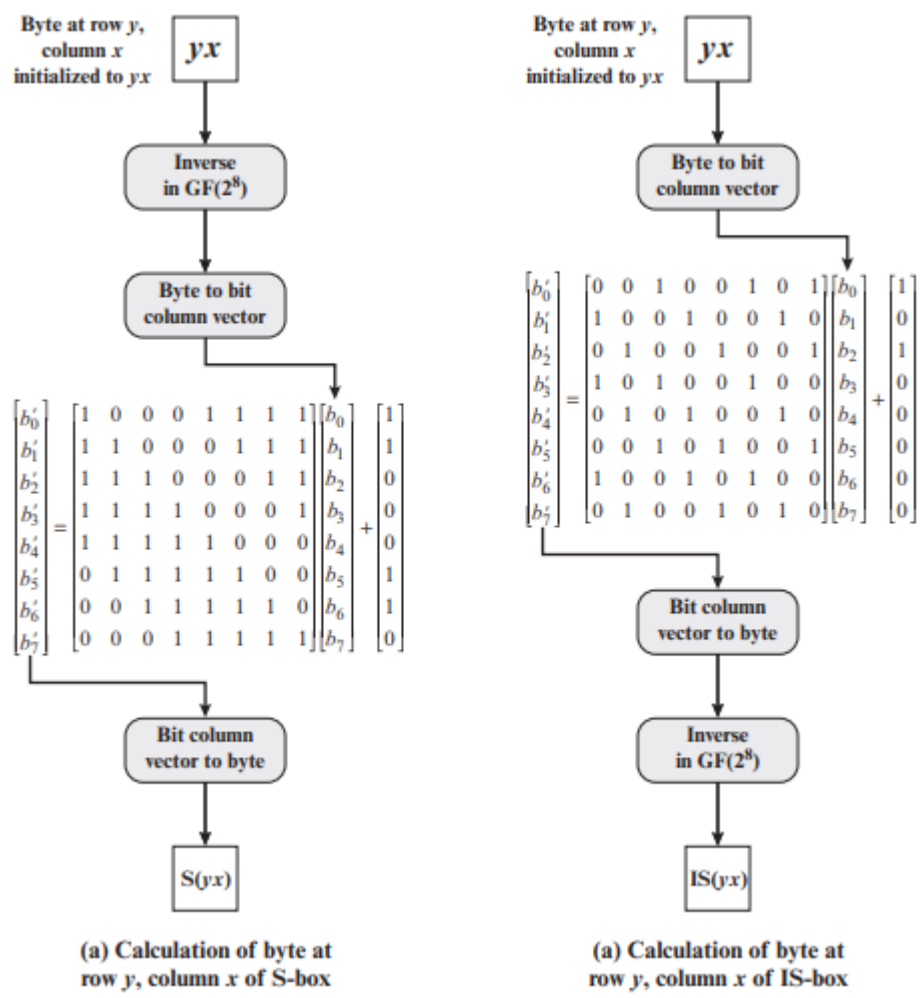
Here is an example of the Sub Bytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

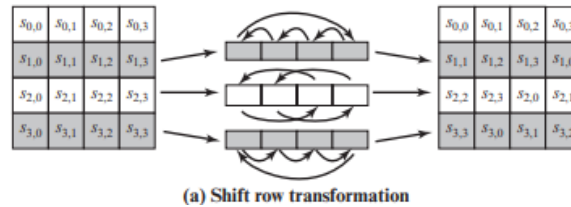
S box and inverse S box is constructed using following fashion



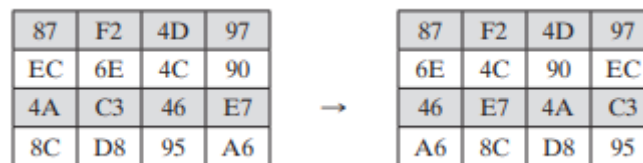
## Shift Rows Transformation

### Forward Transformation

In forward shift row transformation, called ShiftRows, the first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed.



The following is an example of ShiftRows.



### Inverse Transformation

The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a 1-byte circular right shift for the second row, and so on.

## Mix Columns Transformation

### Forward Transformation

The forward mix column transformation, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on State.

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

The MixColumns transformation on a single column of State can be expressed as

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

The following is an example of MixColumns:

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Let us verify the first column of this example.

$$\begin{aligned}
 (\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} &= \{47\} \\
 \{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} &= \{37\} \\
 \{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) &= \{94\} \\
 (\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) &= \{ED\}
 \end{aligned}$$

Multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1.

For the first equation, we have  $\{02\} \cdot \{87\} = (0000\ 1110) \oplus (0001\ 1011) = (0001\ 0101)$  and  $\{03\} \cdot \{6E\} = \{6E\} \oplus (\{02\} \cdot \{6E\}) = (0110\ 1110) \oplus (1101\ 1100) = (1011\ 0010)$ . Then,

$$\begin{aligned}
 \{02\} \cdot \{87\} &= 0001\ 0101 \\
 \{03\} \cdot \{6E\} &= 1011\ 0010 \\
 \{46\} &= 0100\ 0110 \\
 \{A6\} &= \underline{1010\ 0110} \\
 &0100\ 0111 = \{47\}
 \end{aligned}$$

The other equations can be similarly verified.

### Inverse Transformation

The inverse mix column transformation, called InvMixColumns, is defined by the following matrix multiplication:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}
 \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}
 =
 \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

### AddRound Key Transformation

#### Forward and Inverse Transformation

In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key.

The following is an example of AddRoundKey:

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

 $\oplus$ 

AC	19	28	57
77	FA	D1	5C
66	DC	29	00
F3	21	41	6A

 $=$ 

EB	59	8B	1B
40	2E	A1	C3
F2	38	13	42
1E	84	E7	D6

The first matrix is State, and the second matrix is the round key. The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

### AES Key Expansion

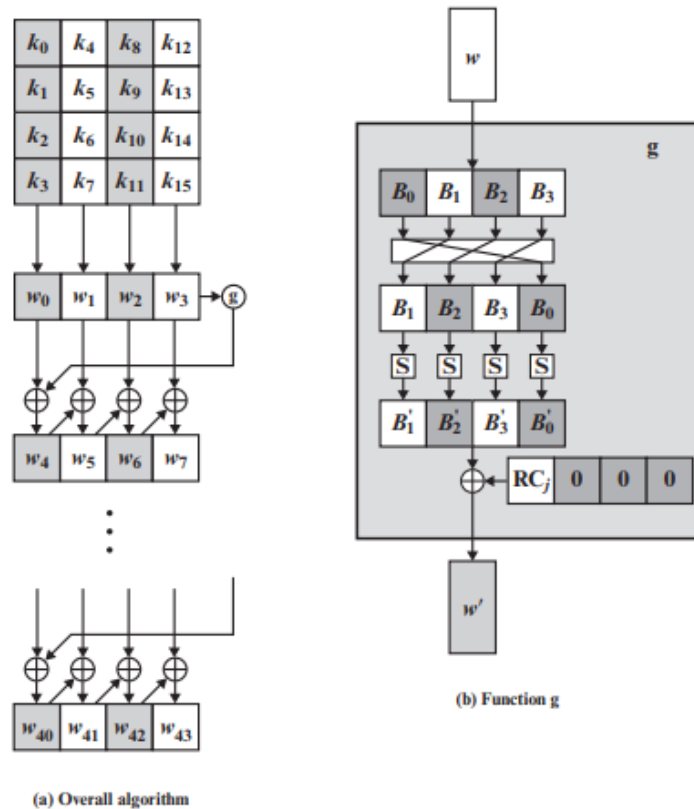
The AES key expansion algorithm takes as input a four-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a fourword round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher.

#### Algorithm:

1. The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time.
2. Each added word  $w[i]$  depends on the immediately preceding word,  $w[i - 1]$ , and the word four positions back,  $w[i - 4]$ .
3. In three out of four cases, a simple XOR is used. For a word whose position in the  $w$  array is a multiple of 4, a more complex function  $g$  is used.
4. RotWord performs a one-byte circular left shift on a word. This means that an input word  $[B0, B1, B2, B3]$  is transformed into  $[B1, B2, B3, B0]$ .
5. SubWord performs a byte substitution on each byte of its input word, using the S-box.
6. The result of steps 4 and 5 is XORed with a round constant,  $Rcon[j]$ .

The values of  $RC[j]$  in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36



## Public-Key Cryptosystems

Asymmetric algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic.

- It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic.

- Either of the two related keys can be used for encryption, with the other used for decryption.

A public-key encryption scheme has six ingredients

**Plaintext:** This is the readable message or data that is fed into the algorithm as input.

**Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.

**Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the algorithm depend on the public or private key that is provided as input.

**Ciphertext:** This is the encrypted message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.

**Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps are the following.

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

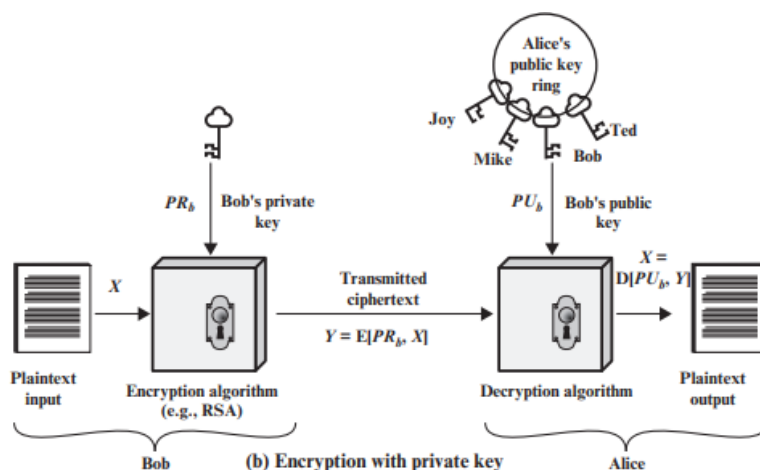
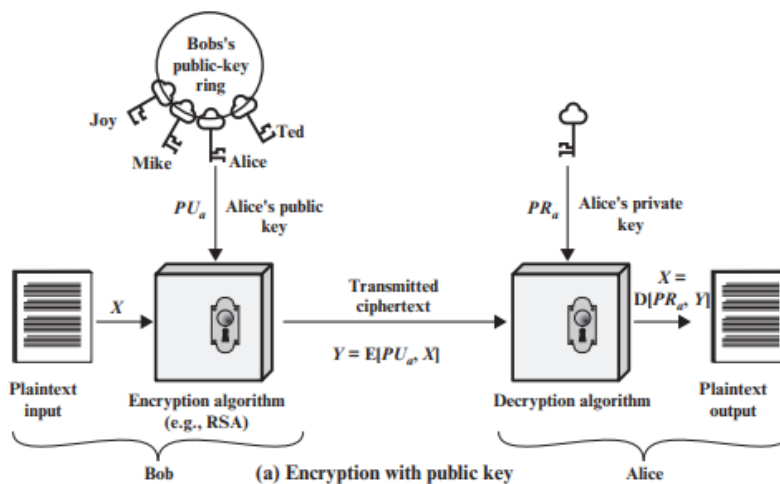
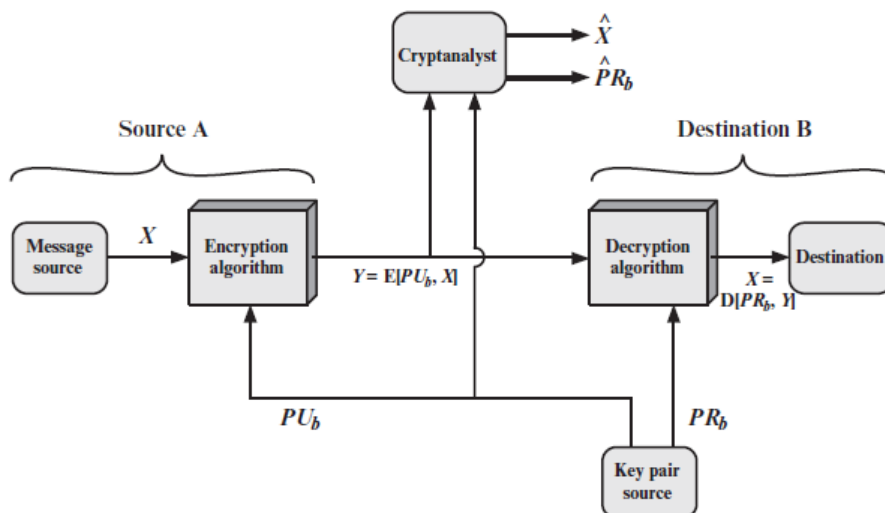


Table 9.2 Conventional and Public-Key Encryption

Conventional Encryption	Public-Key Encryption
<p><i>Needed to Work:</i></p> <ol style="list-style-type: none"> <li>1. The same algorithm with the same key is used for encryption and decryption.</li> <li>2. The sender and receiver must share the algorithm and the key.</li> </ol> <p><i>Needed for Security:</i></p> <ol style="list-style-type: none"> <li>1. The key must be kept secret.</li> <li>2. It must be impossible or at least impractical to decipher a message if the key is kept secret.</li> <li>3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key.</li> </ol>	<p><i>Needed to Work:</i></p> <ol style="list-style-type: none"> <li>1. One algorithm is used for encryption and a related algorithm for decryption with a pair of keys, one for encryption and one for decryption.</li> <li>2. The sender and receiver must each have one of the matched pair of keys (not the same one).</li> </ol> <p><i>Needed for Security:</i></p> <ol style="list-style-type: none"> <li>1. One of the two keys must be kept secret.</li> <li>2. It must be impossible or at least impractical to decipher a message if one of the keys is kept secret.</li> <li>3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key.</li> </ol>

### Applications of Public-key cryptosystem:

#### 1. Public-Key Cryptosystem: Confidentiality



- There is some source A that produces a message in plaintext,  $X = [X_1, X_2, \dots, X_M]$ . The  $M$  elements of  $X$  are letters in some finite alphabet. The message is intended for destination B.
- B generates a related pair of keys: a public key,  $PU_b$ , and a private key,  $PR_b$ .
- $PR_b$  is known only to B, whereas  $PU_b$  is publicly available and therefore accessible by A.
- With the message  $X$  and the encryption key  $PU_b$  as input, A forms the ciphertext  $Y = [Y_1, Y_2, \dots, Y_N]$ :

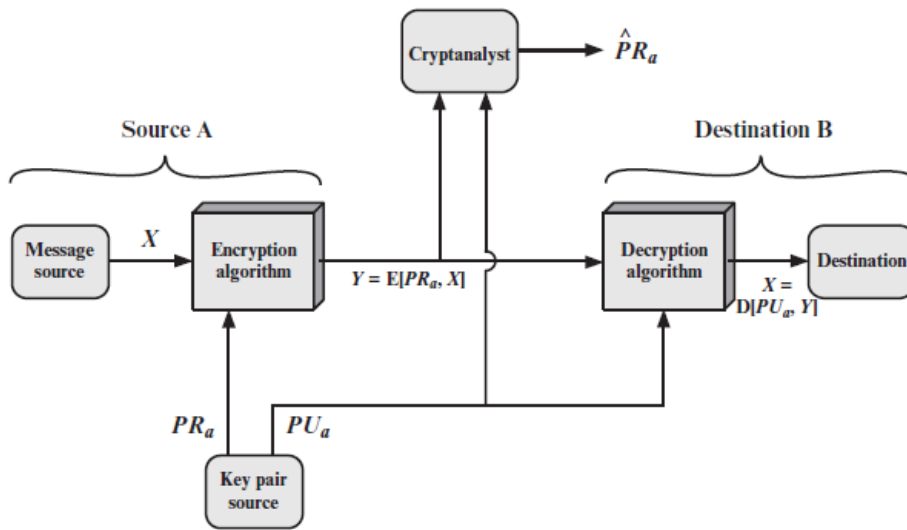
$$Y = E(PU_b, X)$$

- The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$X = D(PR_b, Y)$$

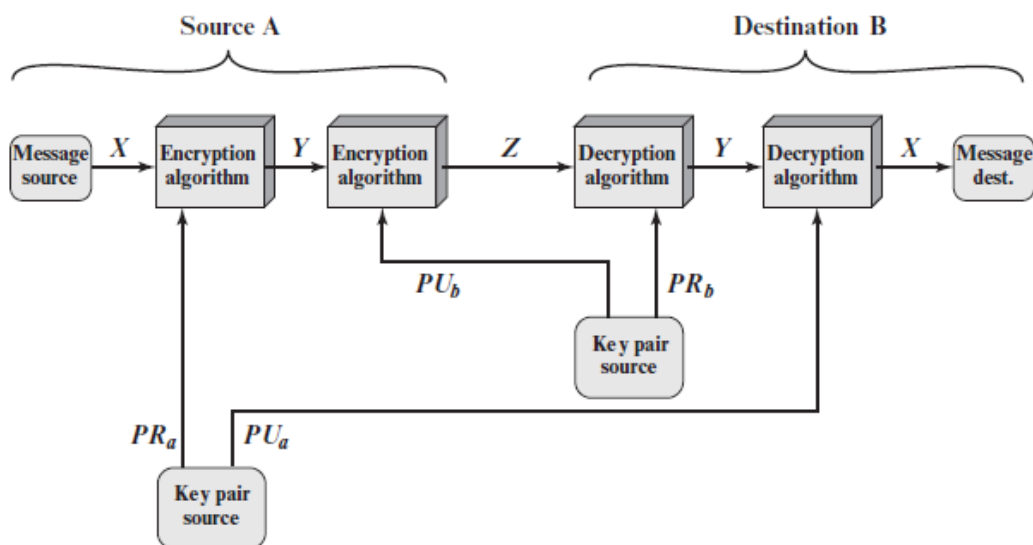
- Thus confidentiality is provided because B only can decrypt the message using private key.

## 2. Public-Key Cryptosystem: Authentication



- In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key.
- Because the message was encrypted using A's private key, only A could have prepared the message. Therefore, the entire encrypted message serves as a **digital signature**.
- In addition, it is impossible to alter the message without access to A's private key, so the message is authenticated both in terms of source and in terms of data integrity.
- Whereas this scheme does not produce confidentiality.

## 3. Public-Key Cryptosystem: Confidentiality and Authentication



It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme

$$\begin{aligned} Z &= E(PU_b, E(PR_a, X)) \\ X &= D(PU_a, D(PR_b, Z)) \end{aligned}$$

- In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature.
- Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided.
- The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.

### The RSA Algorithm:

The Rivest-Shamir-Adleman (RSA) scheme is the most widely accepted and implemented general-purpose approach to public-key encryption.

#### Algorithm:

1. Select  $p, q$  where  $p$  and  $q$  both are prime and  $p \neq q$ .
2. Calculate  $n = p * q$ .
3. Calculate  $\phi(n) = (p - 1)(q - 1)$ .
4. Select integer  $e$  such that  $e$  is relatively prime to  $\phi(n)$  i.e.  $\gcd(\phi(n), e) = 1$ ;  $1 < e < \phi(n)$ .

Determine  $d$  such that  $de = 1 \pmod{\phi(n)}$  and  $d < \phi(n)$ .  $d$  can be calculated using the extended Euclid's algorithm.  $d = ((\phi(n) * i) + 1) / e$

5. Public key is  $PU = \{e, n\}$  Private key is  $PR = \{d, n\}$
6. Encryption :  
Plaintext:  $M < n$   
Ciphertext:  $C = M^e \pmod{n}$
7. Decryption:  
Ciphertext:  $C$   
Plaintext:  $M = C^d \pmod{n}$

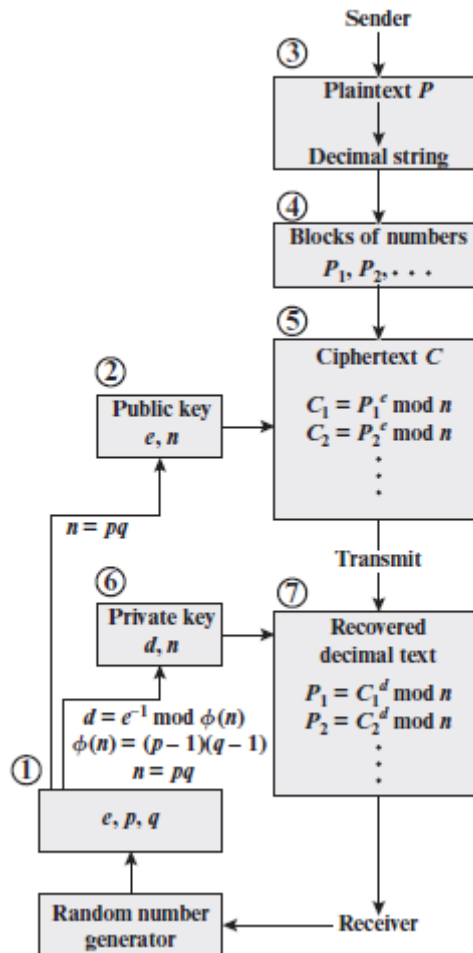


Fig. RSA Processing of Multiple Blocks

**For this example, the keys were generated as follows.**

1. Select two prime numbers,  $p = 17$  and  $q = 11$ .
2. Calculate  $n = pq = 17 * 11 = 187$ .
3. Calculate  $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$ .
4. Select  $e$  such that  $e$  is relatively prime to  $\phi(n) = 160$  and less than  $\phi(n)$ ; we choose  $e = 7$ .
5. Determine  $d$  such that  $de = 1 \pmod{160}$  and  $d < 160$ . The correct value is  $d = 23$ , because  $((160 * 1) + 1) / 7 = 23$ .
6. The resulting keys are public key  $PU = \{7, 187\}$  and private key  $PR = \{23, 187\}$ .

The example shows the use of these keys for a plaintext input of  $M = 88$ . For encryption, we need to calculate  $C = 88^7 \pmod{187}$ . Exploiting the properties of modular arithmetic, we can do this as follows.

$$88^7 \pmod{187} = [(88^4 \pmod{187}) * (88^2 \pmod{187}) * (88^1 \pmod{187})] \pmod{187}$$

$$88^1 \pmod{187} = 88$$

$$88^2 \pmod{187} = 7744 \pmod{187} = 77$$

$$88^4 \pmod{187} = 59,969,536 \pmod{187} = 132$$

$$\mathbf{88^7 \pmod{187} = (88 * 77 * 132) \pmod{187} = 894,432 \pmod{187} = 11}$$

For decryption, we calculate  $M = 11^{23} \pmod{187}$ :

$$11^{23} \pmod{187} = [(11^1 \pmod{187}) * (11^2 \pmod{187}) * (11^4 \pmod{187}) * (11^8 \pmod{187}) * (11^8 \pmod{187})] \pmod{187}$$

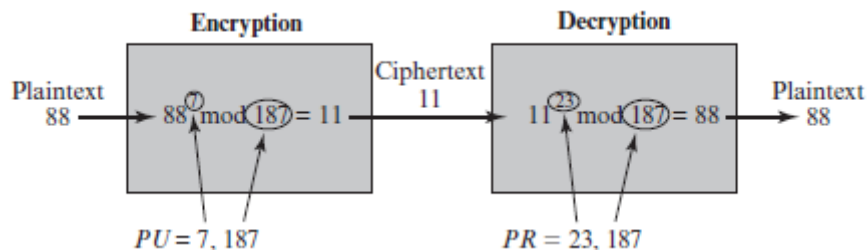
$$11^1 \bmod 187 = 11$$

$$11^2 \bmod 187 = 121$$

$$11^4 \bmod 187 = 14,641 \bmod 187 = 55$$

$$11^8 \bmod 187 = 214,358,881 \bmod 187 = 33$$

$$11^{23} \bmod 187 = (11 * 121 * 55 * 33 * 33) \bmod 187 = 79,720,245 \bmod 187 = 88$$



## The Security of RSA

Five possible approaches to attacking the RSA algorithm are

- **Brute force:** This involves trying all possible private keys.
- **Mathematical attacks:** There are several approaches, all equivalent in effort to factoring the product of two primes.
- **Timing attacks:** These depend on the running time of the decryption algorithm.
- **Hardware fault-based attack:** This involves inducing hardware faults in the processor that is generating digital signatures.
- **Chosen ciphertext attacks:** This type of attack exploits properties of the RSA algorithm.

### Brute force:

Brute force is trying all possible private keys. The defense against the brute-force approach is to use a large key space. Thus, the larger the number of bits in  $d$ , the better. However, because the calculations involved, both in key generation and in encryption/decryption, are complex, the larger the size of the key, the slower the system will run.

### Mathematical attacks:

We can identify three approaches to attacking RSA mathematically.

1. Factor  $n$  into its two prime factors. This enables calculation of  $\phi(n) = (p - 1) * (q - 1)$ , which in turn enables determination of  $d = e^{-1} \pmod{\phi(n)}$ .
2. Determine  $\phi(n)$  directly, without first determining  $p$  and  $q$ . Again, this enables determination of  $d = e^{-1} \pmod{\phi(n)}$ .
3. Determine  $d$  directly, without first determining  $\phi(n)$ .

To avoid values of  $n$  that may be factored more easily, the algorithm's inventors suggest the following constraints on  $p$  and  $q$ .

1.  $p$  and  $q$  should differ in length by only a few digits. Thus, for a 1024-bit key (309 decimal digits), both  $p$  and  $q$  should be on the order of magnitude of  $10^{75}$  to  $10^{100}$ .
2. Both  $(p - 1)$  and  $(q - 1)$  should contain a large prime factor.
3.  $\gcd(p - 1, q - 1)$  should be small.

### **Timing Attacks:**

Snooper can determine a private key by keeping track of how long a computer takes to decipher messages.

Although the timing attack is a serious threat, there are simple countermeasures that can be used, including the following.

1. **Constant exponentiation time:** Ensure that all exponentiations take the same amount of time before returning a result. This is a simple fix but does degrade performance.
2. **Random delay:** Better performance could be achieved by adding a random delay to the exponentiation algorithm to confuse the timing attack.
3. **Blinding:** Multiply the ciphertext by a random number before performing exponentiation. This process prevents the attacker from knowing what cipher text bits are being processed inside the computer and therefore prevents the bit-by-bit analysis essential to the timing attack.

### **Fault-based attack:**

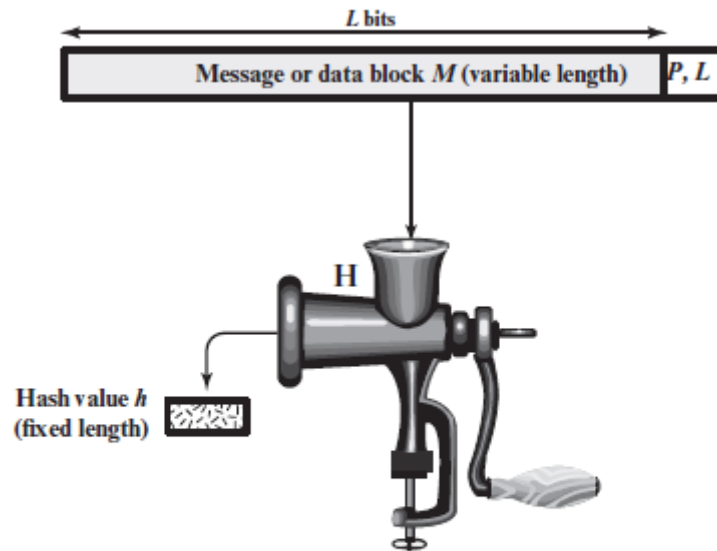
- The approach is an attack on a processor that is generating RSA digital signatures. The attack induces faults in the signature computation by reducing the power to the processor. The faults cause the software to produce invalid signatures, which can then be analyzed by the attacker to recover the private key.
- This attack, while worthy of consideration, does not appear to be a serious threat to RSA. It requires that the attacker have physical access to the target machine and that the attacker is able to directly control the input power to the processor.

### **Chosen ciphertext attacks: (CCA)**

- CCA is defined as an attack in which the adversary chooses a number of ciphertexts and is then given the corresponding plaintexts, decrypted with the target's private key. Thus, the adversary could select a plaintext, encrypt it with the target's public key, and then be able to get the plaintext back by having it decrypted with the private key.
- Clearly, this provides the adversary with no new information. Instead, the adversary exploits properties of RSA and selects blocks of data that, when processed using the target's private key, yield information needed for cryptanalysis.
- To overcome this simple attack, practical RSA-based cryptosystems randomly pad the plaintext prior to encryption.

## Cryptographic Hash Functions

A **hash function**  $H$  accepts a variable-length block of data  $M$  as input and produces a fixed-size hash value  $h = H(M)$ . Typically, the input is padded out to an integer multiple of some fixed length (e.g., 1024 bits), and the padding includes the value of the length of the original message in bits.



$P, L =$  padding plus length field

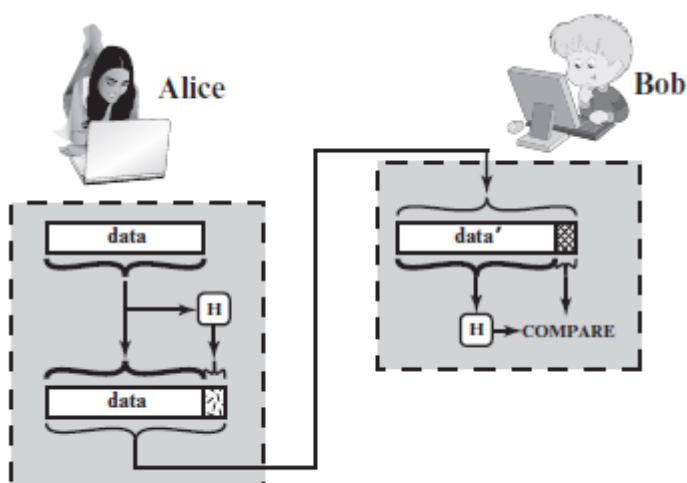
Figure 11.1 Cryptographic Hash Function;  $h = H(M)$

Hash functions are often used to determine whether or not data has changed.

### Applications of cryptographic Hash function:

#### 1. Message Authentication:

- 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 (i.e., there is no modification, insertion, deletion, or replay).
- When a hash function is used to provide message authentication, the hash function value is often referred to as a **message digest**.

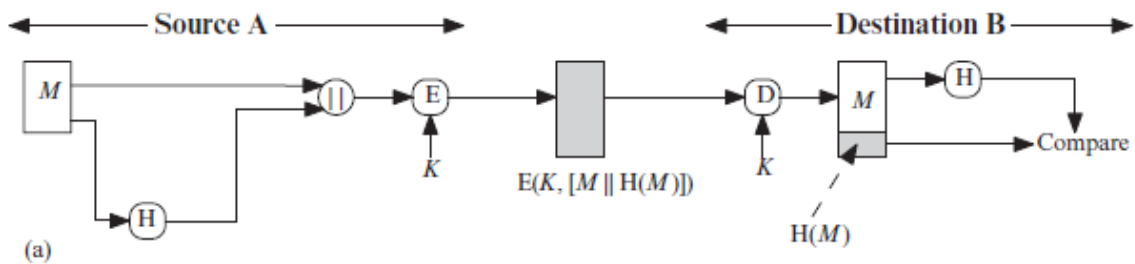


(a) Use of hash function to check data integrity

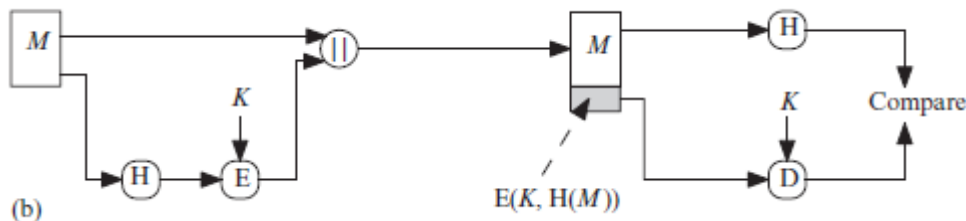
- The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message. The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.

Hash code can be used to provide message authentication in variety of ways:

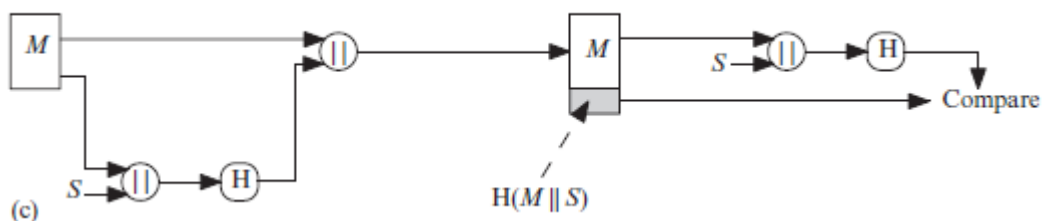
- The message plus concatenated hash code is encrypted using symmetric encryption. Because only A and B share the secret key, the message must have come from A and has not been altered. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



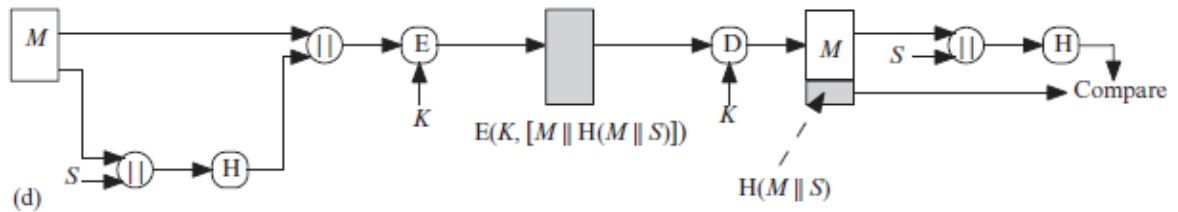
- Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality.



- It is possible to use a hash function but no encryption for message authentication. The technique assumes that the two communicating parties share a common secret value  $S$ . A computes the hash value over the concatenation of  $M$  and  $S$  and appends the resulting hash value to  $M$ . Because B possesses  $S$ , it can recompute the hash value to verify. Because the secret value itself is not sent, an opponent cannot modify an intercepted message and cannot generate a false message.



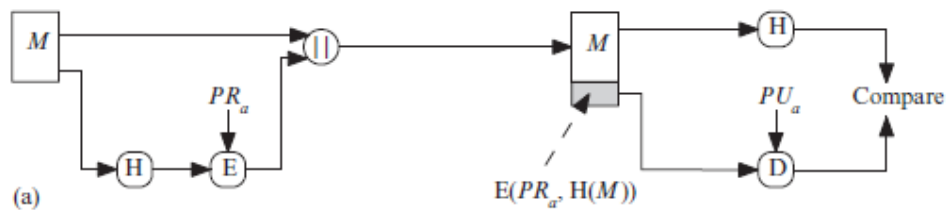
- Confidentiality can be added to the approach of method (c) by encrypting the entire message plus the hash code.



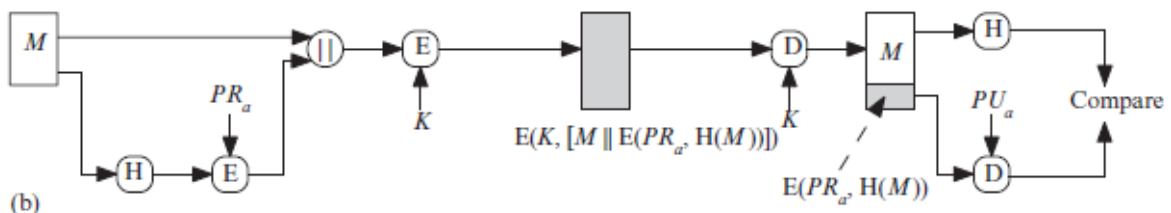
## 2. Digital Signatures:

In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message would need to know the user's private key.

- a) The hash code is encrypted, using public-key encryption with the sender's private key. This provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code.



- b) If confidentiality as well as a digital signature is desired, then the message plus the private-key-encrypted hash code can be encrypted using a symmetric secret key.



## 3. Other Applications:

- Hash functions are commonly used to create a **one-way password file**. Hash of a password is stored by an operating system rather than the password itself. When a user enters a password, the hash of that password is compared to the stored hash value for verification. This approach to password protection is used by most operating systems.
- Hash functions can be used for **intrusion detection** and **virus detection**. Store  $H(F)$  for each file on a system and secure the hash values (e.g., on a CD-R that is kept secure). One can later determine if a file has been modified by recomputing  $H(F)$ .

### Two Simple Hash Function:

The input (message, file, etc.) is viewed as a sequence of  $n$ -bit blocks. The input is processed one block at a time in an iterative fashion to produce an  $n$ -bit hash function.

One of the simplest hash functions is the bit-by-bit exclusive-OR (XOR) of every block. This can be expressed as

$$C_i = b_{i1} \oplus b_{i2} \oplus \dots \oplus b_{im}$$

where

$C_i = i$ th bit of the hash code,  $1 \leq i \leq n$

$m =$  number of  $n$ -bit blocks in the input

$b_{ij} = i$ th bit in  $j$ th block

$\oplus =$  XOR operation

This operation produces a simple parity bit for each bit position and is known as a longitudinal redundancy check. With more predictably formatted data, the function is less effective.

A simple way to improve matters is to perform a one-bit circular shift, or rotation, on the hash value after each block is processed. The procedure can be summarized as follows.

1. Initially set the  $n$ -bit hash value to zero.
2. Process each successive  $n$ -bit block of data as follows:
  - a. Rotate the current hash value to the left by one bit.
  - b. XOR the block into the hash value.

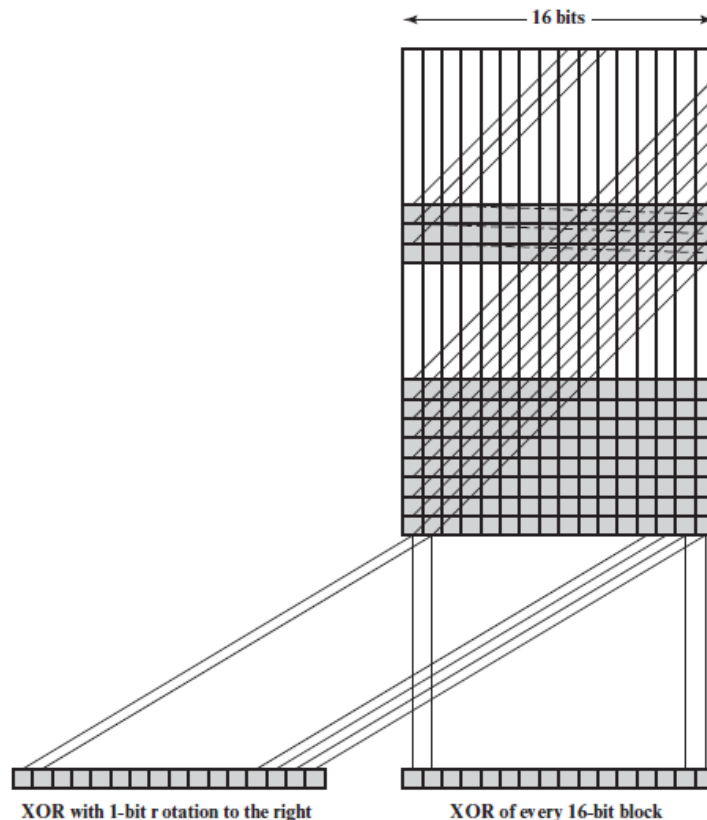


Figure 11.5 Two Simple Hash Functions

## Requirements for a Cryptographic Hash function H

Requirement	Description
Variable input size	H can be applied to a block of data of any size.
Fixed output size	H produces a fixed-length output.
Efficiency	$H(x)$ is relatively easy to compute for any given $x$ , making both hardware and software implementations practical.
Preimage resistant (one-way property)	For any given hash value $h$ , it is computationally infeasible to find $y$ such that $H(y) = h$ .
Second preimage resistant (weak collision resistant)	For any given block $x$ , it is computationally infeasible to find $y \neq x$ with $H(y) = H(x)$ .
Collision resistant (strong collision resistant)	It is computationally infeasible to find any pair $(x, y)$ with $x \neq y$ , such that $H(x) = H(y)$ .
Pseudorandomness	Output of H meets standard tests for pseudorandomness.

A hash function that satisfies the first five properties in Table is referred to as a weak hash function. If the sixth property, **collision resistant**, is also satisfied, then it is referred to as a strong hash function.

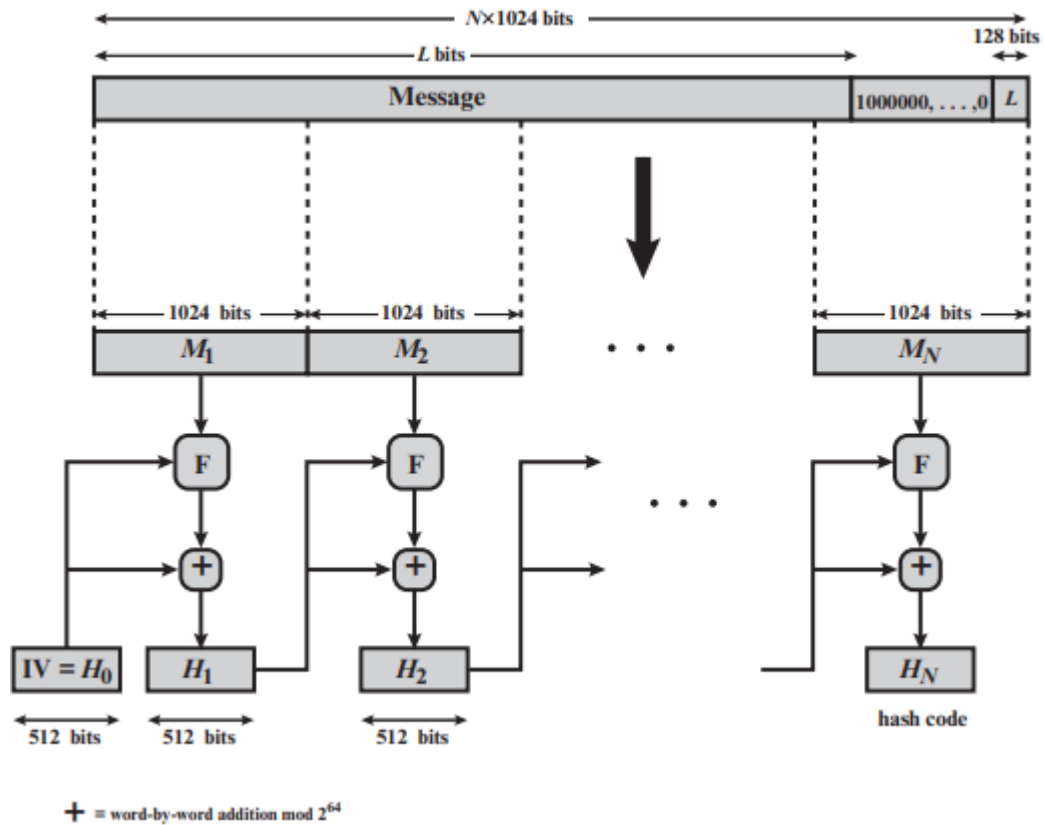
### Secure Hash Algorithm (SHA):

In recent years SHA is the widely used hash function. SHA was developed by the National Institute of Standards and Technology (NIST). Three new versions of SHA, with hash value lengths of 256, 384, and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively are defined.

#### SHA- 512 Algorithm:

The algorithm takes as input a message with a maximum length of less than  $2^{128}$  bits and produces as output a 512-bit message digest. The input is processed in 1024-bit blocks.

Figure depicts the overall processing of a message to produce a digest.



The processing consists of the following steps.

**Step 1 Append padding bits.**

The message is padded so that its length is congruent to 896 modulo 1024 [length  $\equiv 896(\text{mod } 1024)$ ]. Padding is always added, even if the message is already of the desired length. Thus, the number of padding bits is in the range of 1 to 1024. The padding consists of a single 1 bit followed by the necessary number of 0 bits.

**Step 2 Append length.**

A block of 128 bits is appended to the message. It contains the length of the original message in bits (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 1024 bits in length. In Figure, the expanded message is represented as the sequence of 1024-bit blocks  $M_1, M_2, \dots, M_N$ , so that the total length of the expanded message is  $N * 1024$  bits.

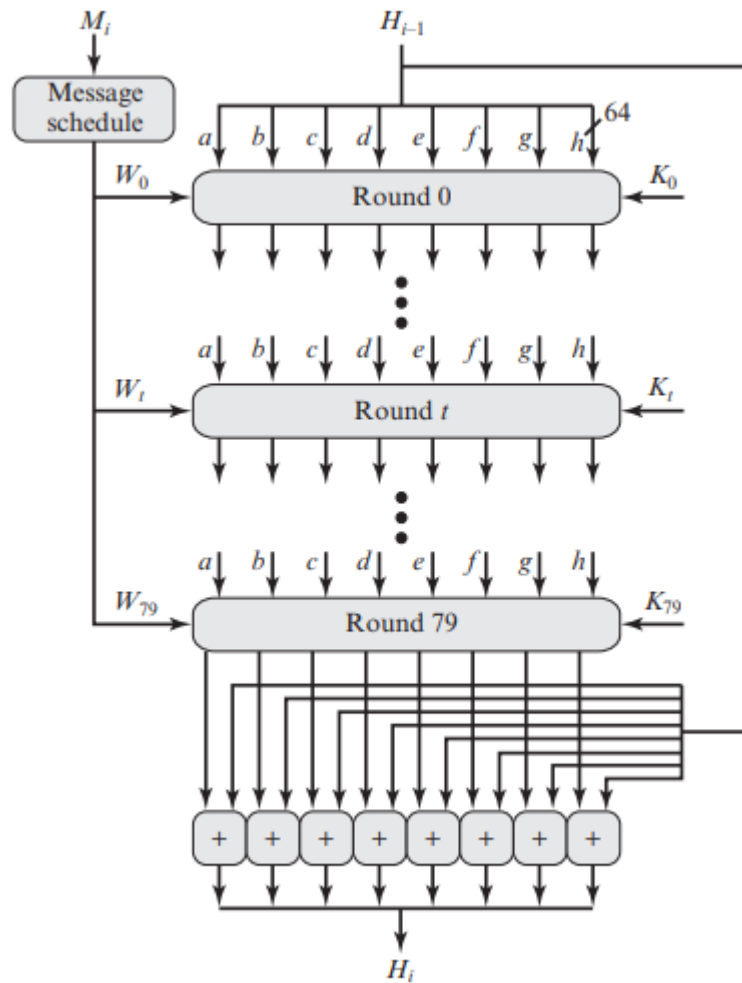
**Step 3 Initialize hash buffer.**

A 512-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as eight 64-bit registers (a, b, c, d, e, f, g, h). These words were obtained by taking the first sixty-four bits of the fractional parts of the square roots of the first eight prime numbers.

**Step 4 Process message in 1024-bit (128-byte) blocks.**

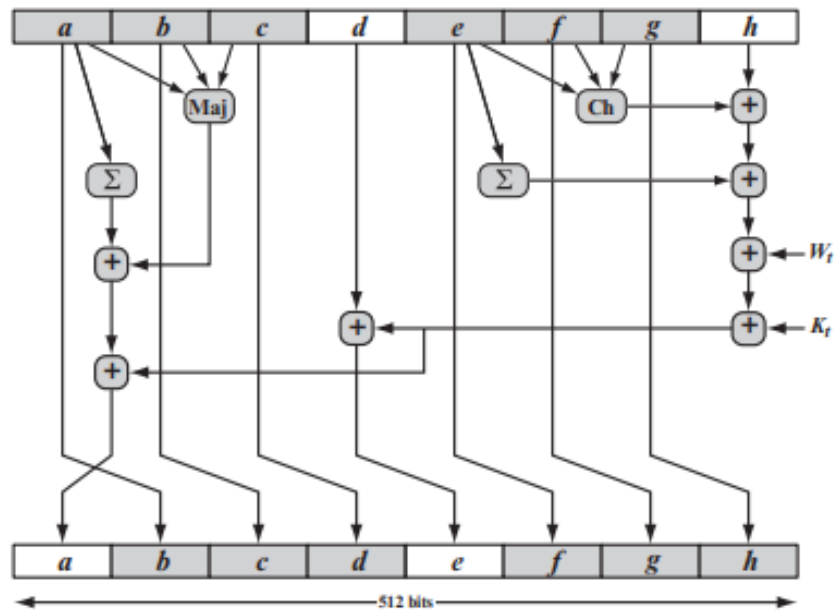
The heart of the algorithm is a module that consists of 80 rounds; this module is labeled F in Figure.

## F module



- Each round takes as input the 512-bit buffer value, abcdefgh, and updates the contents of the buffer.
- Each round  $t$  makes use of a 64-bit value  $W_t$ , derived from the current 1024-bit block being processed ( $M_i$ ).
- Each round also makes use of an additive constant  $K_t$ , where  $0 \leq t \leq 79$  indicates one of the 80 rounds. These words represent the first 64 bits of the fractional parts of the cube roots of the first 80 prime numbers.
- The output of the eightieth round is added to the input to the first round ( $H_{i-1}$ ) to produce  $H_i$ . The addition is done independently for each of the eight words in the buffer with each of the corresponding words in  $H_{i-1}$ , using addition modulo  $2^{64}$ .

## Round Function



Each round is defined by the following set of equations:

$$T_1 = h + \text{Ch}(e, f, g) + (\sum_1^{512} e) + W_t + K_t$$

$$T_2 = (\sum_0^{512} a) + \text{Maj}(a, b, c)$$

$$h = g$$

$$g = f$$

$$f = e$$

$$e = d + T_1$$

$$d = c$$

$$c = b$$

$$b = a$$

$$a = T_1 + T_2$$

where

$t$  = step number;  $0 \leq t \leq 79$

$\text{Ch}(e, f, g) = (e \text{ AND } f) \oplus (\text{NOT } e \text{ AND } g)$   
*the conditional function: If e then f else g*

$\text{Maj}(a, b, c) = (a \text{ AND } b) \oplus (a \text{ AND } c) \oplus (b \text{ AND } c)$   
*the function is true only if the majority (two or three) of the arguments are true*

$(\sum_0^{512} a) = \text{ROTR}^{28}(a) \oplus \text{ROTR}^{34}(a) \oplus \text{ROTR}^{39}(a)$

$(\sum_1^{512} e) = \text{ROTR}^{14}(e) \oplus \text{ROTR}^{18}(e) \oplus \text{ROTR}^{41}(e)$

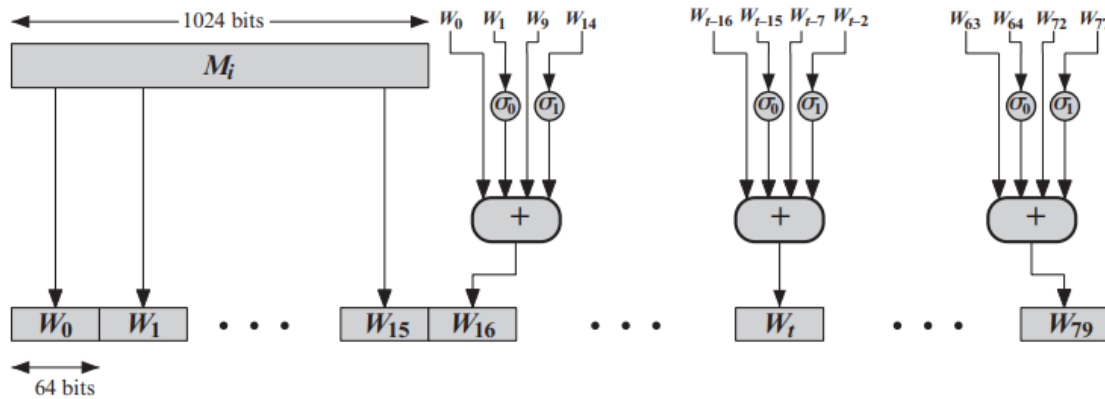
$\text{ROTR}^n(x) =$  circular right shift (rotation) of the 64-bit argument  $x$  by  $n$  bits

$W_t$  = a 64-bit word derived from the current 1024-bit input block

$K_t$  = a 64-bit additive constant

+ = addition modulo 264

### Generation of $W_t$



In the first 16 steps of processing, the value of  $W_t$  is equal to the corresponding word in the message block. For the remaining 64 steps, the value of  $W$  is obtained by following equation

$$W_t = \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16}$$

where

$$\sigma_0^{512}(x) = \text{ROTR}^1(x) \oplus \text{ROTR}^8(x) \oplus \text{SHR}^7(x)$$

$$\sigma_1^{512}(x) = \text{ROTR}^{19}(x) \oplus \text{ROTR}^{61}(x) \oplus \text{SHR}^6(x)$$

$\text{ROTR}^n(x)$  = circular right shift (rotation) of the 64-bit argument  $x$  by  $n$  bits

$\text{SHR}^n(x)$  = right shift of the 64-bit argument  $x$  by  $n$  bits with padding by zeros on the left

+ = addition modulo  $2^{64}$

### Step 5 Output.

After all  $N$  1024-bit blocks have been processed, the output from the  $N$ th stage is the 512-bit message digest.

$$H_0 = \text{IV}$$

$$H_i = \text{SUM}_{64}(H_{i-1}, \text{abcdefgh}_i)$$

$$\text{MD} = H_N$$

where

IV = initial value of the abcdefgh buffer, defined in step 3

$\text{abcdefgh}_i$  = the output of the last round of processing of the  $i$ th message block

$N$  = the number of blocks in the message (including padding and length fields)

$\text{SUM}_{64}$  = addition modulo  $2^{64}$  performed separately on each word of the pair of inputs

MD = final message digest value

## Digital Signature:

Message authentication protects two parties who exchange messages from any third party. However, it does not protect the two parties against each other. For example 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 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.

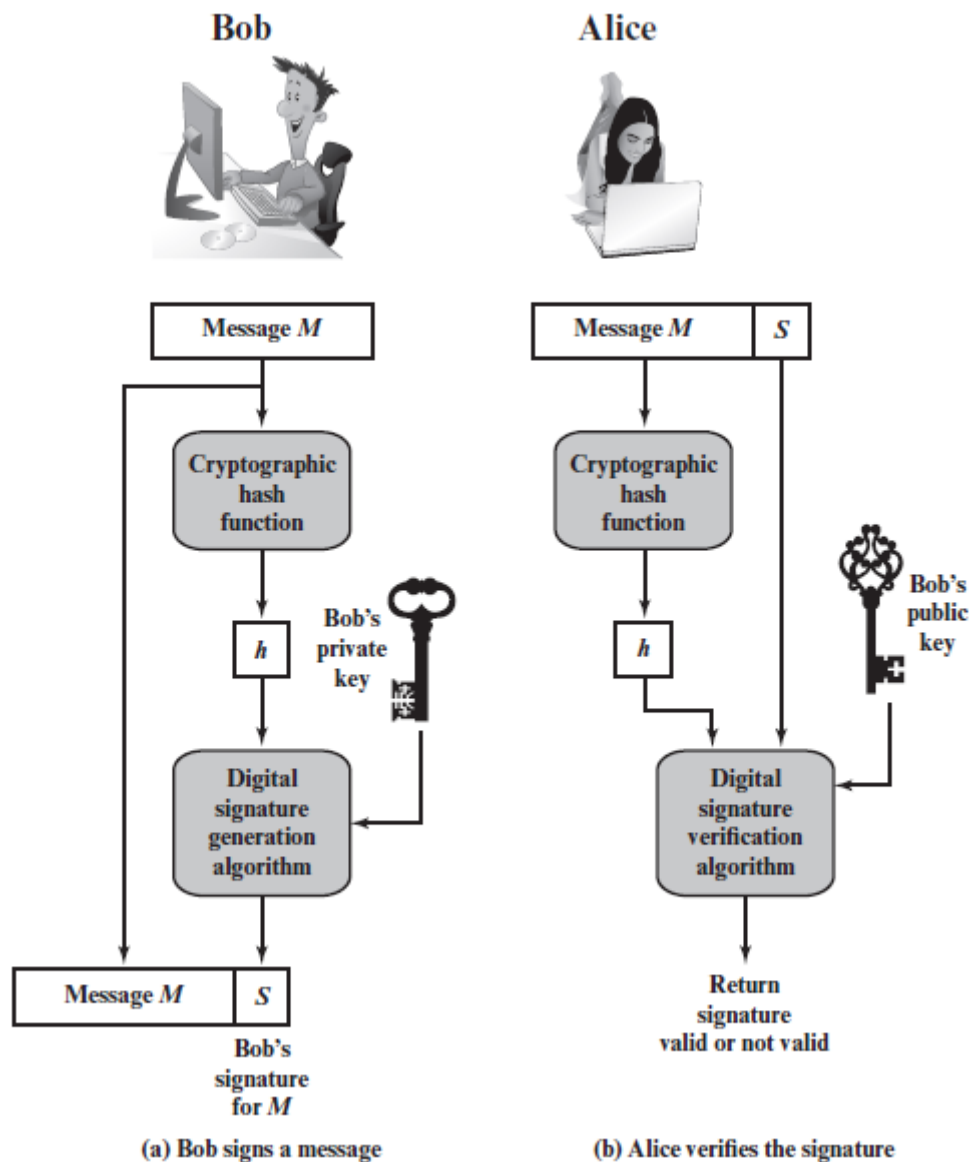
In situations where there is not complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature.

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.

Fig shows the simplified depiction of essential elements of digital signature process.



- Suppose that Bob wants to send a message to Alice. Although it is not important that the message be kept secret, he wants Alice to be certain that the message is indeed from him. For this purpose, Bob uses a secure hash function, such as SHA-512, to generate a hash value for the message.
- That hash value, together with Bob's private key serves as input to a digital signature generation algorithm, which produces a short block that functions as a **digital signature**.
- Bob sends the message with the signature attached. When Alice receives the message plus signature, she (1) calculates a hash value for the message; (2) provides the hash value and Bob's public key as inputs to a digital signature verification algorithm.
- If the algorithm returns the result that the signature is valid, Alice is assured that the message must have been signed by Bob.
- No one else has Bob's private key and therefore no one else could have created a signature that could be verified for this message with Bob's public key.

#### Digital Signature Requirements:

- The signature must be a bit pattern that depends on the message being signed.
- The signature must use some information only known 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.

### 1. Elgamal Digital Signature Scheme

The Elgamal signature scheme involves the use of the private key for digital signature generation and the public key for digital signature verification.

#### Algorithm:

1. Select a prime number  $q$ .

2. Select a primitive root  $\alpha$  of  $q$ .

{ if  $a$  is a primitive root of  $n$ , then its powers  $a, a^2, \dots, a^{\phi(n)}$  are distinct (mod  $n$ ) and are all relatively prime to  $n$ . }

3. Generate a random integer  $X_A$  such that  $1 < X_A < q-1$

4. Compute  $Y_A = \alpha^{X_A} \text{ mod } q$ .

5. Now A's Private Key is  $X_A$  and public key is  $\{q, \alpha, Y_A\}$

6. Generate hash code  $m$  for message  $M$ .  $m = H(M)$  such that  $m$  is an integer in the range  $0 \leq m \leq q-1$ .

7. Choose a random integer  $K$  such that  $1 \leq K \leq q-1$  and  $\text{gcd}(K, q-1) = 1$ .

i.e.  $K$  is relatively prime to  $q-1$ .

8. Compute  $S_1 = \alpha^K \bmod q$ .
9. Compute  $S_2 = K^{-1} (m - X_A S_1) \bmod q-1$  where  $K^{-1} = ((q-1)^i + 1)/K$
10. The signature consists of the pair  $(S_1, S_2)$ .

**Verification at receiver side:**

1. Compute  $V_1 = \alpha^m \bmod q$ .
2. Compute  $V_2 = (Y_A)^{S_1} (S_1)^{S_2} \bmod q$ .

**The signature is valid if  $V_1 = V_2$ .**

Example

1. Let  $q = 19$ .
2. Choose  $\alpha = 10$  which is a primitive root of 19.
3. Choose  $X_A = 16$ .
4.  $Y_A = \alpha^{X_A} \bmod q$   
 $= 10^{16} \bmod 19 = 4$ .
5. A's Private Key is 16 and public key is  $\{q, \alpha, Y_A\} = \{19, 10, 4\}$ .
6. Let hash value  $m = 14$ .
7. Choose  $K = 5$  which is relatively prime to  $q - 1 = 18$ .
8.  $S_1 = \alpha^K \bmod q = 10^5 \bmod 19 = 3$
9.  $S_2 = K^{-1} (m - X_A S_1) \bmod q-1$   
 $K^{-1} = ((q-1)^i + 1)/K$   
 $i=1$   
 $K^{-1} = (18+1)/5 = 3.8$   
 $i=2$   
 $K^{-1} = ((18*2) + 1)/5 = 7.4$   
 $i=3$   
 $K^{-1} = ((18*3) + 1)/5 = 11$ .  
Hence  $K^{-1} = 11$

$$S_2 = 11 (14 - (16)(3)) \bmod 18 = -374 \bmod 18$$

$$= 18 - [374 \bmod 18]$$

$$= 18 - 14 = 4$$

10. The signature consists of the pair  $(3, 4)$ .

Verification

1.  $V_1 = \alpha^m \bmod q = 10^{14} \bmod 19 = 16$ .
  2.  $V_2 = (Y_A)^{S_1} (S_1)^{S_2} \bmod q = (4^3)(3^4) \bmod 19 = 5184 \bmod 19 = 16$ .
- Thus, the signature is valid because  $V_1 = V_2$ .

## 2. Schnorr Digital Signature Scheme

The first part of this scheme is the generation of a private/public key pair, which consists of the following steps.

1. Choose primes  $p$  and  $q$ , such that  $q$  is a prime factor of  $p - 1$ .
2. Choose an integer  $a$ , such that  $a^q = 1 \bmod p$ . The values  $a$ ,  $p$ , and  $q$  comprise a global public key that can be common to a group of users.

3. Choose a random integer  $s$  with  $0 < s < q$ . This is the user's private key.
4. Calculate  $v = a^{-s} \pmod p$ . This is the user's public key. A user with private key  $s$  and public key  $v$  generates a signature as follows.

1. Choose a random integer  $r$  with  $0 < r < q$  and compute  $x = a^r \pmod p$ . This computation is a pre-processing stage independent of the message  $M$  to be signed.

2. Concatenate the message with  $x$  and hash the result to compute the value  $e$ :

$$e = H(M || x)$$

3. Compute  $y = (r + se) \pmod q$ . The signature consists of the pair  $(e, y)$ . Any other user can verify the signature as follows.

1. Compute  $x' = a^y v^e \pmod p$ .
2. Verify that  $e = H(M || x')$ .

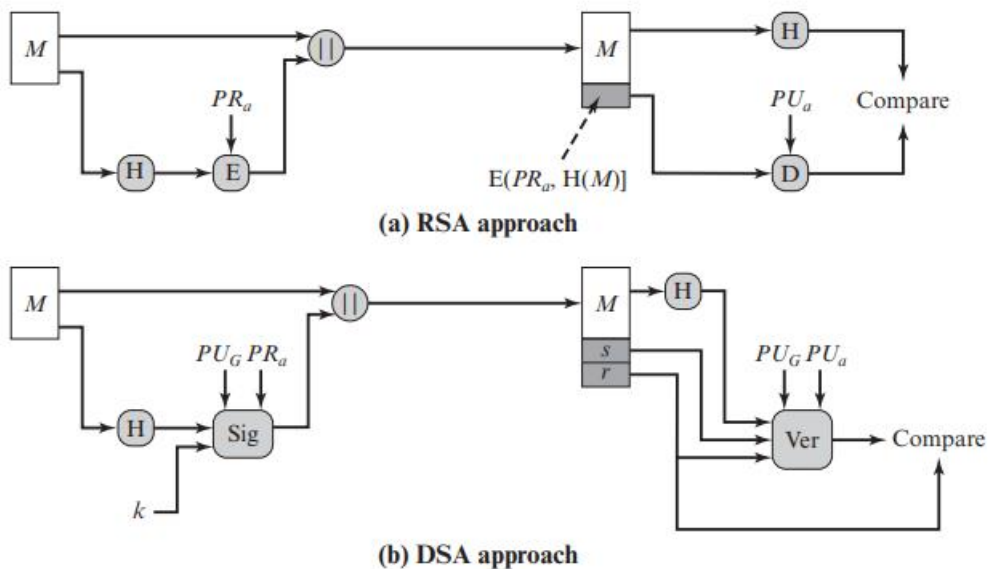
To see that the verification works, observe that  $x' \equiv a^y v^e \equiv a^y a^{-se} \equiv a^{y-se} \equiv a^r \equiv x \pmod p$ . Hence,  $H(M || x') = H(M || x)$ .

### Comparison of Elgamal and Schnorr

- Elgamal signature scheme is more time consuming in compare to Schnorr Scheme.
- Schnorr scheme is 6 times faster than Elgamal and produce signature which is 6 times smaller.

### 3. Digital Signature Algorithm

#### RSA Approach



- 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.

### **DSA Approach**

- The DSA approach also makes use of a hash function. The hash code is provided as input to a signature function along with a random number  $k$  generated for this particular signature.
- The signature function also depends on the sender's private key (PRa) and a set of parameters known to a group of communicating principals. We can consider this set to constitute a global public key (PUG). The result is a signature consisting of two components, labeled  $s$  and  $r$ .
- At the receiving end, the hash code of the incoming message is generated. The hash code and the signature are inputs to a verification function. The verification function also depends on the global public key as well as the sender's public key (PUa), 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  $r$  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.

### **Digital Signature Algorithm**

#### **Choose Global Public-Key Components**

$p$  prime number where  $2^{L-1} < p < 2^L$  for  $512 \leq L \leq 1024$  and  $L$  a multiple of 64; i.e., bit length  $L$  between 512 and 1024 bits in increments of 64 bits

$q$  prime divisor of  $(p - 1)$ , where  $2^{N-1} < q < 2^N$  i.e., bit length of  $N$  bits

$g = h(p - 1)/q$  is an exponent mod  $p$ ,  
where  $h$  is any integer with  $1 < h < (p - 1)$  such that  $h^{(p-1)/q} \bmod p > 1$

#### **Choose User's Private Key**

$x$  random or pseudorandom integer with  $0 < x < q$

#### **Generate User's Public Key**

$y = g^x \bmod p$

#### **User's Per-Message Secret Number**

$k$  random or pseudorandom integer with  $0 < k < q$

#### **Signing**

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1} (H(M) + xr)] \bmod q$

Signature = (r, s)

### Verifying

$$w = (s')^{-1} \text{ mod } q$$

$$u_1 = [H(M')w] \text{ mod } q$$

$$u_2 = (r')^w \text{ mod } q$$

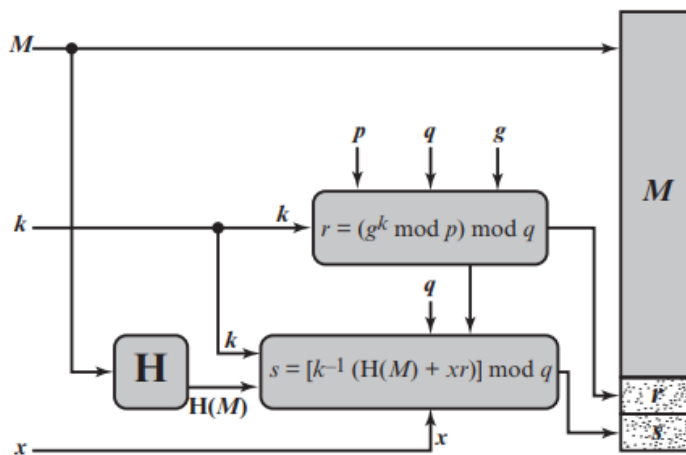
$$v = [(g^{u_1} y^{u_2}) \text{ mod } p] \text{ mod } q$$

$$\text{TEST: } v = r'$$

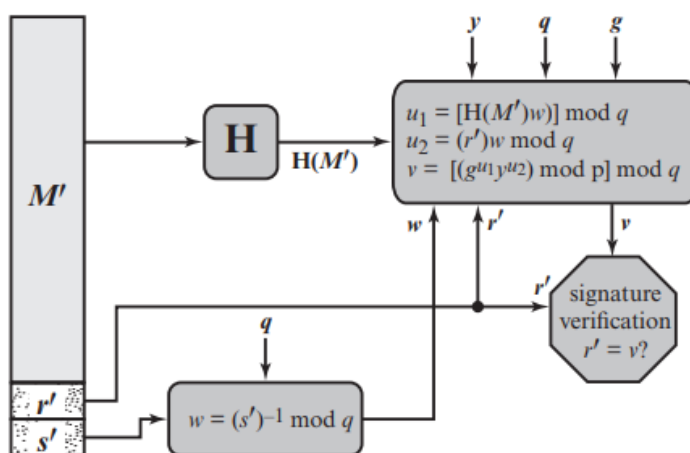
M = message to be signed

H(M) = hash of M using SHA-1

M', r', s' = received versions of M, r, s



(a) Signing



(b) Verifying

Verification is performed using the formulas shown in Figure. The receiver generates a quantity  $v$  that is a function of the public key components, the sender's public key, the

hash code of the incoming message, and the received versions of  $r$  and  $s$ . If this quantity matches the  $r$  component of the signature, then the signature is validated.

#### 4. Elliptic Curve Digital Signature Algorithm: ECDSA

ECDSA is enjoying increasing acceptance due to the efficiency advantage of elliptic curve cryptography, which yields security comparable to that of other schemes with a smaller key bit length.

##### Algorithm

##### Global domain Parameters

- $q$  a prime number
- $a, b$  integers that specify the elliptic curve equation defined over  $Z_q$  with the equation  $y^2 = x^3 + ax + b$
- $G$  a base point represented by  $G = (x_g, y_g)$  on the elliptic curve equation
- $n$  order of point  $G$ ; that is,  $n$  is the smallest positive integer such that  $nG = O$ . This is also the number of points on the curve.

##### Key generation

Each signer must generate a pair of keys, one private and one public. The signer, let us call him Bob, generates the two keys using the following steps:

1. Select a random integer  $d$ ,  $d \in [1, n - 1]$
2. Compute  $Q = dG$ . This is a point in  $Eq(a, b)$
3. Bob's public key is  $Q$  and private key is  $d$ .

##### Digital Signature generation and authentication

With the public domain parameters and a private key in hand, Bob generates a digital signature of 320 bytes for message  $m$  using the following steps:

1. Select a random or pseudorandom integer  $k$ ,  $k \in [1, n - 1]$
2. Compute point  $P = (x, y) = kG$  and  $r = x \bmod n$ . If  $r = 0$  then go to step 1
3. Compute  $t = k^{-1} \bmod n$
4. Compute  $e = H(m)$ , where  $H$  is one of the SHA-2 or SHA-3 hash functions.
5. Compute  $s = k^{-1} (e + dr) \bmod n$ . If  $s = 0$  then go to step 1
6. The signature of message  $m$  is the pair  $(r, s)$ .

Alice knows the public domain parameters and Bob's public key. Alice is presented with Bob's message and digital signature and verifies the signature using the following steps:

1. Verify that  $r$  and  $s$  are integers in the range 1 through  $n - 1$
2. Using SHA, compute the 160-bit hash value  $e = H(m)$
3. Compute  $w = s^{-1} \bmod n$
4. Compute  $u_1 = ew$  and  $u_2 = rw$
5. Compute the point  $X = (x_1, y_1) = u_1G + u_2Q$
6. If  $X = O$ , reject the signature else compute  $v = x_1 \bmod n$
7. Accept Bob's signature if and only if  $v = r$

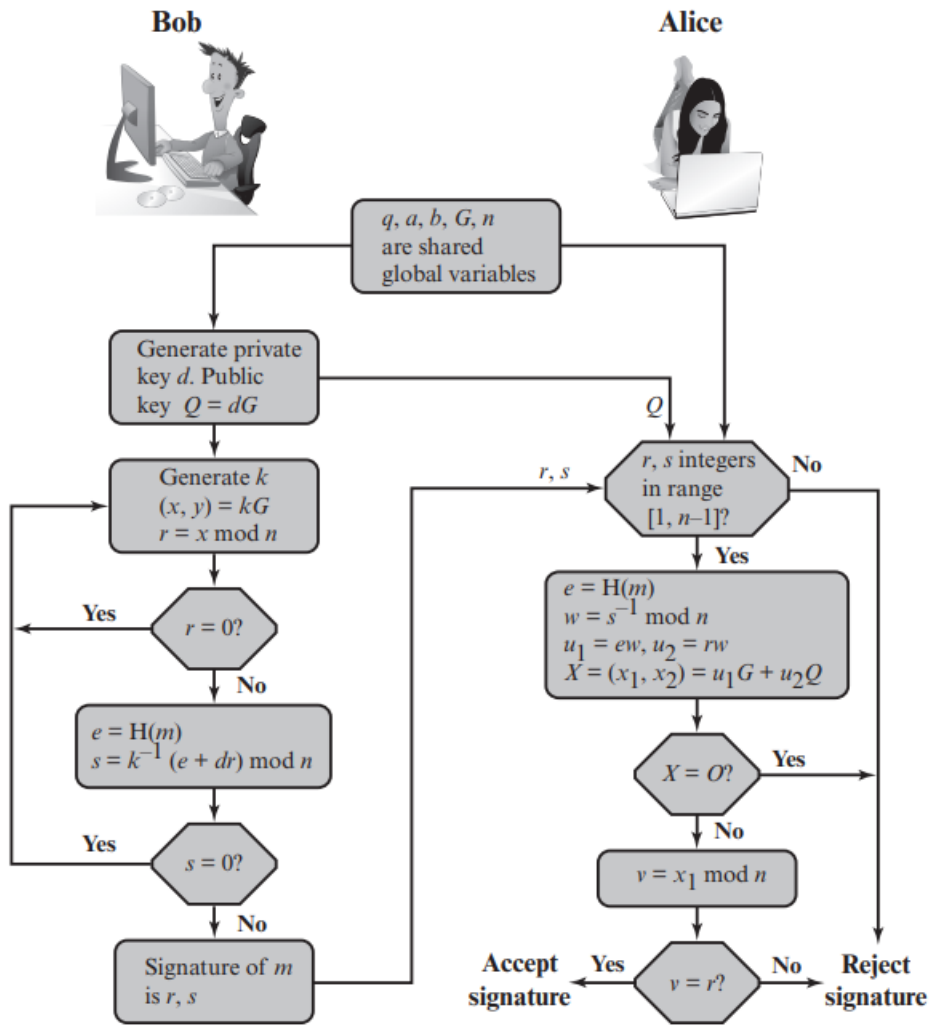
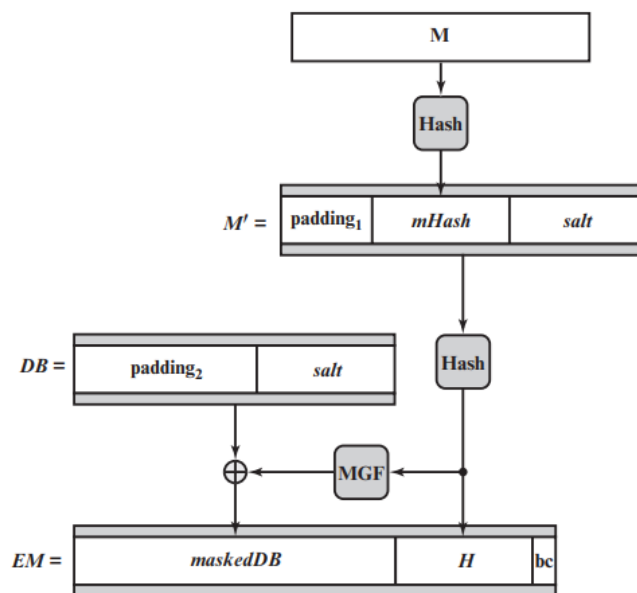


Figure 13.5 ECDSA Signing and Verifying

## 5. RSA-PSS Digital Signature Algorithm

### RSA-PSS Encoding



The encoding process consists of the following steps.

1. Generate the hash value of M:  $mHash = Hash(M)$
2. Generate a pseudorandom octet string salt and form block  

$$M' = padding_1 || mHash || salt$$
3. Generate the hash value of M':  $H = Hash(M')$
4. Form data block  $DB = padding_2 || salt$
5. Calculate the MGF value of H:  $dbMask = MGF(H, emLen - hLen - 1)$
6. Calculate  $maskedDB = DB \oplus dbMsk$
7. Set the leftmost  $8emLen - emBits$  bits of the leftmost octet in  $maskedDB$  to 0
8.  $EM = maskedDB || H || 0xbc$

<b>Options</b>	Hash	hash function with output $hLen$ octets. The current preferred alternative is SHA-1, which produces a 20-octet hash value.
	MGF	mask generation function. The current specification calls for MGF1.
	$sLen$	length in octets of the salt. Typically $sLen = hLen$ , which for the current version is 20 octets.
<b>Input</b>	$M$	message to be encoded for signing.
	$emBits$	This value is one less than the length in bits of the RSA modulus $n$ .
<b>Output</b>	$EM$	encoded message. This is the message digest that will be encrypted to form the digital signature.
<b>Parameters</b>	$emLen$	length of $EM$ in octets = $\lceil emBits/8 \rceil$ .
	$padding_1$	hexadecimal string 00 00 00 00 00 00 00 00; that is, a string of 64 zero bits.
	$padding_2$	hexadecimal string of 00 octets with a length $(emLen - sLen - hLen - 2)$ octets, followed by the hexadecimal octet with value 01.
	$salt$	a pseudorandom number.
	$bc$	the hexadecimal value BC.

### RSA-PSS EM Verification

1. Generate the hash value of M:  $mHash = Hash(M)$
2. If  $emLen < hLen + sLen + 2$ , output “inconsistent” and stop.
3. If the rightmost octet of EM does not have hexadecimal value BC, output “inconsistent” and stop.
4. Let  $maskedDB$  be the leftmost  $emLen - hLen - 1$  octets of EM, and let H be the next  $hLen$  octets .
5. If the leftmost  $8emLen - emBits$  bits of the leftmost octet in  $maskedDB$  are not all equal to zero, output “inconsistent” and stop
6. Calculate  $dbMask = MGF(H, emLen - hLen - 1)$
7. Calculate  $DB = maskedDB \oplus dbMsk$
8. Set the leftmost  $8emLen - emBits$  bits of the leftmost octet in DB to zero
9. If the leftmost  $(emLen - hLen - sLen - 1)$  octets of DB are not equal to  $padding_2$ ,

output "inconsistent" and stop

10. Let salt be the last sLen octets of DB

11. Form block  $M' = \text{padding}_1 || \text{mHash} || \text{salt}$

12. Generate the hash value of  $M'$ :  $H' = \text{Hash}(M')$

13. If  $H = H'$ , output "consistent." Otherwise, output "inconsistent"

