MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

PG DEPARTMENT OF COMPUTER SCIENCE

SUBJECT NAME: CRYPTOGRAPHY

SUBJECT CODE: PSDEE

SEMESTER: III

PREPARED BY: PROF. W.SHARMILA

Title of the Course/ Paper	Cryptography		
Elective - 2	II Year & Third Semester	Credit: 4	

Unit 1: Conventional Encryption: Conventional encryption model – DES –RC 5 – Introduction to AES Random number generation.

Unit-2: Number Theory: Modular arithmetic – Euler's theorem – Euclid's algorithm – Chinese remain theorem – Primarily and factorization –Discrete logarithms – RSA algorithm

Unit 3: Public key Cryptography: Principles – RSA algorithm – key management- Diff – Hellman key exchange

Unit 4: Message Authorization and Hash functions: Hash functions- Authentication requirements –

Authentication function- Message authentication codes –Secure Hash algorithms

Unit 5: Digital Signature and Authentication Protocols : Digital Signature- Authentication Protocol Digital signature standard.

Recommended Texts:

1) Stallings, W., 2005 , Cryptography and Network Security Principles and Practice, Pearson Educat Delhi.

Reference Books

1) Charlie Kaufman, Radia Perlman, Mike specimen, Network Security- Private Communication public wo

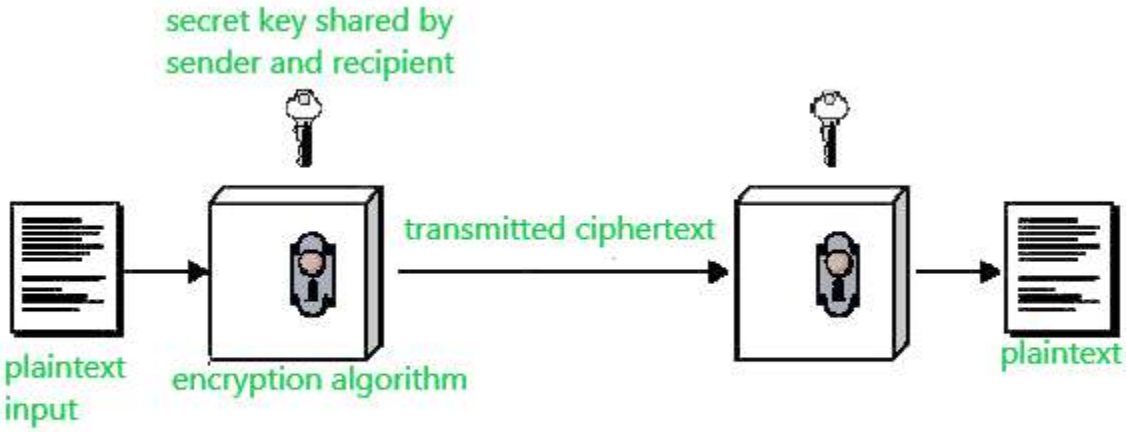
2) Michael Welsehenbach, 2005, Cryptography in C & C++", John Wi

UNIT I

Conventional Encryption

Conventional encryption are cryptographic system which uses same key used by sender to encrypt message and by receiver to decrypt message. It was only type of encryption in use prior to development of public-key encryption.

It is still much preferred of the two types of encryption systems due to its simplicity. It is a relatively fast process since it uses a single key for both encryption and decryption In this encryption model, the sender encrypts plaintext using receiver's secret key, which can be later used by receiver to decrypt ciphertext. Below is figure that illustrate this concept.



Suppose A wants to send a message to B, that message is called plaintext. Now, to avoid hackers to read plaintext, plaintext is encrypted using algorithm and a secret key (at 1). This encrypted plaintext is calledciphertext. Using same secret key and encryption algorithm run in reverse(at 2), B can get plaintext of A and thus message is read and security is maintained.

The idea that uses in this technique is very old and that's why this model is called conventional encryption.

A conventional encryption has mainly 5 ingredients :

1. **Plain text** –
   It is the original data that is given to the algorithm as a input
2. **Encryption algorithm** –
   This encryption algorithm performs various transformations on plain text to convert it into cipher text.
3. **Secret key** –
   Secret key is also an input to the algorithm. The encryption algorithm will produce different output based on the keys used at that time.
4. **Cipher text** –
   It contains encrypted information because it contains a form of original plaintext that is unreadable by a human or computer without proper cipher to decrypt it. It is output from algorithm.
5. **Decryption algorithm** –
   This is used to run encryption algorithm in reverse. Cipher text and Secret key is input here and it produces plain text as output.

Requirements for secure use of conventional encryption :

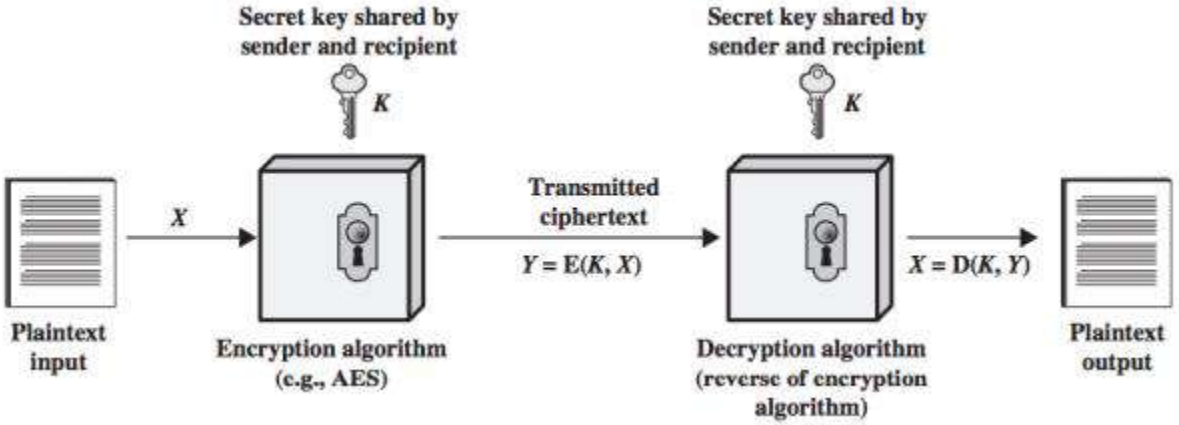
1. We need a strong encryption algorithm.
2. Sender and Receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure.

Advantages of Conventional Encryption :

1. **Simple** –
   This type of encryption is easy to carry out.
2. **Uses less computer resources** –
   Conventional encryption does not require a lot of computer resources when compared to public key encryption.
3. **Fast** –
   Conventional encryption is much faster than asymmetric key encryption.

Disadvantages of conventional encryption :

1. Origin and authenticity of message cannot be guaranteed, since both sender and receiver use the same key, messages cannot be verified to have come from a particular user.
2. It isn't much secured when compared to public key encryption.
3. If the receiver lost the key, he/she cant decrypt the message and thus making the whole process useless.
4. This scheme does not scale well to a large number of users because both the sender and the receiver have to agree on a secret key before transmission.



Encryption Requirements *

There are two requirements for secure use of conventional encryption:

The encryption algorithm must be strong.

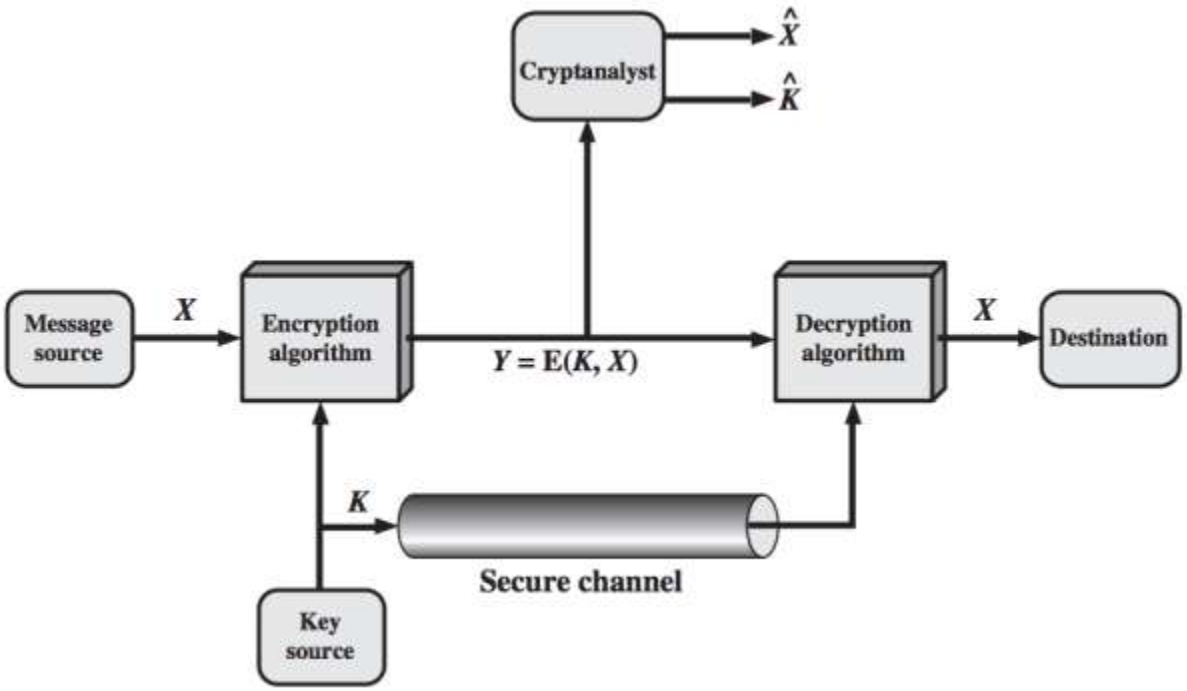
- At a minimum, an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key.
- In a stronger form, the opponent should be unable to decrypt ciphertexts or discover the key even if he or she has a number of ciphertexts together with the plaintext for each ciphertext.

2. Sender and receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communication using this key is readable.

We assume that it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decryption algorithm. This means we do not need to keep the algorithm secret; we need to keep only the key secret. This feature of symmetric encryption makes low-cost chip implementations of data encryption algorithms widely available and incorporated into a number of products. With the use of symmetric encryption, the principal security problem is maintaining the secrecy of the key.

Model of Symmetric Cryptosystem *

The essential elements of a symmetric encryption scheme is described in the following figure:



- A source produces a message in plaintext, X=[X1,X2,...,XM]X=[X1,X2,...,XM].
- A key of the form K=[K1,K2,...,KJ]K=[K1,K2,...,KJ] is generated.
    - If the key is generated at the message source, then it must also be provided to the destination by means of some secure channel.

- Alternatively, a third party could generate the key and securely deliver it to both source and destination.

- The ciphertext Y=[Y1,Y2,...,YN]Y=[Y1,Y2,...,YN] is produced by the encryption algorithm with the message *X* and the encryption key *K* as input.

The encryption process is:

Y=E(K,X)Y=E(K,X)

This notation indicates that *Y* is produced by using encryption algorithm E as a function of the plaintext *X*, with the specific function determined by the value of the key *K*.

The intended receiver with the key is able to invert the transformation:

X=D(K,Y)X=D(K,Y)

An opponent, observing *Y* but not having access to *K* or *X*, may attempt to recover *X* or *K* or both. It is assumed that the opponent knows the encryption (E) and decryption (D) algorithms. The opponent may do one of the following:

- Recover *X* by generating a plaintext estimate X^X^, if the opponent is interested in only this particular message.
- Recover *K* by generating an estimate K^K^, if the opponent is interested in being able to read future messages.

Cryptography

Cryptographic systems are characterized along three independent dimensions:

1. **Type of operations for transforming plaintext to ciphertext**. All encryption algorithms are based on two general principles:
    - **Substitution**: each element in the plaintext (bit, letter, group of bits or letters) is mapped into another element,
    - **Transposition**: elements in the plaintext are rearranged.

The fundamental requirement is that no information be lost (all operations are reversible). *Product systems* involve multiple stages of substitutions and transpositions.

2. **Number of keys used**.
    - If both sender and receiver use the same key, the system is referred to as symmetric, single-key, secret-key, or conventional encryption.
    - If the sender and receiver use different keys, the system is referred to as asymmetric, two-key, or public-key encryption.
3. **How the plaintext is processed**.
    - A block cipher processes the input one block of elements at a time, producing an output block for each input block.
    - A stream cipher processes the input elements continuously, producing output one element at a time, as it goes along.

Cryptanalysis and Brute-Force Attack

The objective of attacking an encryption system is to recover the key in use rather than simply to recover the plaintext of a single ciphertext. There are two general approaches to attacking a conventional encryption scheme:

- **Cryptanalysis** (cryptanalytic attacks): This attack relies on the nature of the algorithm plus some knowledge of the general characteristics of the plaintext or some sample plaintext–ciphertext pairs. It exploits the characteristics of the algorithm to attempt to deduce a specific plaintext or to deduce the key being used.
- **Brute-force attack**: The attacker tries every possible key on a piece of ciphertext until an intelligible translation into plaintext is obtained. On average, half of all possible keys must be tried to achieve success.

If either type of attack succeeds in deducing the key, then future and past messages encrypted with that key are compromised.

Cryptanalytic attacks *

The following table summarizes the various types of cryptanalytic attacks based on the amount of information known to the cryptanalyst.

Type of Attack	Known to Cryptanalyst
Ciphertext Only	• Encryption algorithm • Ciphertext
Known Plaintext	• Encryption algorithm • Ciphertext • One or more plaintext–ciphertext pairs formed with the secret key
Chosen Plaintext	• Encryption algorithm • Ciphertext • Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key
Chosen Ciphertext	• Encryption algorithm • Ciphertext • Ciphertext chosen by

Type of Attack	Known to Cryptanalyst
	cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key
Chosen Text	Combination of "Chosen Plaintext" and "Chosen Ciphertext"

In general, we can assume that the opponent does know the algorithm used for encryption. If the key space is very large, the brute-force approach of trying all possible keys, which is one possible attack, becomes impractical. Thus, the opponent must anaylyze the ciphertext itself, applying various statistical tests to it. To use this approach, the opponent must have some general idea of the type of plaintext that is concealed.

Ciphertext-only attack *

The ciphertext-only attack is the easiest to defend against because the opponent has the least amount of information to work with.

Known-plaintext attack *

In many cases, the analyst has more information than ciphertext only:

- The analyst may be able to capture one or more plaintext messages and their encryptions.
- The analyst may know that certain plaintext patterns will appear in a message.

For example, a file that is encoded in the Postscript format always begins with the same pattern, or there may be a standardized header or banner to an electronic funds transfer message. All these are examples of *known plaintext*. With this knowledge, the analyst may be able to deduce the key on the basis of the way in which the known plaintext is transformed. This is known as the known-plaintext attack.

Probable-word attack *

The probable-word attack is closely related to the known-plaintext attack. If the opponent is working with the encryption of some general prose message, he or she may have little knowledge of what is in the message. However, if the opponent is after some very specific information, then parts of the message may be known.

For example:

- If an entire accounting file is being transmitted, the opponent may know the placement of certain key words in the header of the file.
- The source code for a program developed by a company might include a copyright statement in some standardized position.

Chosen-plaintext attack *

If the analyst is able to get the source system to insert into the system a message chosen by the analyst, then a chosen-plaintext attack is possible. An example of this strategy is differential cryptanalysis, explored in Chapter 3. In general, if the analyst is able to choose the messages to encrypt, the analyst may deliberately pick patterns that can be expected to reveal the structure of the key.

The other two types of attack: chosen ciphertext and chosen text, are less commonly employed as cryptanalytic techniques but are nevertheless possible avenues of attack.

Generally, an encryption algorithm is designed to withstand a known-plaintext attack; only weak algorithms fail to withstand a ciphertext-only attack.

- The cost of breaking the cipher exceeds the value of the encrypted information.
- The time required to break the cipher exceeds the useful lifetime of the information.

Brute-force attack *

A **brute-force attack** involves trying every possible key until an intelligible translation of the ciphertext into plaintext is obtained.

On average, half of all possible keys must be tried to achieve success. A brute-force attack is more than simply running through all possible keys. Unless known plaintext is provided, the analyst must be able to recognize plaintext as plaintext:

- If the message is just plain text in English, then the result pops out easily, although the task of recognizing English would have to be automated.
- If the text message has been compressed before encryption, then recognition is more difficult.
- If the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate.

Thus, to supplement the brute-force approach, some degree of knowledge about the expected plaintext is needed, and some means of automatically distinguishing plaintext from garble is also needed.

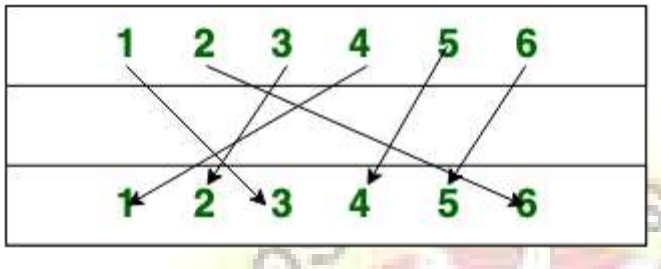
Difference between Substitution Cipher Technique and Transposition Cipher Technique

Both **Substitution cipher technique** and **Transposition cipher technique** are the types of Traditional cipher which are used to convert the plain text into cipher text.

Substitution Cipher Technique:
In Substitution Cipher Technique plain text characters are replaced with other characters, numbers and symbols as well as in substitution Cipher Technique, character's identity is changed while its position remains unchanged.

Transposition Cipher Technique:
Transposition Cipher Technique rearranges the position of the plain text's characters. In

transposition Cipher Technique, The position of the character is changed but character's identity is not changed.



Transportation Cipher

Difference between Substitution Cipher Technique and Transposition Cipher Technique:

S.NO	Substitution Cipher Technique	Transposition Cipher Technique
1.	In substitution Cipher Technique, plain text characters are replaced with other characters, numbers and symbols.	In transposition Cipher Technique, plain text characters are rearranged with respect to the position.
2.	Substitution Cipher's forms are: Mono alphabetic substitution cipher and poly alphabetic substitution cipher.	Transposition Cipher's forms are: Key-less transposition cipher and keyed transposition cipher.
3.	In substitution Cipher Technique, character's identity is changed while its position remains unchanged.	While in transposition Cipher Technique, The position of the character is changed but character's identity is not changed.
4.	In substitution Cipher Technique, The letter with low frequency can detect plain text.	While in transposition Cipher Technique, The Keys which are nearer to correct key can disclose plain text.
5.	The example of substitution Cipher is Caesar Cipher.	The example of transposition Cipher is Reil Fence Cipher.

Substitution Cipher

Hiding some data is known as encryption. When plain text is encrypted it becomes unreadable and is known as ciphertext. In a Substitution cipher, any character of plain text from the given fixed set of characters is substituted by some other character from the same set depending on a key. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

Caesar Cipher Technique

The Caesar cipher is the simplest and oldest method of cryptography. The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. Julius Caesar used the shift cipher (additive cipher) technique to communicate with his officers. For this reason, the shift cipher technique is called the Caesar cipher. The Caesar cipher is a kind of replacement (substitution) cipher, where all letter of plain text is replaced by another letter.

Let's take an example to understand the Caesar cipher, suppose we are shifting with 1, then A will be replaced by B, B will be replaced by C, C will be replaced by D, D will be replaced by C, and this process continues until the entire plain text is finished.

Caesar ciphers is a weak method of cryptography. It can be easily hacked. It means the message encrypted by this method can be easily decrypted.

Plaintext: It is a simple message written by the user.

Ciphertext: It is an encrypted message after applying some technique.

The formula of encryption is:

$E_n(x) = (x + n) \bmod 26$

The formula of decryption is:

$D_n(x) = (xi - n) \bmod 26$

If any case (Dn) value becomes negative (-ve), in this case, we will add 26 in the negative value.

Where,

E denotes the encryption

D denotes the decryption

x denotes the letters value

n denotes the key value (shift value)

Note: "i" denotes the offset of the ith number of the letters, as shown in the table below.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Example: 1 Use the Caesar cipher to encrypt and decrypt the message "JAVATPOINT," and the key (shift) value of this message is 3.

Encryption

We apply encryption formulas by character, based on alphabetical order.

The formula of encryption is:

$E_n(x) = (x + n) \bmod 26$

Plaintext: J → 09	E_n: (09 + 3) mod 26	Ciphertext: 12 → M
Plaintext: A → 00	E_n: (00 + 3) mod 26	Ciphertext: 3 → D
Plaintext: V → 21	E_n: (21 + 3) mod 26	Ciphertext: 24 → Y
Plaintext: A → 00	E_n: (00 + 3) mod 26	Ciphertext: 3 → D

Plaintext: T → 19	E_n: (19 + 3) mod 26	Ciphertext: 22 → W
Plaintext: P → 15	E_n: (15 + 3) mod 26	Ciphertext: 18 → S
Plaintext: O → 14	E_n: (14 + 3) mod 26	Ciphertext: 17 → R
Plaintext: I → 08	E_n: (08 + 3) mod 26	Ciphertext: 11 → L
Plaintext: N → 13	E_n: (13 + 3) mod 26	Ciphertext: 16 → Q
Plaintext: T → 19	E_n: (19 + 3) mod 26	Ciphertext: 22 → W

The encrypted message is "MDYDWSRLQW". Note that the Caesar cipher is monoalphabetic, so the same plaintext letters are encrypted as the same letters. For example, "JAVATPOINT" has "A", encrypted by "D".

Decryption

We apply decryption formulas by character, based on alphabetical order.

The formula of decryption is:

$D_n (x) = (x_i - n) \bmod 26$

If any case (D_n) value becomes negative (-ve), in this case, we will add 26 in the negative value.

Ciphertext: M → 12	D_n: (12 - 3) mod 26	Plaintext: 09 → J
Ciphertext: D → 03	D_n: (03 - 3) mod 26	Plaintext: 0 → A

Ciphertext: Y → 24	D_n: (24 - 3) mod 26	Plaintext: 21 → V
Plaintext: A → 00	E_n: (00 + 3) mod 26	Ciphertext: 3 → D
Plaintext: T → 19	E_n: (19 + 3) mod 26	Ciphertext: 22 → W
Plaintext: P → 15	E_n: (15 + 3) mod 26	Ciphertext: 18 → S
Plaintext: O → 14	E_n: (14 + 3) mod 26	Ciphertext: 17 → R
Plaintext: I → 08	E_n: (08 + 3) mod 26	Ciphertext: 11 → L
Plaintext: N → 13	E_n: (13 + 3) mod 26	Ciphertext: 16 → Q
Plaintext: T → 19	E_n: (19 + 3) mod 26	Ciphertext: 22 → W

The decrypted message is "JAVATPOINT".

Example: 2 Use the Caesar cipher to encrypt and decrypt the message "HELLO," and the key (shift) value of this message is 15.

Encryption

We apply encryption formulas by character, based on alphabetical order.

The formula of encryption is:

E_n (x) = (x + n) mod 26

Plaintext: H → 07	E_n: (07 + 15) mod 26	Ciphertext: 22 → W

Plaintext: E → 04	E_n: (04 + 15) mod 26	Ciphertext: 19 → T
Plaintext: L → 11	E_n: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: L → 11	E_n: (11 + 15) mod 26	Ciphertext: 00 → A
Plaintext: O → 14	E_n: (14 + 15) mod 26	Ciphertext: 03 → D

Note that the Caesar cipher is monoalphabetic, so the same plaintext letters are encrypted as the same letters. Like, "HELLO" has "L", encrypted by "A".

The encrypted message of this plain text is "WTAAD".

Decryption

We apply decryption formulas by character, based on alphabetical order.

The formula of decryption is:

D_n (x) = (xi - n) mod 26

Ciphertext: W → 22	D_n: (22 - 15) mod 26	Plaintext: 07 → H
Ciphertext: T → 19	D_n: (19 - 15) mod 26	Plaintext: 04 → E
Ciphertext: A → 00	D_n: (00 - 15) mod 26	Plaintext: 11 → L
Ciphertext: A → 00	D_n: (00 - 15) mod 26	Plaintext: 11 → L
Ciphertext: D → 03	D_n: (03 - 15) mod 26	Plaintext: 14 → O

The decrypted message is "HELLO".

Note: If any case (D_n) value becomes negative (-ve), in this case, we will add 26 in the negative value. Like, the third letter of the ciphertext.

D_n = (00 - 15) mod 26

= -15

The value of dn is negative, so 26 will be added to it.

= -15 + 26

= 11

Advantages of Caesar cipher

Its benefits are as follows: -

1. It is very easy to implement.
2. This method is the simplest method of cryptography.
3. Only one short key is used in its entire process.
4. If a system does not use complex coding techniques, it is the best method for it.
5. It requires only a few computing resources.

Disadvantages of Caesar cipher

Its disadvantages are as follows: -

1. It can be easily hacked. It means the message encrypted by this method can be easily decrypted.
2. It provides very little security.
3. By looking at the pattern of letters in it, the entire message can be decrypted.

Monoalphabetic and Polyalphabetic Cipher

Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if 'A' is encrypted as 'D', for any number of occurrence in that plaintext, 'A' will always get encrypted to 'D'.

All of the substitution ciphers we have discussed earlier in this chapter are monoalphabetic; these ciphers are highly susceptible to cryptanalysis.

Polyalphabetic Cipher is a substitution cipher in which the cipher alphabet for the plain alphabet may be different at different places during the encryption process. The next two examples, **playfair and Vigenere Cipher are polyalphabetic ciphers**.

Playfair Cipher

In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution cipher.

In playfair cipher, initially a key table is created. The key table is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table as we need only 25 alphabets instead of 26. If the plaintext contains J, then it is replaced by I.

The sender and the receiver deicide on a particular key, say 'tutorials'. In a key table, the first characters (going left to right) in the table is the phrase, excluding the duplicate letters. The rest of the table will be filled with the remaining letters of the alphabet, in natural order. The key table works out to be −

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

Process of Playfair Cipher

- First, a plaintext message is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter. Let us say we want to encrypt the message "hide money". It will be written as −

  HI DE MO NE YZ

- The rules of encryption are −
  - If both the letters are in the same column, take the letter below each one (going back to the top if at the bottom)

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

'H' and 'I' are in same column, hence take letter below them to replace. HI → QC

- If both letters are in the same row, take the letter to the right of each one (going back to the left if at the farthest right)

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H

'D' and 'E' are in same row, hence take letter to the right of them to replace. DE → EF

K	M	N	P	Q
V	W	X	Y	Z

- If neither of the preceding two rules are true, form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

T	U	O	R	I
A	L	S	B	C
D	E	F	G	H
K	M	N	P	Q
V	W	X	Y	Z

'M' and 'O' nor on same column or same row, hence form rectangle as shown, and replace letter by picking up opposite corner letter on same row
MO -> NU

Using these rules, the result of the encryption of 'hide money' with the key of 'tutorials' would be –

QC EF NU MF ZV

Decrypting the Playfair cipher is as simple as doing the same process in reverse. Receiver has the same key and can create the same key table, and then decrypt any messages made using that key.

Security Value

It is also a substitution cipher and is difficult to break compared to the simple substitution cipher. As in case of substitution cipher, cryptanalysis is possible on the Playfair cipher as well, however it would be against 625 possible pairs of letters (25x25 alphabets) instead of 26 different possible alphabets.

The Playfair cipher was used mainly to protect important, yet non-critical secrets, as it is quick to use and requires no special equipment.

Hill Cipher

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

Examples:

Input : Plaintext: ACT

Key: GYBNQKURP

Output :Ciphertext: POH

Input : Plaintext: GFG

Key: HILLMAGIC

Output :Ciphertext: SWK

Encryption

We have to encrypt the message ‘ACT’ (n=3).The key is ‘GYBNQKURP’ which can be written as the nxn matrix:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

The message ‘ACT’ is written as vector:

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

The enciphered vector is given as:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} (\text{mod } 26)$$

which corresponds to ciphertext of 'POH'

Decryption

To decrypt the message, we turn the ciphertext back into a vector, then simply multiply by the inverse matrix of the key matrix (IFKVIVVMI in letters).The inverse of the matrix used in the previous example is:

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} (\text{mod } 26)$$

For the previous Ciphertext 'POH':

$$\begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \equiv \begin{bmatrix} 260 \\ 574 \\ 539 \end{bmatrix} \equiv \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \pmod{26}$$

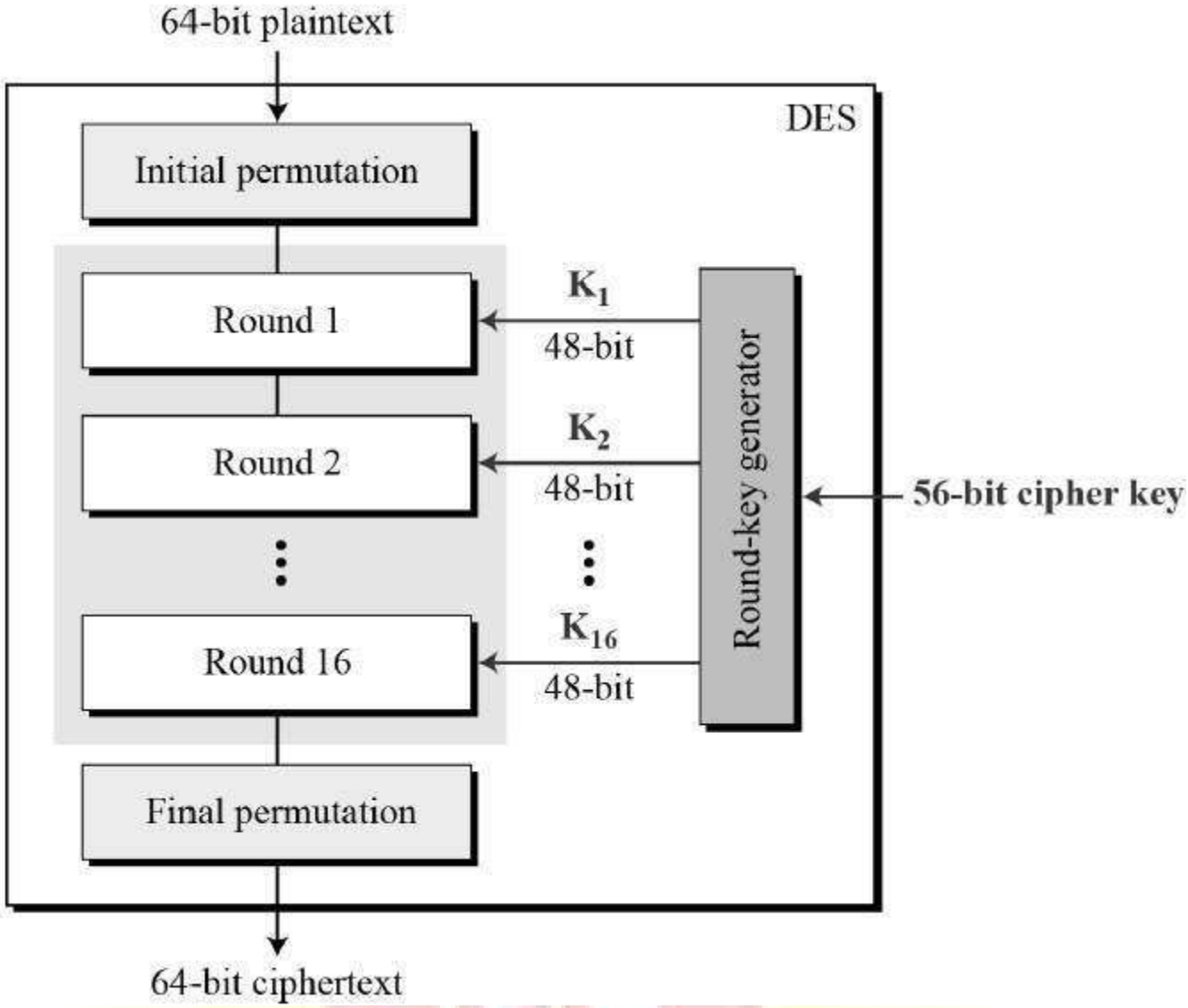
which gives us back 'ACT'.

Assume that all the alphabets are in upper case.

Below is the the implementation of the above idea for n=3.

he Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration −

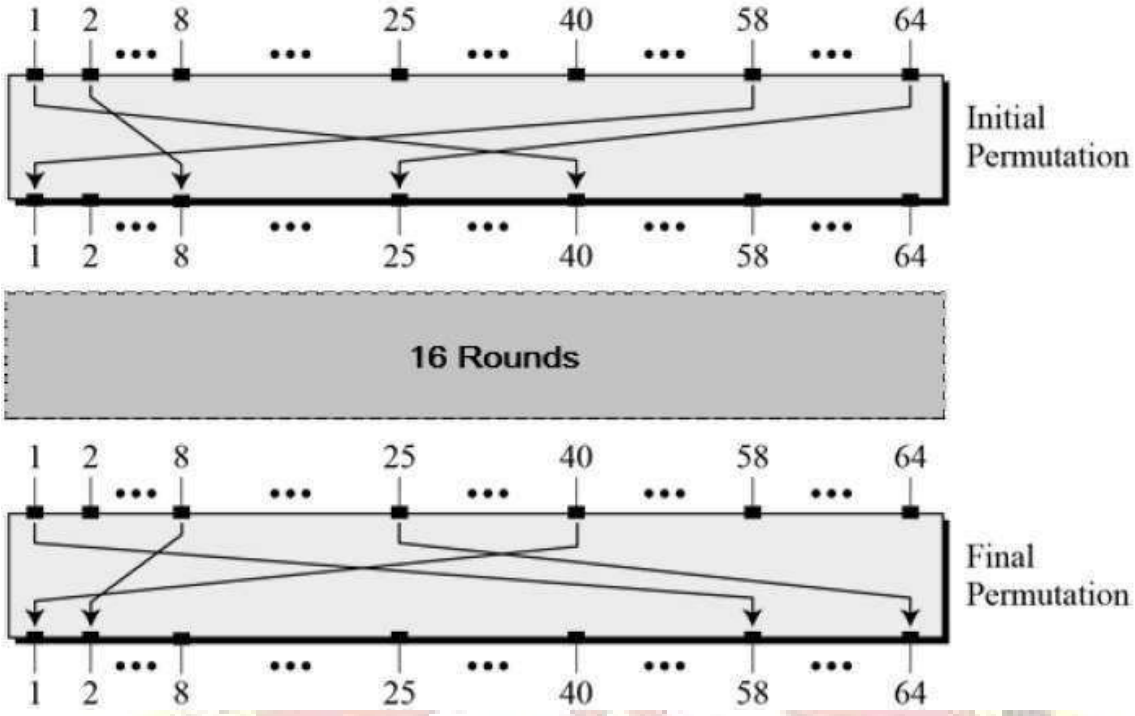


Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation

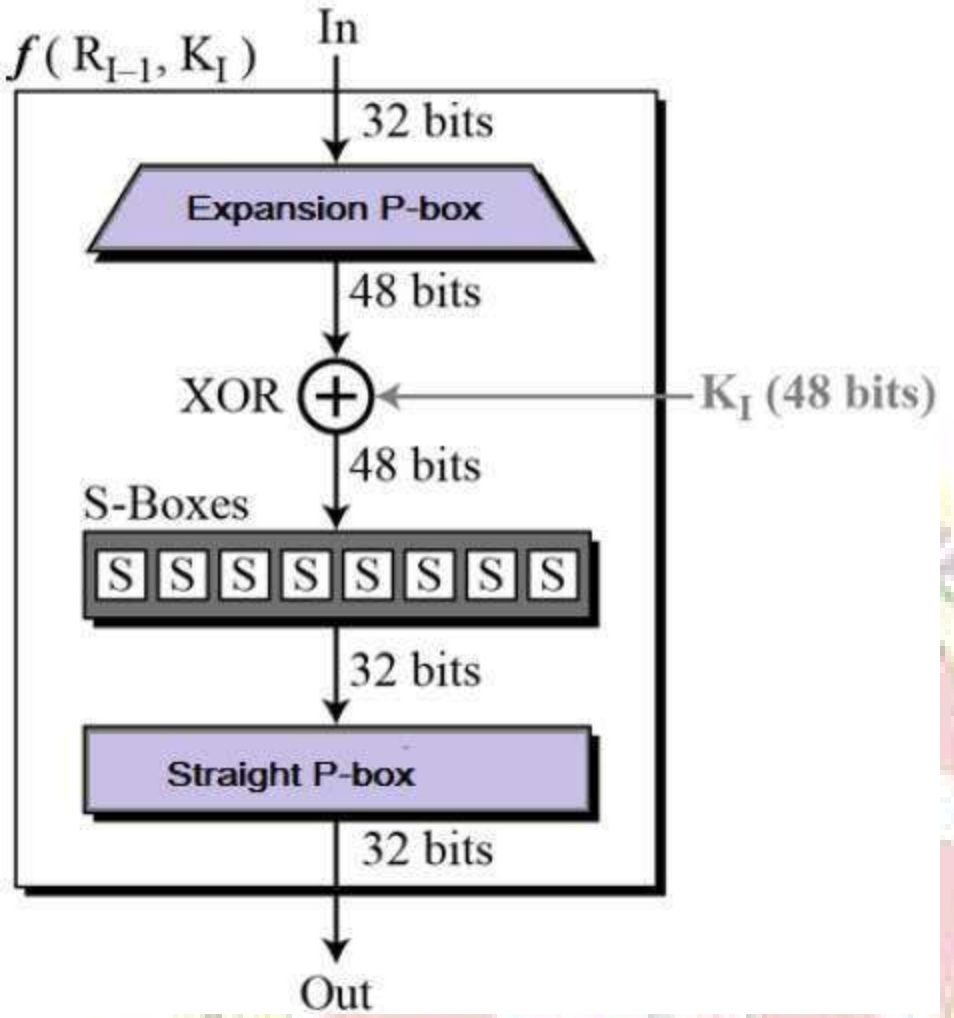
Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The initial and final permutations are shown as follows −

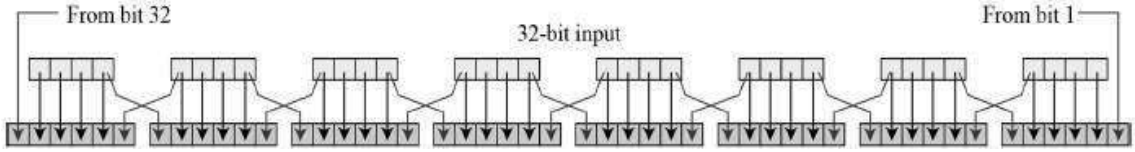


Round Function

The heart of this cipher is the DES function, *f*. The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



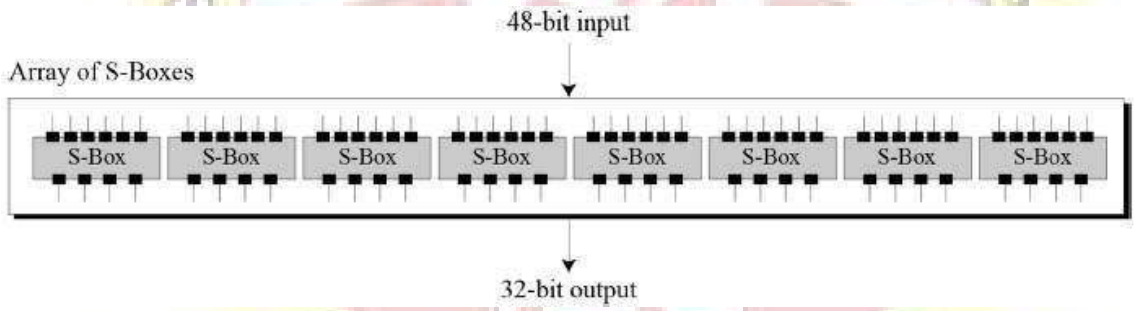
- **Expansion Permutation Box** − Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration −



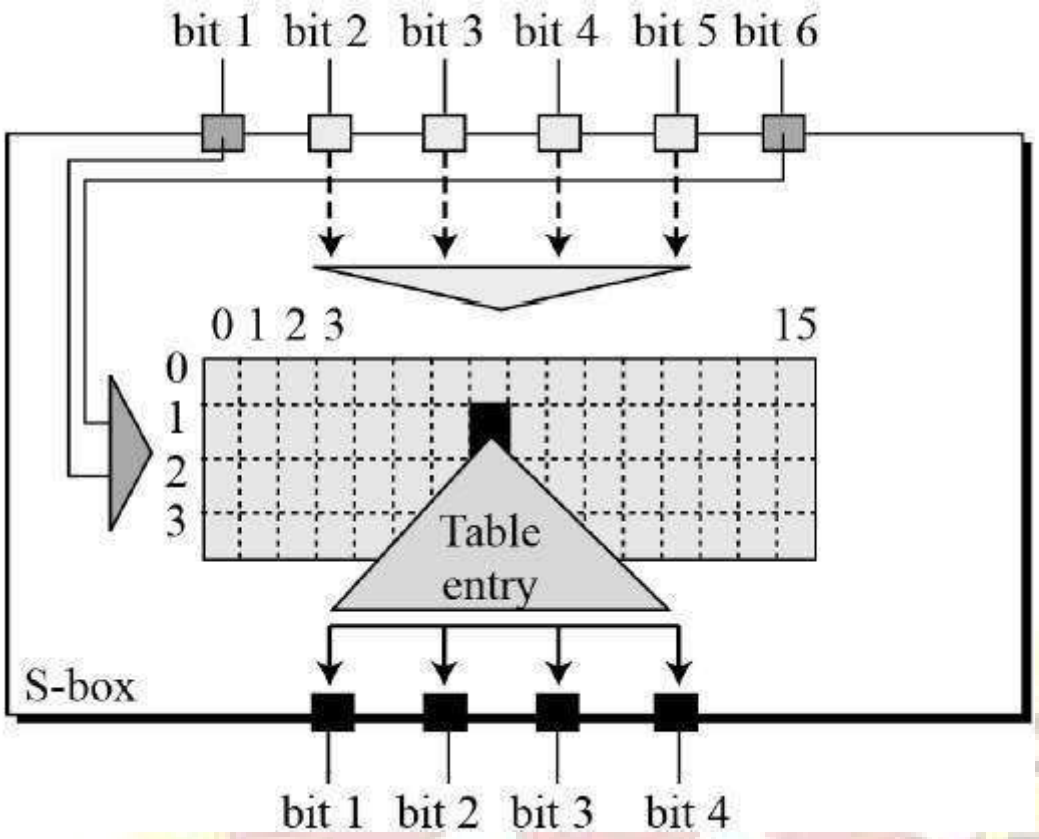
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown −

32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

- **XOR (Whitener).** − After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
- **Substitution Boxes.** − The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration −



- The S-box rule is illustrated below −

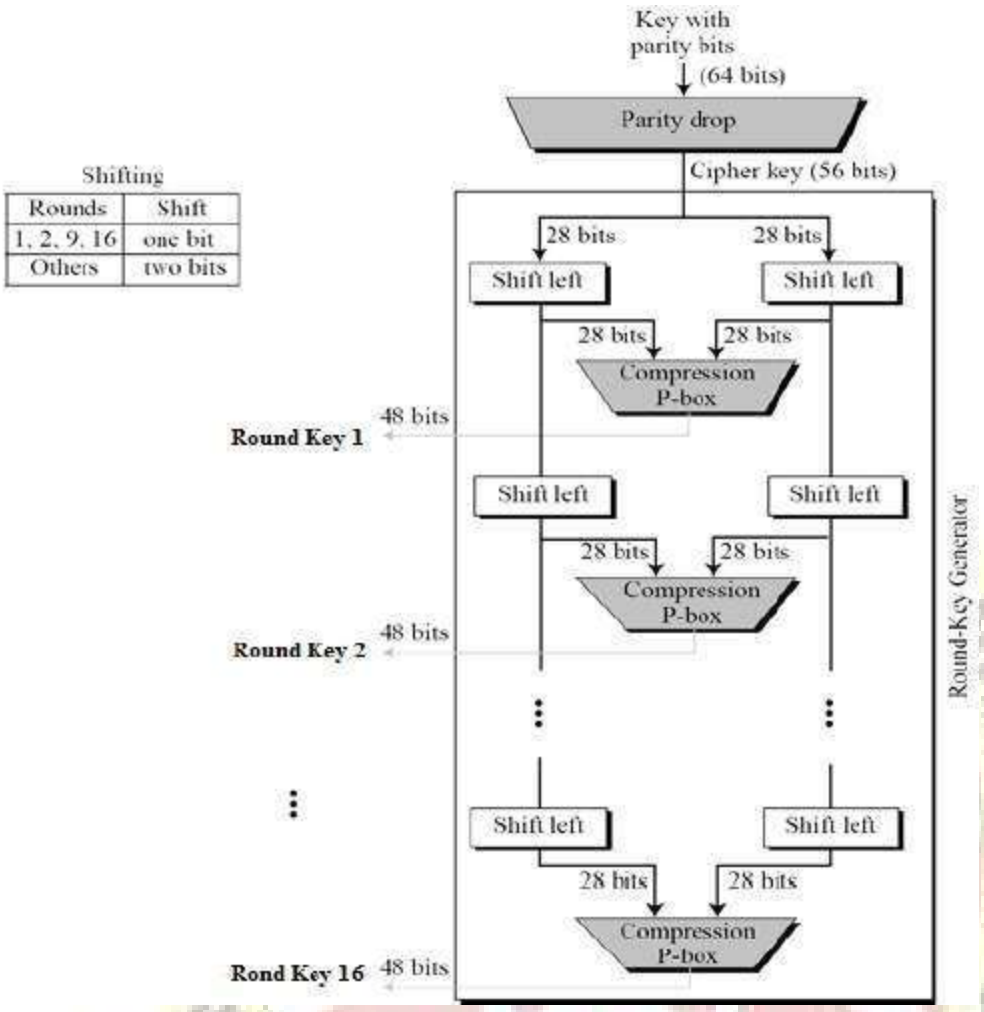


- There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.
- **Straight Permutation** − The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration −



The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

DES Analysis

The DES satisfies both the desired properties of block cipher. These two properties make cipher very strong.

- **Avalanche effect** − A small change in plaintext results in the very great change in the ciphertext.
- **Completeness** − Each bit of ciphertext depends on many bits of plaintext.

During the last few years, cryptanalysis have found some weaknesses in DES when key selected are weak keys. These keys shall be avoided.

DES has proved to be a very well designed block cipher. There have been no significant cryptanalytic attacks on DES other than exhaustive key search.

The more popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six time faster than triple DES.

A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows −

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

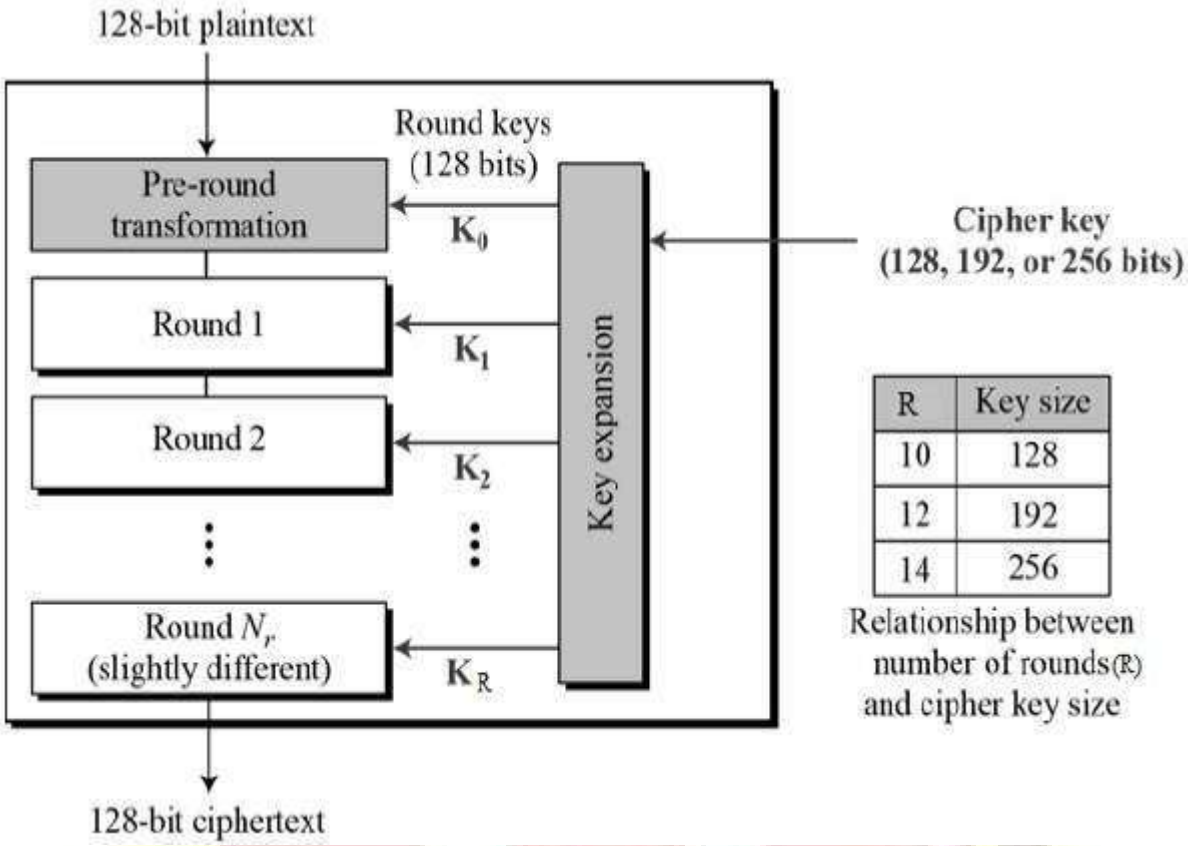
Operation of AES

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

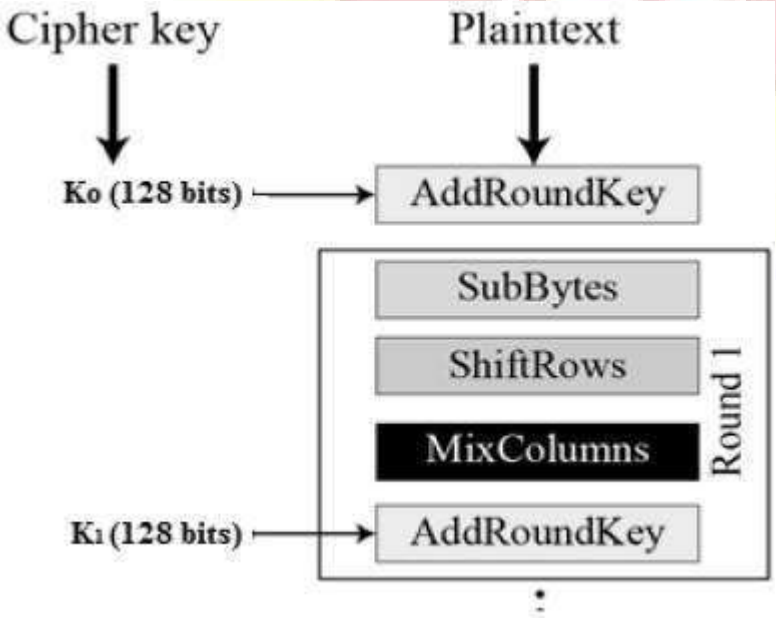


R	Key size
10	128
12	192
14	256

Relationship between number of rounds (R) and cipher key size

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns

- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

AES Analysis

In present day cryptography, AES is widely adopted and supported in both hardware and software. Till date, no practical cryptanalytic attacks against AES has been discovered. Additionally, AES has built-in flexibility of key length, which allows a degree of 'future-proofing' against progress in the ability to perform exhaustive key searches.

However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

RC5 Encryption Algorithm

RC5 is a symmetric key block encryption algorithm designed by Ron Rivest in 1994. It is notable for being simple, fast (on account of using only primitive computer operations like XOR, shift, etc.) and consumes less memory.

Example:

Key : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Plain Text : 00000000 00000000
Cipher Text : EEDBA521 6D8F4B15

RC5 is a block cipher and addresses two word blocks at a time. Depending on input plain text block size, number of rounds and key size, various instances of RC5 can be defined and each instance is denoted as RC5-w/r/b where w=word size in bits, r=number of rounds and b=key size in bytes. Allowed values are:

Parameter	Possible Value

Parameter	Possible Value
block/word size (bits)	16, 32, 64
Number of Rounds	0 – 255
Key Size (bytes)	0 – 255

Note – Since at a time, RC5 uses 2 word blocks, the plain text block size can be 32, 64 or 128 bits.

Notation used in the algorithm:

Symbol	Operation
x <<< y	Cyclic left shift of x by y bits
+	Two's complement addition of words where addition is modulo 2^w
^	Bit wise Exclusive-OR

Step-1: Initialization of constants P and Q.

RC5 makes use of 2 magic constants P and Q whose value is defined by the word size w.

Word Size (bits)	P (Hexadecimal)	Q (Hexadecimal)
16	b7e1	9e37
32	b7e15163	9e3779b9
64	b7e151628aed2a6b	9e3779b97f4a7c15

For any other word size, P and Q can be determined as:

P = Odd((e-2) 2^w)

Q = Odd((ϕ-2) 2^w)

Here, Odd(x) is the odd integer nearest to x, e is the base of natural logarithms and ϕ is the golden ratio.

Step-2: Converting secret key K from bytes to words.
Secret key K of size b bytes is used to initialize array L consisting of c words where c = b/u, u = w/8 and w = word size used for that particular instance of RC5. For example, if we choose w=32 bits and Key k is of size 96 bytes then, u=32/8=4, c=b/u=96/4=24. L is pre initialized to 0 value before adding secret key K to it.

```
fori=b-1 to 0
L[i/u] = (L[u/i] <<< 8) + K[i]
```

Step-3: Initializing sub-key S.
Sub-key S of size t=2(r+1) is initialized using magic constants P and Q.

```
S[0] = P
fori = 1 to 2(r+1)-1
   S[i] = S[i-1] + Q)
```

Step-4: Sub-key mixing.
The RC5 encryption algorithm uses Sub key S. L is merely, a temporary array formed on the basis of user entered secret key.
Mix in user's secret key with S and L.

```
i = j = 0
A = B = 0
do 3 * max(t, c) times:
   A = S[i] = (S[i] + A + B) <<< 3
   B = L[j] = (L[j] + A + B) <<< (A + B)
i = (i + 1) % t
   j = (j + 1) % c
```

Step-5: Encryption.

We divide the input plain text block into two registers A and B each of size w bits. After undergoing the encryption process the result of A and B together forms the cipher text block.

RC5 Encryption Algorithm:

1. One time initialization of plain text blocks A and B by adding S[0] and S[1] to A and B respectively. These operations are mod 2^w.
2. XOR A and B. A=A^B
3. Cyclic left shift new value of A by B bits.
4. Add S[2*i] to the output of previous step. This is the new value of A.
5. XOR B with new value of A and store in B.
6. Cyclic left shift new value of B by A bits.
7. Add S[2*i+1] to the output of previous step. This is the new value of B.
8. Repeat entire procedure (except one time initialization) r times.

```
A = A + S[0]
B = B + S[1]
fori = 1 to r do:
   A = ((A ^ B) <<< B) + S[2 * i]
   B = ((B ^ A) <<< A) + S[2 * i + 1]
return A, B
```

Alternatively, RC5 Decryption can be defined as:

```
fori = r down to 1 do:
   B = ((B - S[2 * i + 1]) >>> A) ^ A
   A = ((A - S[2 * i]) >>> B) ^ B
B = B - S[1]
A = A - S[0]
return A, B
```

Pseudo Random Number Generator (PRNG)

Pseudo Random Number Generator(PRNG) refers to an algorithm that uses mathematical formulas to produce sequences of random numbers. PRNGs generate a sequence of numbers approximating the properties of random numbers.

A PRNG starts from an arbitrary starting state using a **seed state**. Many numbers are generated in a short time and can also be reproduced later, if the starting point in the sequence is known. Hence, the numbers are **deterministic and efficient**.

Linear Congruential Generator is most common and oldest algorithm for generating pseudo-randomized numbers. The generator is defined by the recurrence relation:

$\mathbf{X_{n+1} = (aX_n + c) \bmod m}$

where X is the sequence of pseudo-random values

m, $0 < m$ - modulus

a, $0 < a < m$ - multiplier

c, $0 \leq c < m$ - increment

x_0, $0 \leq x_0 < m$ - the seed or start value

We generate the next random integer using the previous random integer, the integer constants, and the integer modulus. To get started, the algorithm requires an initial Seed, which must be provided by some means. The appearance of randomness is provided by performing**modulo arithmetic.**.

Characteristics of PRNG

- **Efficient:** PRNG can produce many numbers in a short time and is advantageous for applications that need many numbers
- **HiDeterministic:** A given sequence of numbers can be reproduced at a later date if the starting point in the sequence is known.Determinism is handy if you need to replay the same sequence of numbers again at a later stage.
- **Periodic:** PRNGs are periodic, which means that the sequence will eventually repeat itself. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes

Applications of PRNG

PRNGs are suitable for applications where many random numbers are required and where it is useful that the same sequence can be replayed easily. Popular examples of such applications are **simulation and modeling applications**. PRNGs are not suitable for applications where it is important that the numbers are really unpredictable, such as **data encryption and gambling.**

Steganography is the technique of hiding secret data within an ordinary, non-secret, file or message in order to avoid detection; the secret data is then extracted at its destination. The use of steganography can be combined with encryption as an extra step for hiding or protecting data. The word *steganography*is derived from the Greek words *steganos*(meaning *hidden* or c*overed*) and the Greek root *graph* (meaning *to write*).

Steganography can be used to conceal almost any type of digital content, including text, image, video or audio content; the data to be hidden can be hidden inside almost any other type of digital content. The content to be concealed through steganography -- called *hidden text* -- is often encrypted before being incorporated into the innocuous-seeming *cover text* file or data stream. If not encrypted, the hidden text is commonly processed in some way in order to increase the difficulty of detecting the secret content.

Steganography techniques

In modern digital steganography, data is first encrypted or obfuscated in some other way and then inserted, using a special algorithm, into data that is part of a particular file format such as a JPEGimage, audio or video file. The secret message can be embedded into ordinary data files in many different ways. One technique is to hide data in bits that represent the same color pixels repeated in a row in an image file. By applying the encrypted data to this redundant data in some inconspicuous way, the result will be an image file that appears identical to the original image but that has "noise" patterns of regular, unencrypted data.

The practice of adding a watermark -- a trademark or other identifying data hidden in multimedia or other content files -- is one common use of steganography. Watermarking is a technique often used by online publishers to identify the source of media files that have been found being shared without permission.

While there are many different uses of steganography, including embedding sensitive information into file types, one of the most common techniques is to embed a text file into an image file. When this is done, anyone viewing the image file should not be able to see a difference between the original image file and the encrypted file; this is accomplished by storing the message with less significant bites in the data file. This process can be completed manually or with the use of a steganography tool.

Advantages over cryptography

Steganography is distinct from cryptography, but using both together can help improve the security of the protected information and prevent detection of the secret communication. If steganographically-hidden data is also encrypted, the data may still be safe from detection -- though the channel will no longer be safe from detection. There are advantages to using steganography combined with encryption over encryption-only communication.

The primary advantage of using steganography to hide data over encryption is that it helps obscure the fact that there is sensitive data hidden in the file or other content carrying the hidden text. Whereas an encrypted file, message or network packet payload is clearly marked and identifiable as such, using steganographic techniques helps to obscure the presence of the secure channel.

Steganography software

Steganography software is used to perform a variety of functions in order to hide data, including encoding the data in order to prepare it to be hidden inside another file, keeping track of which bits of the cover text file contain hidden data, encrypting the data to be hidden and extracting hidden data by its intended recipient.

There are proprietary as well as open source and other free-to-use programs available for doing steganography. OpenStego is an open source steganography program; other programs can be characterized by the types of data that can be hidden as well as what types of files that data can be hidden inside. Some online steganography software tools include Xiao Steganography, used to hide secret files in BMP images or WAV files; Image Steganography, a Javascript tool that hides images inside other image files; and Crypture, a command line tool that is used to perform steganography

UNIT II

Number Theory and Cryptography

Modular Arithmetic (Clock Arithmetic)

Modular arithmetic is a system of arithmetic for integers, where values reset to zero and begin to increase again, after reaching a certain predefined value, called the modulus (*modulo*). Modular arithmetic is widely used in computer science and cryptography.

Definition Let Z_N be a set of all non-negative integers that are smaller than N:

$Z_N = \{0,1,2,...,N-1\}$

where:

- ***N** is a positive integer,*
- *if **N** is a prime, it will be denoted **p** (and the whole set as $\boldsymbol{Z_p}$).*

To determine the value of an integer for a modulus N, one should divide this number by N. Its value in Z_N is equal to the remainder of the division. In modular arithmetic, it is possible to define all typical operations, as in normal arithmetic. They work as one may expect. It is possible to use the same commutative, associative, and distributive laws.

Modular inversion

Integers in modular arithmetic may (but not must) have inverse numbers.

Definition The inverse of x in Z_N is a number y in Z_N, that satisfies the equation:

$x \cdot y = 1$ (in Z_N)

where:

- ***y** is denoted **x^{-1}***

For example, if N is an odd number, then the inverse of 2 in Z_N is (N+1)/2:

$x \cdot (N+1)/2 = N + 1 = 1$ (in Z_N)

Theorem A number x is invertible in Z_N if and only if the numbers x and N are relatively prime.

- *The theorem can be proved using the fact that it is possible to present the greatest common divisor of two integers as a sum of two products each of the numbers and another properly selected integer:*

  $a \cdot x + b \cdot y = gcd(x,y)$

Determining inverse numbers in Z_N allows solving linear equations in modular arithmetic:

the equation: $a \cdot x + b = 0$ (in Z_N)

has the solution: $x = -b \cdot a^{-1}$ (in Z_N)

Definition The symbol Z^*_N denotes a set of all elements of Z_N that are invertible in Z_N; that means the set of numbers x that belong to Z_N, and x and N are relatively prime.

- *for example for a prime number **p**:*

  $Z^*_p = Z_p \setminus \{0\} = \{1, 2, ..., p - 1\}$

Calculating inverse numbers in Z_N

Inverse numbers in Z_N can be determined in time $O(\log_2 N)$ using the Euclidean algorithm, which allows to compute the greatest common divisor of two integers.

Extended Euclidean algorithm finds the coefficients of Bézout's identity, that are integer numbers a and b such that:

$a \cdot x + b \cdot y = gcd(x,y)$

In order to receive the inverse number, one should perform the following transformations:

$a \cdot x + b \cdot y = 1$

$b \cdot y = 1$ (in Z_a)

Thus, b is the inverse number of y in Z_a.

The numbers a and b should be calculated during each step of the Euclidean algorithm, using the received values of those numbers in previous steps and the quotients:

$a_i = a_{i-2} - q_{i-1} \cdot a_{i-1}$

$b_i = b_{i-2} - q_{i-1} \cdot b_{i-1}$

If algorithm's steps are numbered from 1, one should assume the following initial values of the coefficients:

$a_{-1} = 1$

$a_0 = 0$

$b_{-1} = 0$

$b_0 = 1$

The final values of the coefficients a and b are received in the same step when the greatest common divisor of x and y is calculated (thus, in the step with the last non-zero remainder).

Euler's Totient Function and Euler's Theorem

The Euler's totient function, or phi (φ) function is a very important number theoretic function having a deep relationship to prime numbers and the so-called order of integers. The totient $\varphi(n)$ of a positive integer n greater than 1 is defined to be the number of positive integers less than n that are coprime to n. $\varphi(1)$ is defined to be 1. The following table shows the function values for the first several natural numbers:

n	**φ(*n*)**	**numbers coprime to n**
1	1	1
2	1	1
3	2	1, 2
4	2	1,3
5	4	1,2,3,4
6	2	1,5
7	6	1,2,3,4,5,6

n	φ(*n*)	numbers coprime to n
8	4	1,3,5,7
9	6	1,2,4,5,7,8
10	4	1,3,7,9
11	10	1,2,3,4,5,6,7,8,9,10
12	4	1,5,7,11
13	12	1,2,3,4,5,6,7,8,9,10,11,12
14	6	1,3,5,9,11,13
15	8	1,2,4,7,8,11,13,14

Can you find some relationships between *n* and φ(*n*)? One thing you may have noticed is that:

when *n* is a prime number (e.g. 2, 3, 5, 7, 11, 13), **φ(*n*) = *n*-1**.

But how about the composite numbers? You may also have noticed that, for example, 15 = 3*5 and φ(*15*) = φ(*3*)*φ(*5*) = 2*4 = 8. This is also true for 14,12,10 and 6. However, it does not hold for 4, 8, 9. For example, 9 = 3*3 , but φ(*9*) = 6 ≠ φ(3)*φ(*3*) = 2*2 =4. In fact, this multiplicative relationship is conditional:

when *m* and *n* are coprime, φ(*mn*) = φ(*m*)*φ(*n*)**.

The general formula to compute φ(n) is the following:

If the prime factorisation of n is given by n =$p_1^{e_1}$*...*$p_n^{e_n}$, then φ(n) = n *(1 - 1/p_1)* ... (1 - 1/p_n).

For example:

- $9 = 3^2$, $\varphi(9) = 9^* (1-1/3) = 6$
- $4 = 2^2$, $\varphi(4) = 4^* (1-1/2) = 2$
- $15 = 3^*5$, $\varphi(15) = 15^* (1-1/3)^*(1-1/5) = 15^*(2/3)^*(4/5) = 8$

Euler's theorem generalises Fermat's theorem to the case where the modulus is not prime. It says that:

if n is a positive integer and a, n are coprime, then $a^{\varphi(n)} \equiv 1 \mod n$ where $\varphi(n)$ is the Euler's totient function.

Let's see some examples:

- $165 = 15^*11$, $\varphi(165) = \varphi(15)^*\varphi(11) = 80$. $8^{80} \equiv 1 \mod 165$
- $1716 = 11^*12^*13$, $\varphi(1716) = \varphi(11)^*\varphi(12)^*\varphi(13) = 480$. $7^{480} \equiv 1 \mod 1716$
- $\varphi(13) = 12$, $9^{12} \equiv 1 \mod 13$

We can see that Fermat's little theorem is a special case of Euler's Theorem: for any prime n, $\varphi(n) = n-1$ and any number a $0< a <n$ is coprime to n. From Euler's Theorem, we can easily get several useful corollaries. First:

if n is a positive integer and a, n are coprime, then $a^{\varphi(n)+1} \equiv a \mod n$.

This is because $a^{\varphi(n)+1} = a^{\varphi(n)}*a$, $a^{\varphi(n)} \equiv 1 \mod n$ and $a \equiv a \mod n$, so $a^{\varphi(n)+1} \equiv a \mod n$. From here, we can go even further:

if n is a positive integer and a, n are coprime, $b \equiv 1 \mod \varphi(n)$, then $a^b \equiv a \mod n$.

If $b \equiv 1 \mod \varphi(n)$, then it can be written as $b = k^*\varphi(n)+1$ for some k. Then $a^b = a^{k^*\varphi(n)+1} = (a^{\varphi(n)})^k*a$. Since $a^{\varphi(n)} \equiv 1 \mod n$, $(a^{\varphi(n)})^k \equiv 1^k \equiv 1 \mod n$. Then $(a^{\varphi(n)})^k*a \equiv a \mod n$. This is why RSA works.

CHINESE REMAINDER THEOREM(CRT)

CRT says that in modulo M arithmetic, if M can be expressed as a product of n integers that are pairwise coprime, then every integer in the set $Z_M = \{0, 1, 2, ..., M - 1\}$ can be reconstructed from residues with respect to those n numbers.

•For example, the prime factors of 10 are 2 and 5. Now let's consider an integer 9 in Z_{10}. Its residue modulo 2 is 1 and the residue modulo 5 is 4. So, according to CRT, 9 can be represented by the tuple (1, 4).

$$M = \prod_{i=1}^{k} m_i$$

where the m_i are pairwise relatively prime; that is, $\gcd(m_i, m_j) = 1$ for $1 \le i, j \le k$, and $i \ne j$. We can represent any integer A in Z_M by a k-tuple whose elements are in Z_{m_i} using the following correspondence:

$$A \leftrightarrow (a_1, a_2, \ldots, a_k)$$

where $A \in Z_M$, $a_i \in Z_{m_i}$, and $a_i = A \bmod m_i$ for $1 \le i \le k$. The CRT makes two assertions.

1. The mapping of Equation is a one-to-one correspondence (called a **bijection**) between Z_M and the Cartesian product $Z_{m_1} \times Z_{m_2} \times \ldots \times Z_{m_k}$. That is, for every integer A such that $0 \le A \le M$, there is a unique k-tuple $(a_1, a_2, \ldots, a_k)$ with $0 \le a_i < m_i$ that represents it, and for every such k-tuple $(a_1, a_2, \ldots, a_k)$, there is a unique integer A in Z_M.
2. Operations performed on the elements of Z_M can be equivalently performed on the corresponding k-tuples by performing the operation independently in each coordinate position in the appropriate system.

-

Let us demonstrate the **first assertion**. The transformation from A to $(a_1, a_2, \ldots, a_k)$, is obviously unique; that is, each a_i is uniquely calculated as $a_i = A \bmod m_i$. Computing A from $(a_1, a_2, \ldots, a_k)$ can be done as follows. Let $M_i = M/m_i$ for $1 \le i \le k$. Note that $M_i = m_1 \times m_2 \times \ldots \times m_{i-1} \times m_{i+1} \times \ldots \times m_k$, so that $M_i \equiv 0 \pmod{m_j}$ for all $j \ne i$. Then let

$$c_i = M_i \times \left(M_i^{-1} \bmod m_i\right) \qquad \text{for } 1 \le i \le k \tag{8.8}$$

By the definition of M_i, it is relatively prime to m_i and therefore has a unique multiplicative inverse mod m_i. So Equation (8.8) is well defined and produces a unique value c_i. We can now compute

$$A \equiv \left(\sum_{i-1}^{k} a_i c_i\right) \pmod{M} \tag{8.9}$$

To show that the value of A produced by Equation (8.9) is correct, we must show that $a_i = A \bmod m_i$ for $1 \le i \le k$. Note that $c_j \equiv M_j \equiv 0 \pmod{m_i}$ if $j \ne i$, and that $c_i \equiv 1 \pmod{m_i}$. It follows that $a_i = A \bmod m_i$.

The **second assertion** of the CRT, concerning arithmetic operations, follows from the rules for modular arithmetic. That is, the second assertion can be stated as follows: If

$$A \leftrightarrow (a_1, a_2, \ldots, a_k)$$
$$B \leftrightarrow (b_1, b_2, \ldots, b_k)$$

then

$$(A + B) \bmod M \leftrightarrow ((a_1 + b_1) \bmod m_1, \ldots, (a_k + b_k) \bmod m_k)$$
$$(A - B) \bmod M \leftrightarrow ((a_1 - b_1) \bmod m_1, \ldots, (a_k - b_k) \bmod m_k)$$
$$(A \times B) \bmod M \leftrightarrow ((a_1 \times b_1) \bmod m_1, \ldots, (a_k \times b_k) \bmod m_k)$$

One of the useful features of the Chinese remainder theorem is that it provides a way to manipulate (potentially very large) numbers mod M in terms of tuples of smaller numbers. This can be useful when M is 150 digits or more. However, note that it is necessary to know beforehand the factorization of M.

DISCRETE LOGARITHMS

Discrete logarithms are fundamental to a number of public-key algorithms, includ- ingDiffie-Hellman key exchange and the digital signature algorithm (DSA). This section provides a brief overview of discrete logarithms. For the interested reader, more detailed developments of this topic can be found in [ORE67] and [LEVE90].

The Powers of an Integer, Modulo n

Recall from Euler's theorem [Equation (8.4)] that, for every a and n that are rela- tively prime,

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

where $\phi(n)$, Euler's totient function, is the number of positive integers less than n and relatively prime to n. Now consider the more general expression:

$$a^m \equiv 1 \pmod{n} \qquad \textbf{(8.10)}$$

If a and n are relatively prime, then there is at least one integer m that satisfies Equation (8.10), namely, $M = \phi(n)$. The least positive exponent m for which Equation (8.10) holds is referred to in several ways:

- The order of a (mod n)
- The exponent to which a belongs (mod n)
- The length of the period generated by a

To see this last point, consider the powers of 7, modulo 19:

$$7^1 \equiv 7 \pmod{19}$$
$$7^2 = 49 = 2 \times 19 + 11 \equiv 11 \pmod{19}$$
$$7^3 = 343 = 18 \times 19 + 1 \equiv 1 \pmod{19}$$
$$7^4 = 2401 = 126 \times 19 + 7 \equiv 7 \pmod{19}$$
$$7^5 = 16807 = 884 \times 19 + 11 \equiv 11 \pmod{19}$$

There is no point in continuing because the sequence is repeating. This can be proven by noting that $7^3 \equiv 1 \pmod{19}$, and therefore, $7^{3+j} \equiv 7^3 7^j \equiv 7^j \pmod{19}$, and hence, any two powers of 7 whose exponents differ by 3 (or a multiple of 3) are congruent to each other (mod 19). In other words, the sequence is periodic, and the length of the period is the smallest positive exponent m such that $7^m \equiv 1 \pmod{19}$.

Table 8.3 shows all the powers of a, modulo 19 for all positive a 6 19. The length of the sequence for each base value is indicated by shading. Note the following:

1. All sequences end in 1. This is consistent with the reasoning of the preceding few paragraphs.

2. The length of a sequence divides $\phi(19) = 18$. That is, an integral number of sequences occur in each row of the table.

3. Some of the sequences are of length 18. In this case, it is said that the base integer a generates (via powers) the set of nonzero integers modulo 19. Each such integer is called a primitive root of the modulus 19.

Table 8.3 Powers of Integers, Modulo 19

a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15}	a^{16}	a^{17}	a^{18}
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	8	16	13	7	14	9	18	17	15	11	3	6	12	5	10	1
3	9	8	5	15	7	2	6	18	16	10	11	14	4	12	17	13	1
4	16	7	9	17	11	6	5	1	4	16	7	9	17	11	6	5	1
5	6	11	17	9	7	16	4	1	5	6	11	17	9	7	16	4	1
6	17	7	4	5	11	9	16	1	6	17	7	4	5	11	9	16	1
7	11	1	7	11	1	7	11	1	7	11	1	7	11	1	7	11	1
8	7	18	11	12	1	8	7	18	11	12	1	8	7	18	11	12	1
9	5	7	6	16	11	4	17	1	9	5	7	6	16	11	4	17	1
10	5	12	6	3	11	15	17	18	9	14	7	13	16	8	4	2	1
11	7	1	11	7	1	11	7	1	11	7	1	11	7	1	11	7	1
12	11	18	7	8	1	12	11	18	7	8	1	12	11	18	7	8	1
13	17	12	4	14	11	10	16	18	6	2	7	15	5	8	9	3	1
14	6	8	17	10	7	3	4	18	5	13	11	2	9	12	16	15	1
15	16	12	9	2	11	13	5	18	4	3	7	10	17	8	6	14	1
16	9	11	5	4	7	17	6	1	16	9	11	5	4	7	17	6	1
17	4	11	16	6	7	5	9	1	17	4	11	16	6	7	5	9	1
18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1	18	1

More generally, we can say that the highest possible exponent to which a number can belong (mod n) is $\phi(n)$. If a number is of this order, it is referred to as a **primitive root** of n. The importance of this notion is that if a is a primitive root of n, then its powers

$$a, a^2, \ldots, a^{\phi(n)}$$

are distinct (mod n) and are all relatively prime to n. In particular, for a prime number p, if a is a primitive root of p, then

$$a, a^2, \ldots, a^{p-1}$$

are distinct (mod p). For the prime number 19, its primitive roots are 2, 3, 10, 13, 14, and 15.

Not all integers have primitive roots. In fact, the only integers with primitive roots are those of the form 2, 4, pa, and $2pa$, where p is any odd prime and a is a positive integer.

Logarithms for Modular Arithmetic

With ordinary positive real numbers, the logarithm function is the inverse of expo- nentiation. An analogous function exists for modular arithmetic.

Let us briefly review the properties of ordinary logarithms. The logarithm of a number is defined to be the power to which some positive base (except 1) must be raised in order to equal the number. That is, for base x and for a value y,

$$y = x^{\log_x(y)}$$

The properties of logarithms include

$$\log_x(1) = 0$$

$$\log_x(x) = 1$$

$$\log_x(yz) = log_x(y) + \log_x(z) \quad \textbf{(8.11)}$$

$$\log_x(y^r) = r \times \log_x(y) \quad \textbf{(8.12)}$$

Consider a primitive root a for some prime number p (the argument can be developed for nonprimes as well). Then we know that the powers of a from 1 through $(p - 1)$ produce each integer from 1 through $(p - 1)$ exactly once. We also know that any integer b satisfies

$$b \equiv r \pmod{p} \quad \text{for some } r, \text{ where } 0 \leq r \leq (p - 1)$$

by the definition of modular arithmetic. It follows that for any integer b and a prim- itive root a of prime number p, we can find a unique exponent i such that

$$b \equiv a^i \pmod{p} \quad \text{where } 0 \leq i \leq (p - 1)$$

This exponent i is referred to as the **discrete logarithm** of the number b for the base a (mod p). We denote this value as $\text{dlog}_{a,p}(b)$.10

The Euclidean Algorithm

Recall that the Greatest Common Divisor (GCD) of two integers A and B is the **largest integer that divides both A and B**.

The **Euclidean Algorithm** is a technique for quickly finding the **GCD** of two integers.

The Algorithm

The Euclidean Algorithm for finding GCD(A,B) is as follows:

- If A = 0 then GCD(A,B)=B, since the GCD(0,B)=B, and we can stop.
- If B = 0 then GCD(A,B)=A, since the GCD(A,0)=A, and we can stop.
- Write A in quotient remainder form (A = B·Q + R)
- Find GCD(B,R) using the Euclidean Algorithm since GCD(A,B) = GCD(B,R)

Example:

Find the GCD of 270 and 192

- A=270, B=192
- A ≠0
- B ≠0
- Use long division to find that 270/192 = 1 with a remainder of 78. We can write this as: 270 = 192 * 1 +78
- Find GCD(192,78), since GCD(270,192)=GCD(192,78)
  A=192, B=78
- A ≠0
- B ≠0
- Use long division to find that 192/78 = 2 with a remainder of 36. We can write this as:
- 192 = 78 * 2 + 36
- Find GCD(78,36), since GCD(192,78)=GCD(78,36)
  A=78, B=36
- A ≠0

- B ≠0
- Use long division to find that 78/36 = 2 with a remainder of 6. We can write this as:
- 78 = 36 * 2 + 6
- Find GCD(36,6), since GCD(78,36)=GCD(36,6)

  A=36, B=6

- A ≠0
- B ≠0
- Use long division to find that 36/6 = 6 with a remainder of 0. We can write this as:
- 36 = 6 * 6 + 0
- Find GCD(6,0), since GCD(36,6)=GCD(6,0)

  A=6, B=0

- A ≠0
- B =0, GCD(6,0)=6

  So we have shown:

  GCD(270,192) = GCD(192,78) = GCD(78,36) = GCD(36,6) = GCD(6,0) = 6

  GCD(270,192) = 6

  Understanding the Euclidean Algorithm

  If we examine the Euclidean Algorithm we can see that it makes use of the following properties:

- GCD(A,0) = A
- GCD(0,B) = B
- **If A = B·Q + R and B≠0 then GCD(A,B) = GCD(B,R)** where Q is an integer, R is an integer between 0 and B-1

  The first two properties let us find the GCD if either number is 0. The third property lets us take a larger, more difficult to solve problem, and **reduce it to a smaller, easier to solve problem**.

The Euclidean Algorithm makes use of these properties by rapidly reducing the problem into easier and easier problems, using the third property, until it is easily solved by using one of the first two properties.

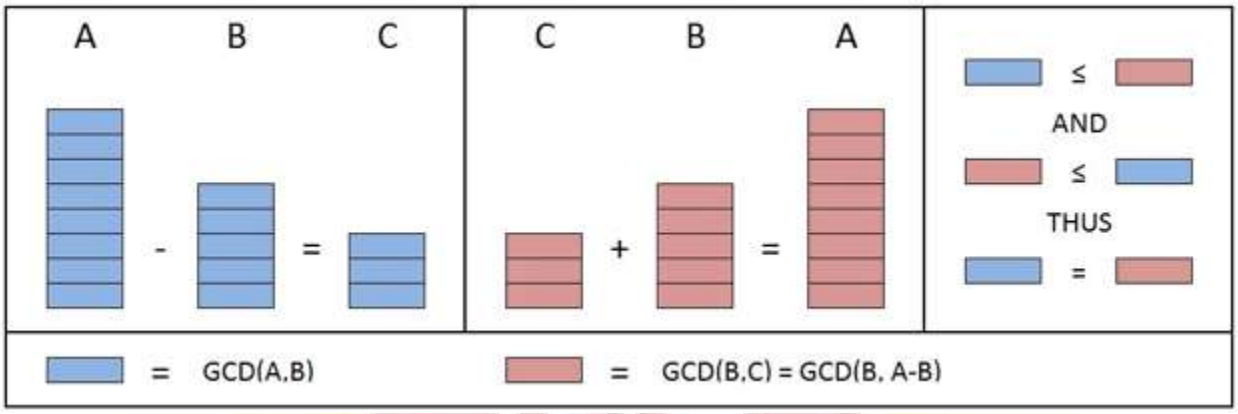
We can understand why these properties work by proving them.

We can prove that GCD(A,0)=A is as follows:

- The largest integer that can evenly divide A is A.
- All integers evenly divide 0, since for any integer, C, we can write $C \cdot 0 = 0$. So we can conclude that A must evenly divide 0.
- The greatest number that divides both A and 0 is A.

The proof for **GCD(0,B)=B** is similar. (Same proof, but we replace A with B).

To prove that **GCD(A,B)=GCD(B,R)** we first need to show that **GCD(A,B)=GCD(B,A-B)**.



Suppose we have three integers **A,B** and **C** such that **A-B=C**.

Proof that the GCD(A,B) evenly divides C

The GCD(A,B), by definition, evenly divides A. As a result, A must be some multiple of GCD(A,B). i.e. $X \cdot GCD(A,B)=A$ where X is some integer

The GCD(A,B), by definition, evenly divides B. As a result, B must be some multiple of GCD(A,B). i.e. $Y \cdot GCD(A,B)=B$ where Y is some integer

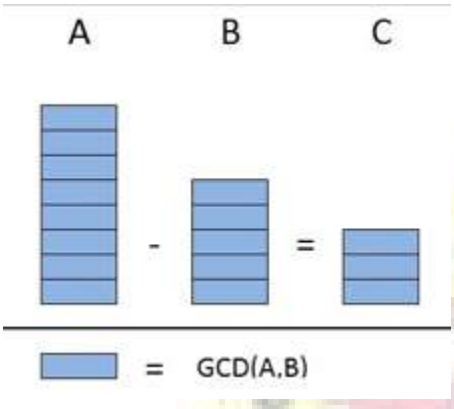
A-B=C gives us:

- $X \cdot GCD(A,B) - Y \cdot GCD(A,B) = C$

- (X - Y)·GCD(A,B) = C

  So we can see that GCD(A,B) evenly divides C.

  An illustration of this proof is shown in the left portion of the figure below:



Proof that the GCD(B,C) evenly divides A

The GCD(B,C), by definition, evenly divides B. As a result, B must be some multiple of GCD(B,C). i.e. M·GCD(B,C)=B where M is some integer

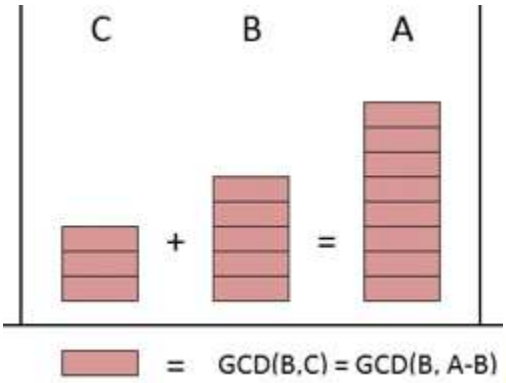
The GCD(B,C), by definition, evenly divides C. As a result, C must be some multiple of GCD(B,C). i.e. N·GCD(B,C)=C where N is some integer

A-B=C gives us:

- B+C=A
- M·GCD(B,C) + N·GCD(B,C) = A
- (M + N)·GCD(B,C) = A

  So we can see that GCD(B,C) evenly divides A.

  An illustration of this proof is shown in the figure below



Proof that GCD(A,B)=GCD(A,A-B)

- GCD(A,B) by definition, evenly divides B.
- We proved that GCD(A,B) evenly divides C.
- Since the GCD(A,B) divides both B and C evenly it is a common divisor of B and C.

  GCD(A,B) must be less than or equal to, GCD(B,C), because GCD(B,C) is the "greatest" common divisor of B and C.

- GCD(B,C) by definition, evenly divides B.
- We proved that GCD(B,C) evenly divides A.
- Since the GCD(B,C) divides both A and B evenly it is a common divisor of A and B.

  GCD(B,C) must be less than or equal to, GCD(A,B), because GCD(A,B) is the **"greatest"** common divisor of A and B.

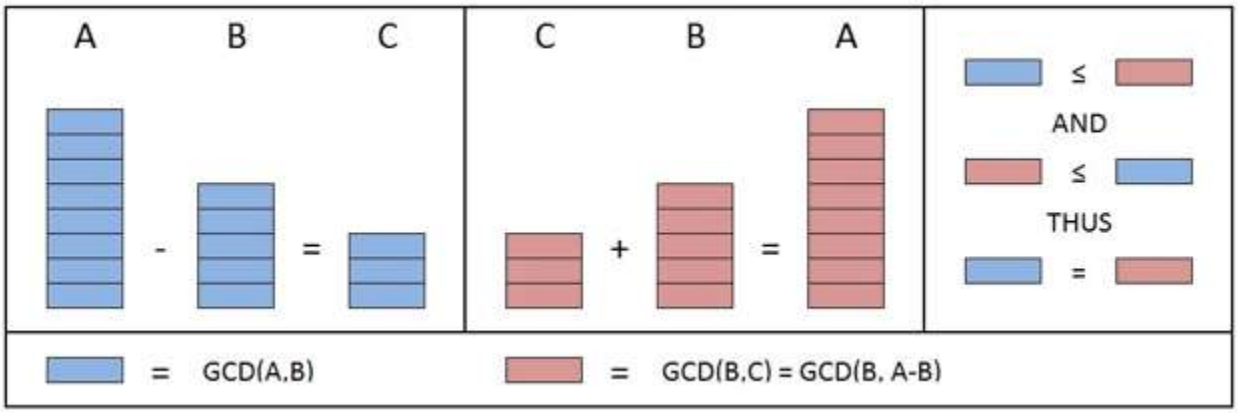
Given that GCD(A,B)≤GCD(B,C) and GCD(B,C)≤GCD(A,B) we can conclude that:

GCD(A,B)=GCD(B,C)

Which is equivalent to:

GCD(A,B)=GCD(B,A-B)

An illustration of this proof is shown in the right portion of the figure below.



Proof that GCD(A,B) = GCD(B,R)

We proved that GCD(A,B)=GCD(B,A-B)

The order of the terms does not matter so we can say GCD(A,B)=GCD(A-B,B)

We can repeatedly apply GCD(A,B)=GCD(A-B,B) to obtain:

GCD(A,B)=GCD(A-B,B)=GCD(A-2B,B)=GCD(A-3B,B)=...=GCD(A-Q·B,B)

But A= B·Q + R so A-Q·B=R

Thus **GCD(A,B)=GCD(R,B)**

The order of terms does not matter, thus:

GCD(A,B)=GCD(B,R)

The Chinese Remainder Theorem

Suppose we wish to solve

x=2(mod5)

x=3(mod7)

for x. If we have a solution y, then y+35 is also a solution. So we only need to look for solutions modulo 35. By brute force, we find the only solution is x=17(mod35).

For any system of equations like this, the Chinese Remainder Theorem tells us there is always a unique solution up to a certain modulus, and describes how to find the solution efficiently.

Theorem: Let p,q be coprime. Then the system of equations

$x=a(\mathrm{mod}p)$

$x=b(\mathrm{mod}q)$

has a unique solution for x modulo pq.

The reverse direction is trivial: given $x\in Zpq$, we can reduce x modulo p and x modulo q to obtain two equations of the above form.

Proof: Let $p1=p^{-1}(\mathrm{mod}q)$ and $q1=q^{-1}(\mathrm{mod}p)$. These must exist since p,q are coprime. Then we claim that if y is an integer such that

$y=aqq1+bpp1(\mathrm{mod}pq)$

then y satisfies both equations:

Modulo p, we have $y=aqq1=a(\mathrm{mod}p)$ since $qq1=1(\mathrm{mod}p)$. Similarly $y=b(\mathrm{mod}q)$. Thus y is a solution for x.

It remains to show no other solutions exist modulo pq. If $z=a(\mathrm{mod}p)$ then $z-y$ is a multiple of p. If $z=b(\mathrm{mod}q)$ as well, then $z-y$ is also a multiple of q. Since p and q are coprime, this implies $z-y$ is a multiple of pq, hence $z=y(\mathrm{mod}pq)$. ■

This theorem implies we can represent an element of Zpq by one element of Zp and one element of Zq, and vice versa. In other words, we have a bijection between Zpq and $Zp\times Zq$.

Examples: We can write $17\in Z35$ as $(2,3)\in Z5\times Z7$. We can write $1\in Zpq$ as $(1,1)\in Zp\times Zq$.

In fact, this correspondence goes further than a simple relabelling. Suppose $x,y\in Zpq$ correspond to $(a,b),(c,d)\in Zp\times Zq$ respectively. Then a little thought shows $x+y$ corresponds to $(a+c,b+d)$, and similarly xy corresponds to (ac,bd).

A practical application: if we have many computations to perform on $x\in Zpq$ (e.g. RSA signing and decryption), we can convert x to $(a,b)\in Zp\times Zq$ and do all the computations on a and b instead

before converting back. This is often cheaper because for many algorithms, doubling the size of the input more than doubles the running time.

Example: To compute $17\times17(\mathrm{mod}35)$, we can compute $(2\times2,3\times3)=(4,2)$ in $Z5\times Z7$, and then apply the Chinese Remainder Theorem to find that (4,2) is $9(\mathrm{mod}35)$.

Let us restate the Chinese Remainder Theorem in the form it is usually presented.

For Several Equations

Theorem: Let $m1,...,mn$ be pairwise coprime (that is $\gcd(mi,mj)=1$ whenever $i\neq j$). Then the system of n equations

$x=a1(\mathrm{mod}m1)$

...

$x=an(\mathrm{mod}mn)$

has a unique solution for x modulo M where $M=m1...mn$.

Proof: This is an easy induction from the previous form of the theorem, or we can write down the solution directly.

Define $bi=M/mi$ (the product of all the moduli except for mi) and $bi'=bi-1(\mathrm{mod}mi)$. Then by a similar argument to before,

$x=\sum i=1naibibi'(\mathrm{mod}M)$

is the unique solution.■

Prime Powers First

An important consequence of the theorem is that when studying modular arithmetic in general, we can first study modular arithmetic a prime power and then appeal to the Chinese Remainder Theorem to generalize any results. For any integer n, we factorize n into primes $n=p1k1...pmkm$ and then use the Chinese Remainder Theorem to get

$Zn=Zp1k1\times...\times Zpmkm$

To prove statements in Zpk, one starts from Zp, and inductively works up to Zpk. Thus the most important case to study is

UNIT III

PUBLIC KEY CRYPTOGRAPHY

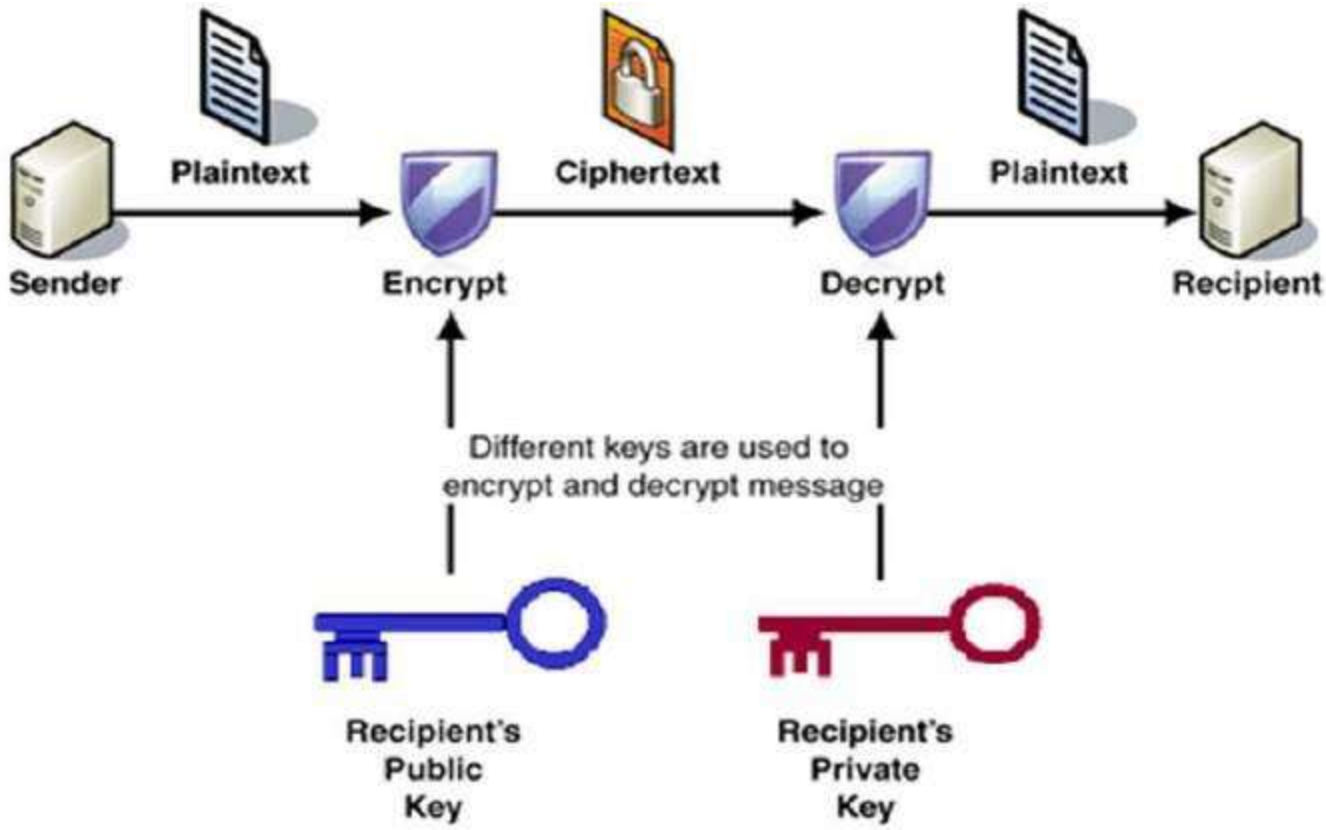
Public Key Cryptography

Unlike symmetric key cryptography, we do not find historical use of public-key cryptography. It is a relatively new concept.

Symmetric cryptography was well suited for organizations such as governments, military, and big financial corporations were involved in the classified communication.

With the spread of more unsecure computer networks in last few decades, a genuine need was felt to use cryptography at larger scale. The symmetric key was found to be non-practical due to challenges it faced for key management. This gave rise to the public key cryptosystems.

The process of encryption and decryption is depicted in the following illustration –



The most important properties of public key encryption scheme are −

- Different keys are used for encryption and decryption. This is a property which set this scheme different than symmetric encryption scheme.
- Each receiver possesses a unique decryption key, generally referred to as his private key.
- Receiver needs to publish an encryption key, referred to as his public key.
- Some assurance of the authenticity of a public key is needed in this scheme to avoid spoofing by adversary as the receiver. Generally, this type of cryptosystem involves trusted third party which certifies that a particular public key belongs to a specific person or entity only.
- Encryption algorithm is complex enough to prohibit attacker from deducing the plaintext from the ciphertext and the encryption (public) key.

- Though private and public keys are related mathematically, it is not be feasible to calculate the private key from the public key. In fact, intelligent part of any public-key cryptosystem is in designing a relationship between two keys.

There are three types of Public Key Encryption schemes. We discuss them in following sections –

RSA Cryptosystem

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem.

We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

- **Generate the RSA modulus (n)**
    - Select two large primes, p and q.
    - Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.
- **Find Derived Number (e)**
    - Number **e** must be greater than 1 and less than (p − 1)(q − 1).
    - There must be no common factor for e and (p − 1)(q − 1) except for 1. In other words two numbers e and (p – 1)(q – 1) are coprime.
- **Form the public key**
    - The pair of numbers (n, e) form the RSA public key and is made public.

- Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.

- **Generate the private key**
  - Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.
  - Number d is the inverse of e modulo (p - 1)(q − 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).
  - This relationship is written mathematically as follows −

$$ed = 1 \bmod (p - 1)(q - 1)$$

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output.

Example

An example of generating RSA Key pair is given below. (For ease of understanding, the primes p & q taken here are small values. Practically, these values are very high).

- Let two primes be p = 7 and q = 13. Thus, modulus n = pq = 7 x 13 = 91.
- Select e = 5, which is a valid choice since there is no number that is common factor of 5 and (p − 1)(q − 1) = 6 × 12 = 72, except for 1.
- The pair of numbers (n, e) = (91, 5) forms the public key and can be made available to anyone whom we wish to be able to send us encrypted messages.
- Input p = 7, q = 13, and e = 5 to the Extended Euclidean Algorithm. The output will be d = 29.
- Check that the d calculated is correct by computing −

$$de = 29 \times 5 = 145 = 1 \bmod 72$$

- Hence, public key is (91, 5) and private keys is (91, 29).

Encryption and Decryption

Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.

Interestingly, RSA does not directly operate on strings of bits as in case of symmetric key encryption. It operates on numbers modulo n. Hence, it is necessary to represent the plaintext as a series of numbers less than n.

RSA Encryption

- Suppose the sender wish to send some text message to someone whose public key is (n, e).
- The sender then represents the plaintext as a series of numbers less than n.
- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as −

$C = P^e \bmod n$

- In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.
- Returning to our Key Generation example with plaintext P = 10, we get ciphertext C −

$C = 10^5 \bmod 91$

RSA Decryption

- The decryption process for RSA is also very straightforward. Suppose that the receiver of public-key pair (n, e) has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

$\text{Plaintext} = C^d \bmod n$

- Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 −

$\text{Plaintext} = 82^{29} \bmod 91 = 10$

RSA Analysis

The security of RSA depends on the strengths of two separate functions. The RSA cryptosystem is most popular public-key cryptosystem strength of which is based on the practical difficulty of factoring the very large numbers.

- **Encryption Function** − It is considered as a one-way function of converting plaintext into ciphertext and it can be reversed only with the knowledge of private key d.
- **Key Generation** − The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one way function, going from p & q values to modulus n is easy but reverse is not possible.

If either of these two functions are proved non one-way, then RSA will be broken. In fact, if a technique for factoring efficiently is developed then RSA will no longer be safe.

The strength of RSA encryption drastically goes down against attacks if the number p and q are not large primes and/ or chosen public key e is a small number.

RSA algorithm is a public key encryption technique and is considered as the most secure way of encryption. It was invented by Rivest, Shamir and Adleman in year 1978 and hence name **RSA** algorithm.

Algorithm

The RSA algorithm holds the following features −

- RSA algorithm is a popular exponentiation in a finite field over integers including prime numbers.
- The integers used by this method are sufficiently large making it difficult to solve.
- There are two sets of keys in this algorithm: private key and public key.

You will have to go through the following steps to work on RSA algorithm −

Step 1: Generate the RSA modulus

The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as shown −

N=p*q

Here, let N be the specified large number.

Step 2: Derived Number (e)

Consider number e as a derived number which should be greater than 1 and less than (p-1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (q-1) except 1

Step 3: Public key

The specified pair of numbers **n** and **e** forms the RSA public key and it is made public.

Step 4: Private Key

Private Key **d** is calculated from the numbers p, q and e. The mathematical relationship between the numbers is as follows −

ed = 1 mod (p-1) (q-1)

The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters.

Encryption Formula

Consider a sender who sends the plain text message to someone whose public key is **(n,e).** To encrypt the plain text message in the given scenario, use the following syntax −

C = Pe mod n

Decryption Formula

The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Considering receiver **C** has the private key **d**, the result modulus will be calculated as −

Plaintext = Cd mod n

Previous Page

Generating RSA keys

The following steps are involved in generating RSA keys −

- Create two large prime numbers namely **p** and **q**. The product of these numbers will be called **n**, where **n= p*q**
- Generate a random number which is relatively prime with **(p-1)** and **(q-1).** Let the number be called as **e**.
- Calculate the modular inverse of e. The calculated inverse will be called as **d**

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Let us learn the mechanism behind RSA algorithm :

>> Generating Public Key :

- Select two prime no's. Suppose **P = 53 and Q = 59**.
- Now First part of the Public key :**n = P*Q = 3127**.
-
- We also need a small exponent say **e** :
- But e Must be
  -
    - An integer.
    -
    - Not be a factor of n.
    -
    - **1 < e <**Φ(n) [Φ(n) is discussed below],
    - Let us now consider it to be equal to 3.
-

- Our Public Key is made of n and e

>> Generating Private Key :

- We need to calculate Φ(n) :
- Such that **Φ(n) = (P-1)(Q-1)**
- so, Φ(n) = 3016
-
- Now calculate Private Key, **d** :
- **d = (k*Φ(n) + 1) / e** for some integer k
- For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt **"HI"** :

- Convert letters to numbers : H = 8 and I = 9
-

- Thus **Encrypted Data c = 89^e mod n**.
- Thus our Encrypted Data comes out to be 1394
-
- Now we will decrypt **1394** :
-
- **Decrypted Data = c^d mod n**.
- Thus our Encrypted Data comes out to be 89
- **8 = H and I = 9 i.e. "HI".**

KEY MANAGEMENT

Key management refers to managing cryptographic keys within a cryptosystem. It deals with generating, exchanging, storing, using and replacing keys as needed at the user level. A key management system will also include key servers, user procedures and protocols, including cryptographic protocol design. The security of the cryptosystem is dependent upon successful key management.

This article introduces into key management from a perspective of a CISO or any person in charge of maintaining information security within an organization.

What is a CKMS Policy?

The term CKMS stands for Cryptographic Key Management System. A CKMS Security Policy provides the rules that are to be used to protect keys and metadata that the CKMS supports. This Policy establishes and specifies rules for this information that will protect its:

- Confidentiality
- Integrity
- Availability
- Authentication of source

This protection covers the complete key life-cycle from the time the key becomes operational to its elimination.

A CKMS Policy may also include selecting all the cryptographic mechanisms and protocols that may be utilized by the Key Management System. The Policy must remain consistent with the organization's higher-level policies. For instance, if an organization's Information Security Policy requires that electronically transmitted information is to receive protection to maintain its confidentiality for 30 years, both the CKMS design and CKMS Security Policy must be able to support that policy.

When designing a Key Management System, a system designer may be not necessarily be a member of the organization that will be using the system. Therefore, he may not have access to the policies of the organization. Often the designer will create a set of policies and features that are commonplace for the organization's market. The designer will normally then provide documentation to explain how these policies and features are used within the CKMS Security Policy. The organization may choose to work with the design to modify the Policy to better fit their needs. Overall, it is the responsibility of the organization to ensure that the Key Management System design is capable of supporting their CKMS Security Policy.

Often, organizations will use a hierarchy of policies to address their policy requirements. Depending upon the organization's needs, their hierarchy may consist of multiple levels. A hierarchy may include:

- Top level – Information Management, which specifies the goals for information security and the requirements and expected control actions for lower levels, including:
  - Industry standards
  - Legal requirements
  - Organizational goals
- Second level – Information Security, which provides more information on the actual procedures that will be implemented and enforced to provide the security as specified by the top level. This level includes:
  - List of potential threats to keeping the organization's information secure
  - Associated risks

- Guidelines of Data Security Policy
- Outputs to the KMS Security Policy

- Third level – KMS Security Policy, which establishes and provides specifics on protecting keys and metadata. This policy includes:
- Protections used for each key type and its associated metadata
- Retention period for keys and metadata according to the sensitivity of the data being protected
- Domain Security Policy

Key Management Compliance

Key management compliance refers to the oversight, assurance and capability of being able to demonstrate that keys are securely managed. This includes the following individual compliance domains:

- *Physical security* – the most visible form of compliance, which may include locked doors to secure system equipment and surveillance cameras. These safeguards can prevent unauthorized access to printed copies of key material and computer systems that run key management software.
- *Logical security* – protects the organization against the theft or unauthorized access of information. This is where the use of cryptographic keys comes in by encrypting data, which is then rendered useless to those who do not have the key to decrypt it.
- *Personnel security* – this involves assigning specific roles or privileges to personnel to access information on a strict need-to-know basis. Background checks should be performed on new employees along with periodic role changes to ensure security.

Facing Problems and Challenges of Key Management

Managing cryptographic keys can be a challenge, especially for larger organizations that rely upon cryptography for various applications. The primary problems that are associated with managing cryptographic keys include:

- Using the correct procedure to update system certificates and keys
- Updating certificate and keys before they expire
- Dealing with proprietary issues when keeping track of crypto updates with legacy systems
- Locating remote devices that need to be updated

- Lacking overview as to the purpose, location and why various systems are used

Ten Security Tips for a Key Management System

1. Document the Security Policy so that it is easily understood.
2. Maintain malware protection
3. Patch vulnerabilities and turn off non-essential services on servers and devices
4. Perform third-party penetration testing
5. Make the system easy to use
6. Set up remote monitoring
7. Define appropriate crypto-periods for keys
8. Assign key management system roles and responsibilities
9. Meet the goals of the organization's information security policies
10. Define and classify cryptographic zones

DIFFIE-HELLMAN KEY EXCHANGE (EXPONENTIAL KEY EXCHANGE)

Diffie-Hellman key exchange, also called exponential key exchange, is a method of digital encryption that uses numbers raised to specific powers to produce decryption keys on the basis of components that are never directly transmitted, making the task of a would-be code breaker mathematically overwhelming.

To implement Diffie-Hellman, the two end users Alice and Bob, while communicating over a channel they know to be private, mutually agree on positive whole numbers p and q, such that p is a prime number and q is a generator of p. The generator q is a number that, when raised to positive whole-number powers less than p, never produces the same result for any two such whole numbers. The value of p may be large but the value of q is usually small.

Once Alice and Bob have agreed on p and q in private, they choose positive whole-number personal keys a and b, both less than the prime-number modulus p. Neither user divulges their personal key to anyone; ideally they memorize these numbers and do not write them down or store them anywhere. Next, Alice and Bob compute public keys a^* and b^* based on their personal keys according to the formulas

$$a* = q^a \bmod p$$

and

$$b* = q^b \bmod p$$

The two users can share their public keys $a*$ and $b*$ over a communications medium assumed to be insecure, such as the Internet or a corporate wide area network (WAN). From these public keys, a number x can be generated by either user on the basis of their own personal keys. Alice computes x using the formula

$$x = (b*)^a \bmod p$$

Bob computes x using the formula

$$x = (a*)^b \bmod p$$

The value of x turns out to be the same according to either of the above two formulas. However, the personal keys a and b, which are critical in the calculation of x, have not been transmitted over a public medium. Because it is a large and apparently random number, a potential hacker has almost no chance of correctly guessing x, even with the help of a powerful computer to conduct millions of trials. The two users can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the decryption key x.

The most serious limitation of Diffie-Hellman in its basic or "pure" form is the lack of authentication. Communications using Diffie-Hellman all by itself are vulnerable to man in the middle attacks. Ideally,Diffie-Hellman should be used in conjunction with a recognized authentication method such as digital signatures to verify the identities of the users over the public communications medium. Diffie-Hellman is well suited for use in data communication but is less often used for data stored or archived over long periods of time.

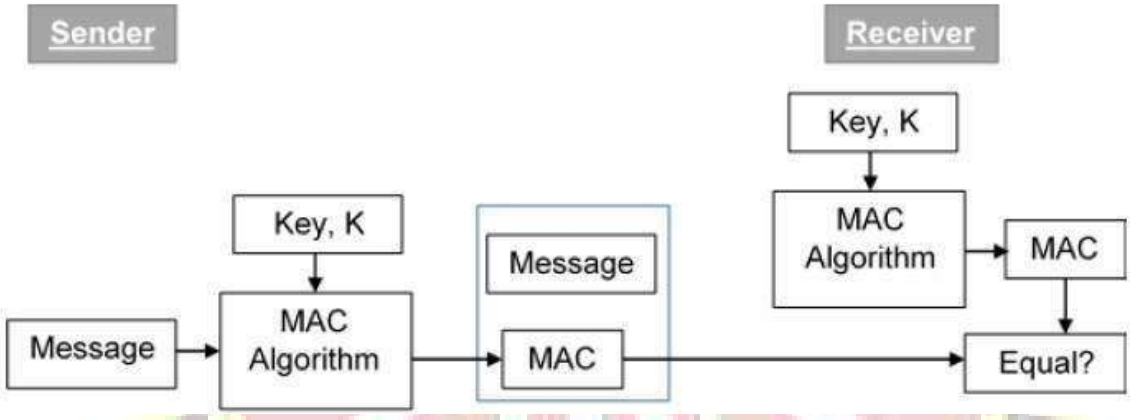
UNIT IV

MESSAGE AUTHENTICATION CODE (MAC)

MAC algorithm is a symmetric key cryptographic technique to provide message authentication. For establishing MAC process, the sender and receiver share a symmetric key K.

Essentially, a MAC is an encrypted checksum generated on the underlying message that is sent along with a message to ensure message authentication.

The process of using MAC for authentication is depicted in the following illustration −



Let us now try to understand the entire process in detail −

- The sender uses some publicly known MAC algorithm, inputs the message and the secret key K and produces a MAC value.
- Similar to hash, MAC function also compresses an arbitrary long input into a fixed length output. The major difference between hash and MAC is that MAC uses secret key during the compression.
- The sender forwards the message along with the MAC. Here, we assume that the message is sent in the clear, as we are concerned of providing message origin authentication, not confidentiality. If confidentiality is required then the message needs encryption.
- On receipt of the message and the MAC, the receiver feeds the received message and the shared secret key K into the MAC algorithm and re-computes the MAC value.

- The receiver now checks equality of freshly computed MAC with the MAC received from the sender. If they match, then the receiver accepts the message and assures himself that the message has been sent by the intended sender.
- If the computed MAC does not match the MAC sent by the sender, the receiver cannot determine whether it is the message that has been altered or it is the origin that has been falsified. As a bottom-line, a receiver safely assumes that the message is not the genuine.

Limitations of MAC

There are two major limitations of MAC, both due to its symmetric nature of operation −

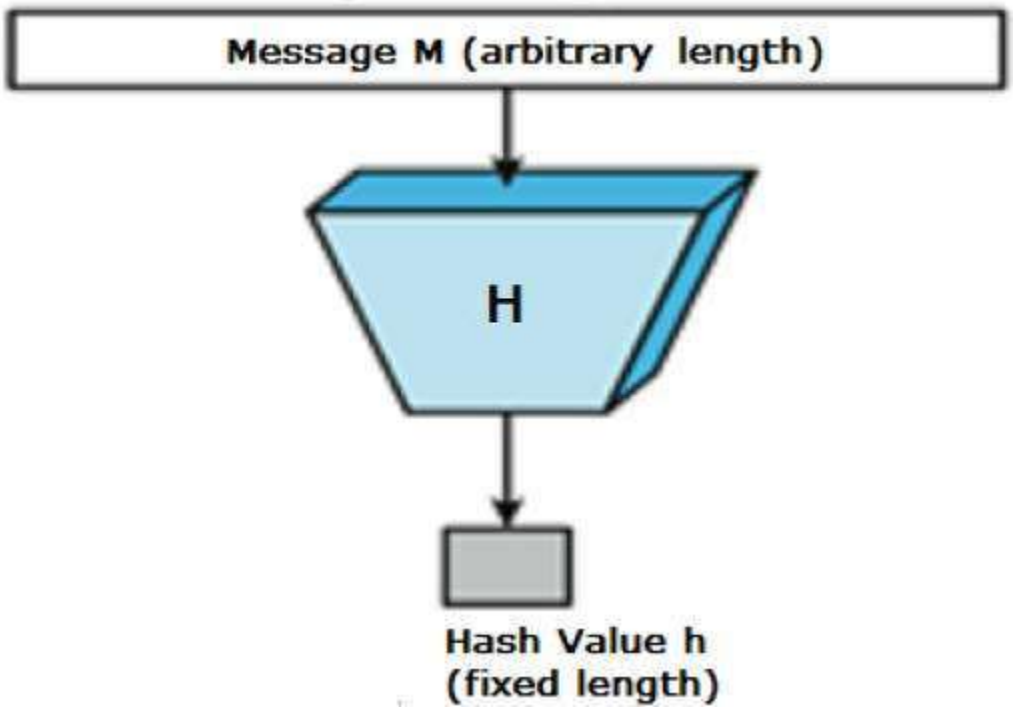
- **Establishment of Shared Secret.**
  - It can provide message authentication among pre-decided legitimate users who have shared key.
  - This requires establishment of shared secret prior to use of MAC.
- **Inability to Provide Non-Repudiation**
  - Non-repudiation is the assurance that a message originator cannot deny any previously sent messages and commitments or actions.
  - MAC technique does not provide a non-repudiation service. If the sender and receiver get involved in a dispute over message origination, MACs cannot provide a proof that a message was indeed sent by the sender.
  - Though no third party can compute the MAC, still sender could deny having sent the message and claim that the receiver forged it, as it is impossible to determine which of the two parties computed the MAC.

Both these limitations can be overcome by using the public key based digital signatures discussed in following section.

Hash functions are extremely useful and appear in almost all information security applications.

A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length.

Values returned by a hash function are called **message digest** or simply **hash values**. The following picture illustrated hash function −



Features of Hash Functions

The typical features of hash functions are −

- **Fixed Length Output (Hash Value)**
  - Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as **hashing the data**.
  - In general, the hash is much smaller than the input data, hence hash functions are sometimes called **compression functions**.
  - Since a hash is a smaller representation of a larger data, it is also referred to as a **digest**.
  - Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.
- **Efficiency of Operation**

- Generally for any hash function h with input x, computation of h(x) is a fast operation.
- Computationally hash functions are much faster than a symmetric encryption.

Properties of Hash Functions

In order to be an effective cryptographic tool, the hash function is desired to possess following properties −

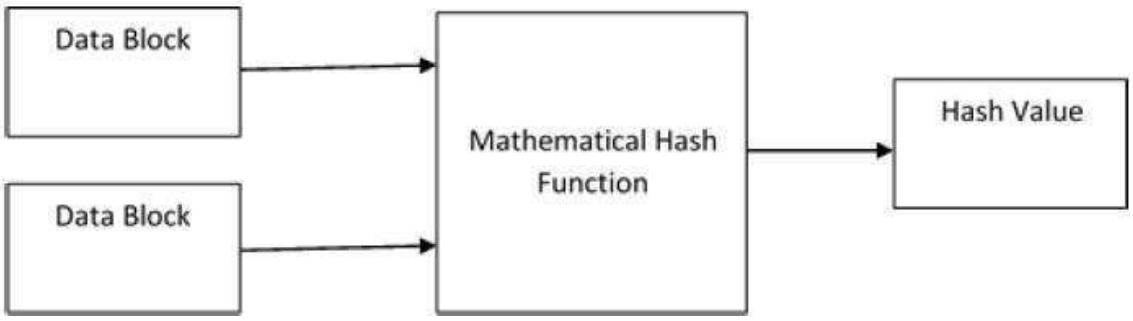
- **Pre-Image Resistance**
  - This property means that it should be computationally hard to reverse a hash function.
  - In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
  - This property protects against an attacker who only has a hash value and is trying to find the input.
- **Second Pre-Image Resistance**
  - This property means given an input and its hash, it should be hard to find a different input with the same hash.
  - In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
  - This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.
- **Collision Resistance**
  - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.

- In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).
- Since, hash function is compressing function with fixed hash length, it is impossible for a hash function not to have collisions. This property of collision free only confirms that these collisions should be hard to find.
- This property makes it very difficult for an attacker to find two input values with the same hash.
- Also, if a hash function is collision-resistant **then it is second pre-image resistant.**

Design of Hashing Algorithms

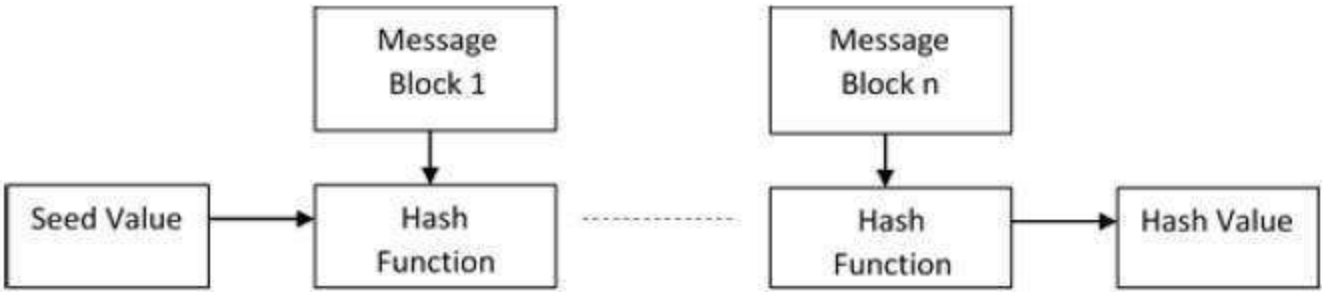
At the heart of a hashing is a mathematical function that operates on two fixed-size blocks of data to create a hash code. This hash function forms the part of the hashing algorithm.

The size of each data block varies depending on the algorithm. Typically the block sizes are from 128 bits to 512 bits. The following illustration demonstrates hash function −



Hashing algorithm involves rounds of above hash function like a block cipher. Each round takes an input of a fixed size, typically a combination of the most recent message block and the output of the last round.

This process is repeated for as many rounds as are required to hash the entire message. Schematic of hashing algorithm is depicted in the following illustration −



Since, the hash value of first message block becomes an input to the second hash operation, output of which alters the result of the third operation, and so on. This effect, known as an **avalanche**effect of hashing.

Avalanche effect results in substantially different hash values for two messages that differ by even a single bit of data.

Understand the difference between hash function and algorithm correctly. The hash function generates a hash code by operating on two blocks of fixed-length binary data.

Hashing algorithm is a process for using the hash function, specifying how the message will be broken up and how the results from previous message blocks are chained together.

Popular Hash Functions

Let us briefly see some popular hash functions −

Message Digest (MD)

MD5 was most popular and widely used hash function for quite some years.

- The MD family comprises of hash functions MD2, MD4, MD5 and MD6. It was adopted as Internet Standard RFC 1321. It is a 128-bit hash function.
- MD5 digests have been widely used in the software world to provide assurance about integrity of transferred file. For example, file servers often provide a pre-computed MD5 checksum for the files, so that a user can compare the checksum of the downloaded file to it.
- In 2004, collisions were found in MD5. An analytical attack was reported to be successful only in an hour by using computer cluster. This collision attack resulted in compromised MD5 and hence it is no longer recommended for use.

Secure Hash Function (SHA)

Family of SHA comprise of four SHA algorithms; SHA-0, SHA-1, SHA-2, and SHA-3. Though from same family, there are structurally different.

- The original version is SHA-0, a 160-bit hash function, was published by the National Institute of Standards and Technology (NIST) in 1993. It had few weaknesses and did not become very popular. Later in 1995, SHA-1 was designed to correct alleged weaknesses of SHA-0.
- SHA-1 is the most widely used of the existing SHA hash functions. It is employed in several widely used applications and protocols including Secure Socket Layer (SSL) security.
- In 2005, a method was found for uncovering collisions for SHA-1 within practical time frame making long-term employability of SHA-1 doubtful.
- SHA-2 family has four further SHA variants, SHA-224, SHA-256, SHA-384, and SHA-512 depending up on number of bits in their hash value. No successful attacks have yet been reported on SHA-2 hash function.
- Though SHA-2 is a strong hash function. Though significantly different, its basic design is still follows design of SHA-1. Hence, NIST called for new competitive hash function designs.
- In October 2012, the NIST chose the Keccak algorithm as the new SHA-3 standard. Keccak offers many benefits, such as efficient performance and good resistance for attacks.

RIPEMD

The RIPEMD is an acronym for RACE Integrity Primitives Evaluation Message Digest. This set of hash functions was designed by open research community and generally known as a family of European hash functions.

- The set includes RIPEMD, RIPEMD-128, and RIPEMD-160. There also exist 256, and 320-bit versions of this algorithm.

- Original RIPEMD (128 bit) is based upon the design principles used in MD4 and found to provide questionable security. RIPEMD 128-bit version came as a quick fix replacement to overcome vulnerabilities on the original RIPEMD.
- RIPEMD-160 is an improved version and the most widely used version in the family. The 256 and 320-bit versions reduce the chance of accidental collision, but do not have higher levels of security as compared to RIPEMD-128 and RIPEMD-160 respectively.

Whirlpool

This is a 512-bit hash function.

- It is derived from the modified version of Advanced Encryption Standard (AES). One of the designer was Vincent Rijmen, a co-creator of the AES.
- Three versions of Whirlpool have been released; namely WHIRLPOOL-0, WHIRLPOOL-T, and WHIRLPOOL.

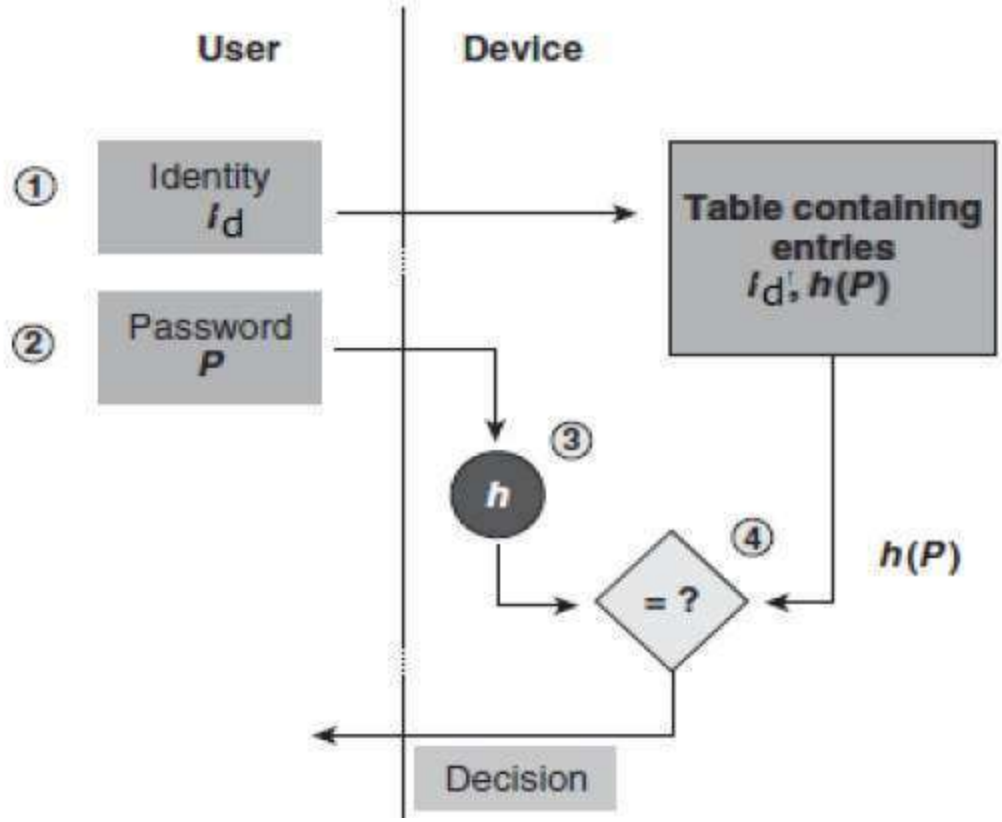
Applications of Hash Functions

There are two direct applications of hash function based on its cryptographic properties.

Password Storage

Hash functions provide protection to password storage.

- Instead of storing password in clear, mostly all logon processes store the hash values of passwords in the file.
- The Password file consists of a table of pairs which are in the form (user id, h(P)).
- The process of logon is depicted in the following illustration −

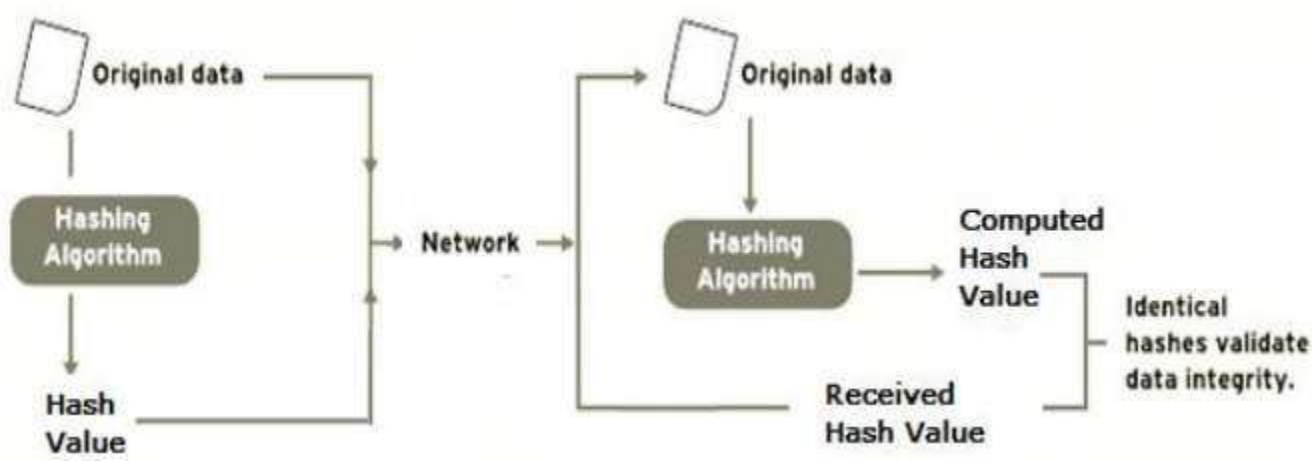


- An intruder can only see the hashes of passwords, even if he accessed the password. He can neither logon using hash nor can he derive the password from hash value since hash function possesses the property of pre-image resistance.

Data Integrity Check

Data integrity check is a most common application of the hash functions. It is used to generate the checksums on data files. This application provides assurance to the user about correctness of the data.

The process is depicted in the following illustration −



The integrity check helps the user to detect any changes made to original file. It however, does not provide any assurance about originality. The attacker, instead of modifying file data, can change the entire file and compute all together new hash and send to the receiver. This integrity check application is useful only if the user is sure about the originality of file.

UNIT V

DIGITAL SIGNATURES

Digital signatures are the public-key primitives of message authentication. In the physical world, it is common to use handwritten signatures on handwritten or typed messages. They are used to bind signatory to the message.

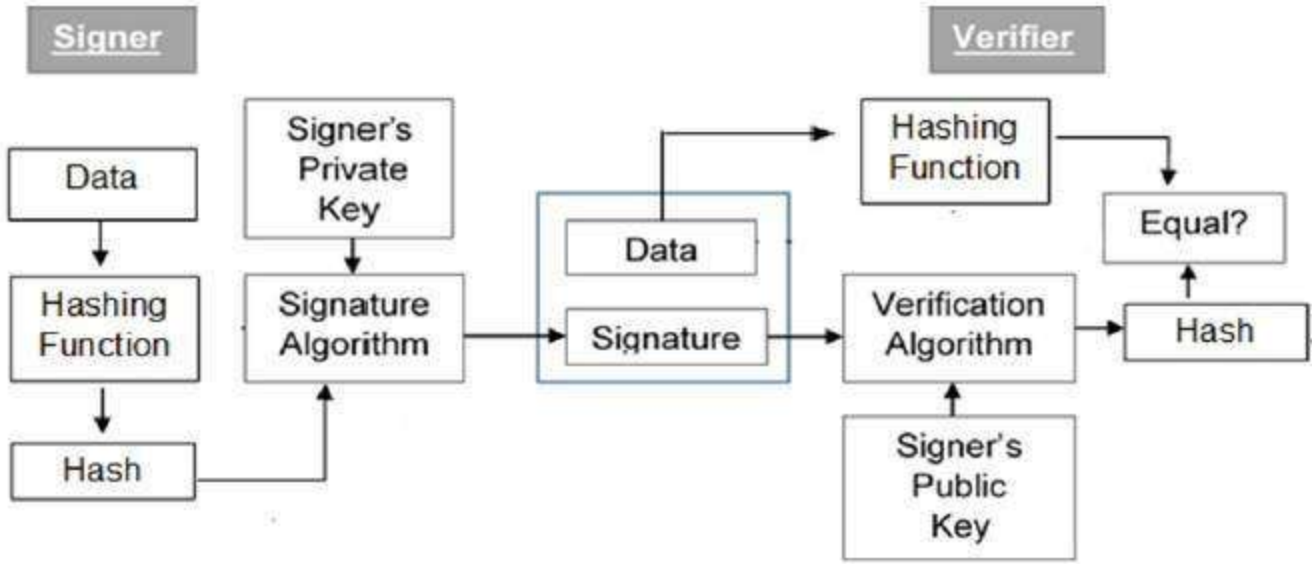
Similarly, a digital signature is a technique that binds a person/entity to the digital data. This binding can be independently verified by receiver as well as any third party.

Digital signature is a cryptographic value that is calculated from the data and a secret key known only by the signer.

In real world, the receiver of message needs assurance that the message belongs to the sender and he should not be able to repudiate the origination of that message. This requirement is very crucial in business applications, since likelihood of a dispute over exchanged data is very high.

Model of Digital Signature

As mentioned earlier, the digital signature scheme is based on public key cryptography. The model of digital signature scheme is depicted in the following illustration −



The following points explain the entire process in detail −

- Each person adopting this scheme has a public-private key pair.
- Generally, the key pairs used for encryption/decryption and signing/verifying are different. The private key used for signing is referred to as the signature key and the public key as the verification key.
- Signer feeds data to the hash function and generates hash of data.
- Hash value and signature key are then fed to the signature algorithm which produces the digital signature on given hash. Signature is appended to the data and then both are sent to the verifier.
- Verifier feeds the digital signature and the verification key into the verification algorithm. The verification algorithm gives some value as output.
- Verifier also runs same hash function on received data to generate hash value.
- For verification, this hash value and output of verification algorithm are compared. Based on the comparison result, verifier decides whether the digital signature is valid.
- Since digital signature is created by 'private' key of signer and no one else can have this key; the signer cannot repudiate signing the data in future.

It should be noticed that instead of signing data directly by signing algorithm, usually a hash of data is created. Since the hash of data is a unique representation of data, it is sufficient to sign the hash in place of data. The most important reason of using hash instead of data directly for signing is efficiency of the scheme.

Let us assume RSA is used as the signing algorithm. As discussed in public key encryption chapter, the encryption/signing process using RSA involves modular exponentiation.

Signing large data through modular exponentiation is computationally expensive and time consuming. The hash of the data is a relatively small digest of the data, hence **signing a hash is more efficient than signing the entire data**.

Importance of Digital Signature

Out of all cryptographic primitives, the digital signature using public key cryptography is considered as very important and useful tool to achieve information security.

Apart from ability to provide non-repudiation of message, the digital signature also provides message authentication and data integrity. Let us briefly see how this is achieved by the digital signature −

- **Message authentication** − When the verifier validates the digital signature using public key of a sender, he is assured that signature has been created only by sender who possess the corresponding secret private key and no one else.
- **Data Integrity** − In case an attacker has access to the data and modifies it, the digital signature verification at receiver end fails. The hash of modified data and the output provided by the verification algorithm will not match. Hence, receiver can safely deny the message assuming that data integrity has been breached.
- **Non-repudiation** − Since it is assumed that only the signer has the knowledge of the signature key, he can only create unique signature on a given data. Thus the receiver can present data and the digital signature to a third party as evidence if any dispute arises in the future.

By adding public-key encryption to digital signature scheme, we can create a cryptosystem that can provide the four essential elements of security namely − Privacy, Authentication, Integrity, and Non-repudiation.

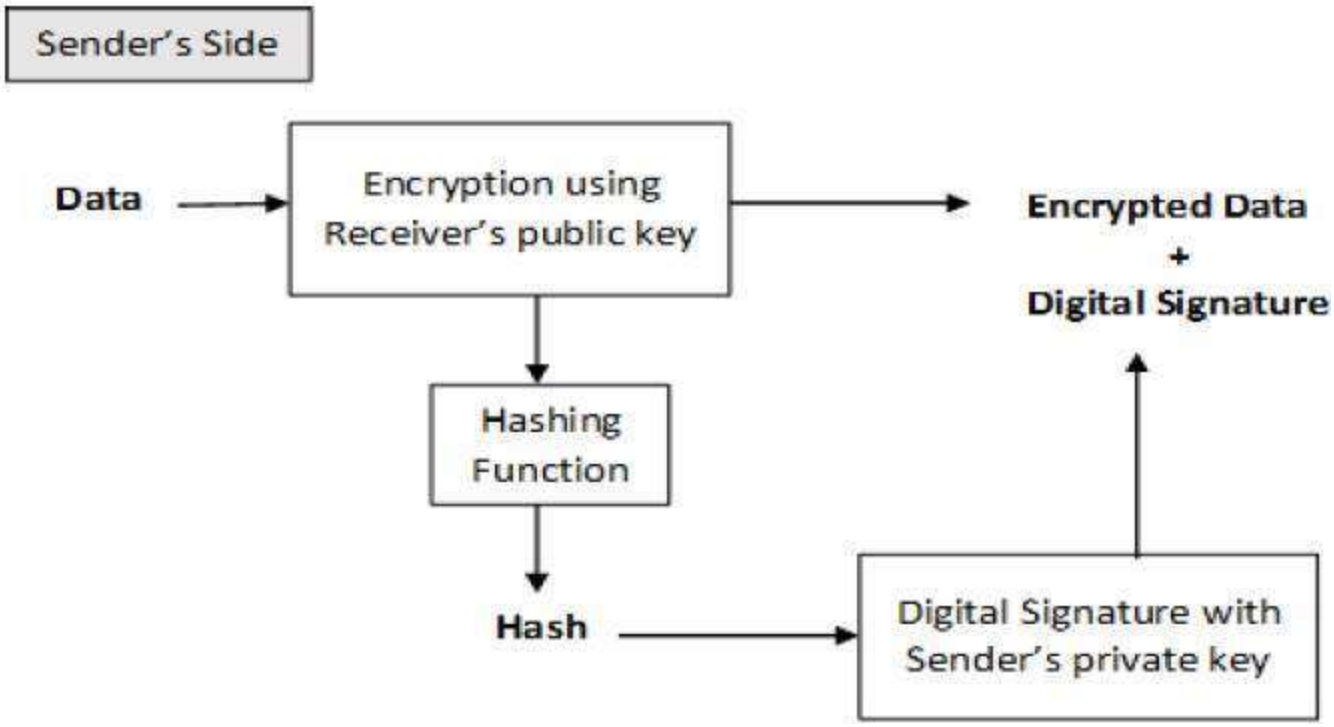
Encryption with Digital Signature

In many digital communications, it is desirable to exchange an encrypted messages than plaintext to achieve confidentiality. In public key encryption scheme, a public (encryption) key of sender is available in open domain, and hence anyone can spoof his identity and send any encrypted message to the receiver.

This makes it essential for users employing PKC for encryption to seek digital signatures along with encrypted data to be assured of message authentication and non-repudiation.

This can archived by combining digital signatures with encryption scheme. Let us briefly discuss how to achieve this requirement. There are **two possibilities, sign-then-encrypt** and **encrypt-then-sign**.

However, the crypto system based on sign-then-encrypt can be exploited by receiver to spoof identity of sender and sent that data to third party. Hence, this method is not preferred. The process of encrypt-then-sign is more reliable and widely adopted. This is depicted in the following illustration −



The receiver after receiving the encrypted data and signature on it, first verifies the signature using sender's public key. After ensuring the validity of the signature, he then retrieves the data through decryption using his private key.

The most distinct feature of Public Key Infrastructure (PKI) is that it uses a pair of keys to achieve the underlying security service. The key pair comprises of private key and public key.

Since the public keys are in open domain, they are likely to be abused. It is, thus, necessary to establish and maintain some kind of trusted infrastructure to manage these keys.

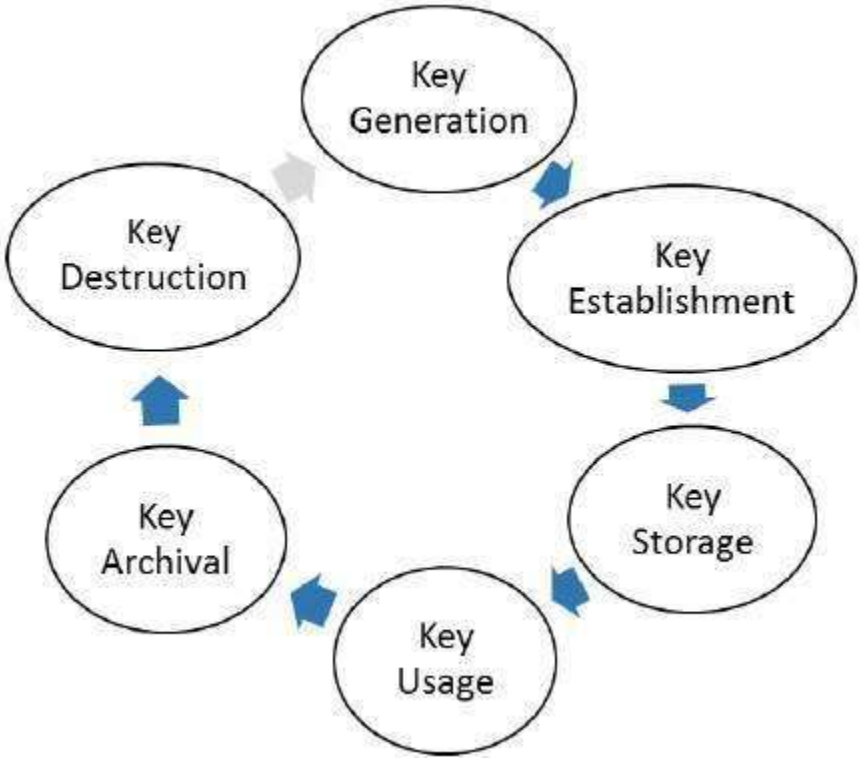
Key Management

It goes without saying that the security of any cryptosystem depends upon how securely its keys are managed. Without secure procedures for the handling of cryptographic keys, the benefits of the use of strong cryptographic schemes are potentially lost.

It is observed that cryptographic schemes are rarely compromised through weaknesses in their design. However, they are often compromised through poor key management.

There are some important aspects of key management which are as follows −

- Cryptographic keys are nothing but special pieces of data. Key management refers to the secure administration of cryptographic keys.
- Key management deals with entire key lifecycle as depicted in the following illustration –



- There are two specific requirements of key management for public key cryptography.
    - **Secrecy of private keys.**Throughout the key lifecycle, secret keys must remain secret from all parties except those who are owner and are authorized to use them.
    - **Assurance of public keys.** In public key cryptography, the public keys are in open domain and seen as public pieces of data. By default there are no assurances of whether a public key is correct, with whom it can be associated, or what it can be used for. Thus key management of public keys needs to focus much more explicitly on assurance of purpose of public keys.

The most crucial requirement of 'assurance of public key' can be achieved through the public-key infrastructure (PKI), a key management systems for supporting public-key cryptography.

Public Key Infrastructure (PKI)

PKI provides assurance of public key. It provides the identification of public keys and their distribution. An anatomy of PKI comprises of the following components.

- Public Key Certificate, commonly referred to as 'digital certificate'.
- Private Key tokens.
- Certification Authority.
- Registration Authority.
- Certificate Management System.

Digital Certificate

For analogy, a certificate can be considered as the ID card issued to the person. People use ID cards such as a driver's license, passport to prove their identity. A digital certificate does the same basic thing in the electronic world, but with one difference.

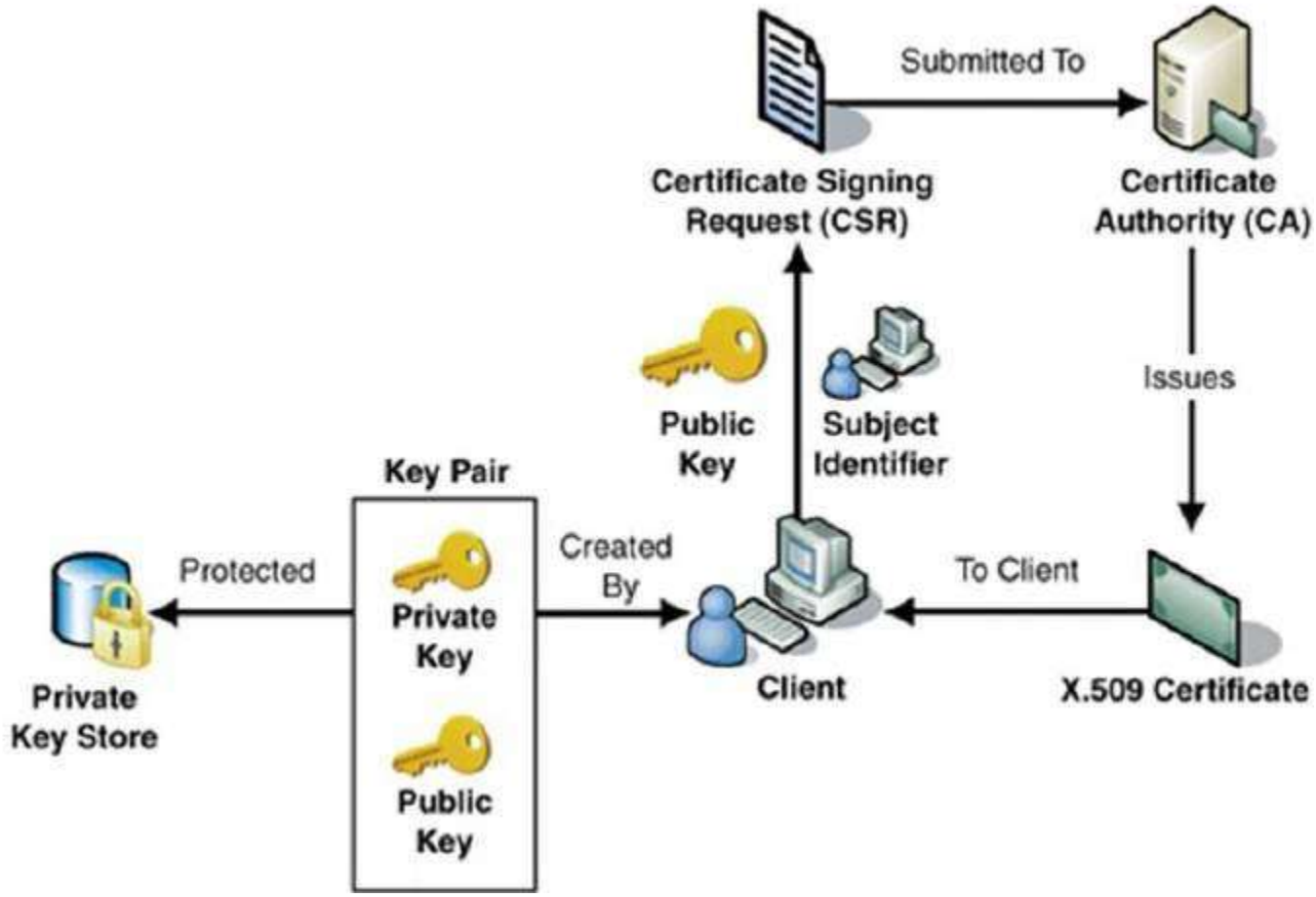
Digital Certificates are not only issued to people but they can be issued to computers, software packages or anything else that need to prove the identity in the electronic world.

- Digital certificates are based on the ITU standard X.509 which defines a standard certificate format for public key certificates and certification validation. Hence digital certificates are sometimes also referred to as X.509 certificates.

  Public key pertaining to the user client is stored in digital certificates by The Certification Authority (CA) along with other relevant information such as client information, expiration date, usage, issuer etc.

- CA digitally signs this entire information and includes digital signature in the certificate.
- Anyone who needs the assurance about the public key and associated information of client, he carries out the signature validation process using CA's public key. Successful validation assures that the public key given in the certificate belongs to the person whose details are given in the certificate.

The process of obtaining Digital Certificate by a person/entity is depicted in the following illustration.



As shown in the illustration, the CA accepts the application from a client to certify his public key. The CA, after duly verifying identity of client, issues a digital certificate to that client.

Certifying Authority (CA)

As discussed above, the CA issues certificate to a client and assist other users to verify the certificate. The CA takes responsibility for identifying correctly the identity of the client asking for a certificate to be issued, and ensures that the information contained within the certificate is correct and digitally signs it.

Key Functions of CA

The key functions of a CA are as follows −

- **Generating key pairs** − The CA may generate a key pair independently or jointly with the client.

- **Issuing digital certificates** − The CA could be thought of as the PKI equivalent of a passport agency − the CA issues a certificate after client provides the credentials to confirm his identity. The CA then signs the certificate to prevent modification of the details contained in the certificate.
- **Publishing Certificates** − The CA need to publish certificates so that users can find them. There are two ways of achieving this. One is to publish certificates in the equivalent of an electronic telephone directory. The other is to send your certificate out to those people you think might need it by one means or another.
- **Verifying Certificates** − The CA makes its public key available in environment to assist verification of his signature on clients' digital certificate.
- **Revocation of Certificates** − At times, CA revokes the certificate issued due to some reason such as compromise of private key by user or loss of trust in the client. After revocation, CA maintains the list of all revoked certificate that is available to the environment.

Classes of Certificates

There are four typical classes of certificate −

- **Class 1** − These certificates can be easily acquired by supplying an email address.
- **Class 2** − These certificates require additional personal information to be supplied.
- **Class 3** − These certificates can only be purchased after checks have been made about the requestor's identity.
- **Class 4** − They may be used by governments and financial organizations needing very high levels of trust.

Registration Authority (RA)

CA may use a third-party Registration Authority (RA) to perform the necessary checks on the person or company requesting the certificate to confirm their identity. The RA may appear to the client as a CA, but they do not actually sign the certificate that is issued.

Certificate Management System (CMS)

It is the management system through which certificates are published, temporarily or permanently suspended, renewed, or revoked. Certificate management systems do not normally delete certificates because it may be necessary to prove their status at a point in time, perhaps for legal reasons. A CA along with associated RA runs certificate management systems to be able to track their responsibilities and liabilities.

Private Key Tokens

While the public key of a client is stored on the certificate, the associated secret private key can be stored on the key owner's computer. This method is generally not adopted. If an attacker gains access to the computer, he can easily gain access to private key. For this reason, a private key is stored on secure removable storage token access to which is protected through a password.

Different vendors often use different and sometimes proprietary storage formats for storing keys. For example, Entrust uses the proprietary .epf format, while Verisign, GlobalSign, and Baltimore use the standard .p12 format.

Hierarchy of CA

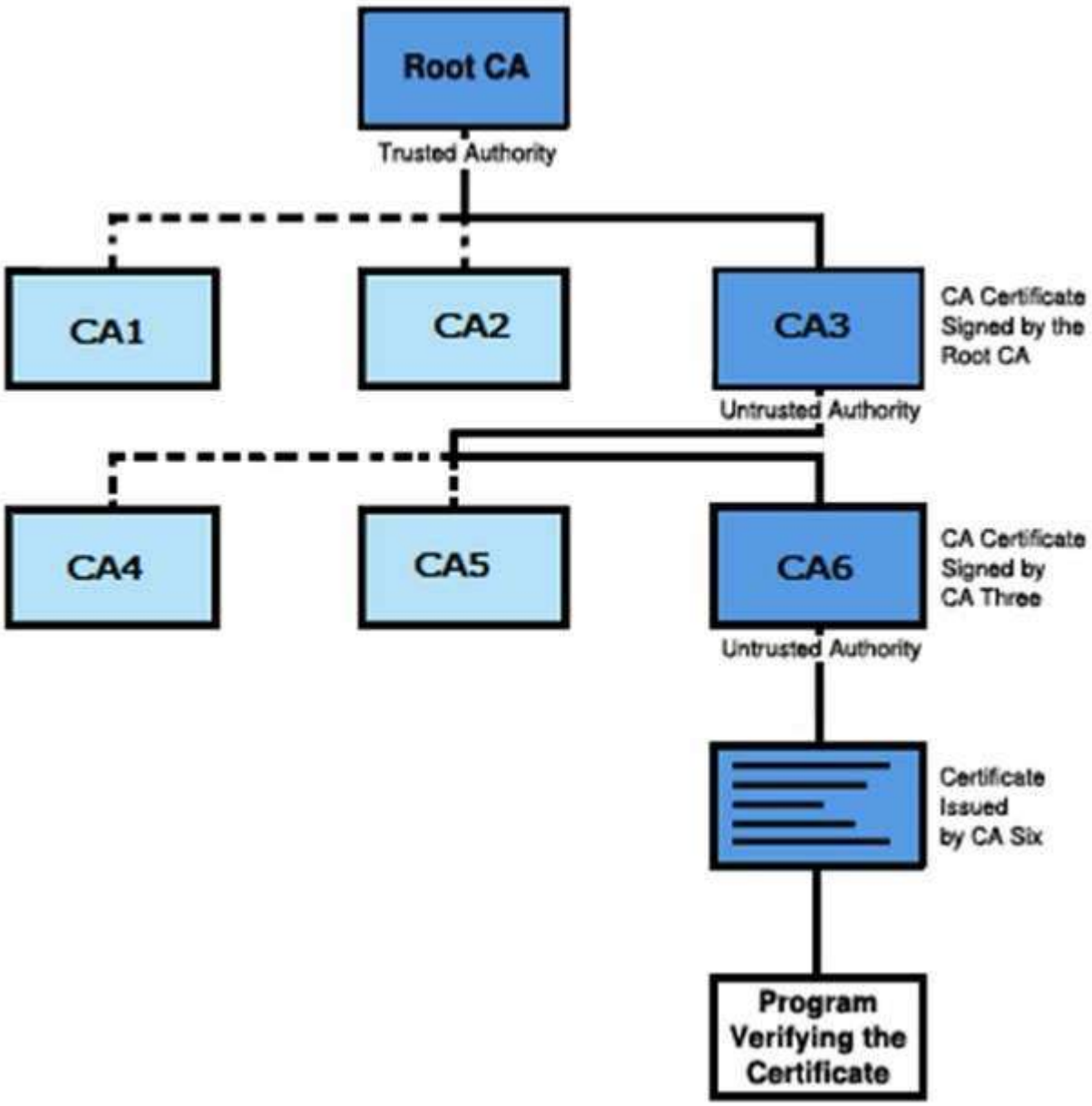
With vast networks and requirements of global communications, it is practically not feasible to have only one trusted CA from whom all users obtain their certificates. Secondly, availability of only one CA may lead to difficulties if CA is compromised.

In such case, the hierarchical certification model is of interest since it allows public key certificates to be used in environments where two communicating parties do not have trust relationships with the same CA.

- The root CA is at the top of the CA hierarchy and the root CA's certificate is a self-signed certificate.
- The CAs, which are directly subordinate to the root CA (For example, CA1 and CA2) have CA certificates that are signed by the root CA.
- The CAs under the subordinate CAs in the hierarchy (For example, CA5 and CA6) have their CA certificates signed by the higher-level subordinate CAs.

Certificate authority (CA) hierarchies are reflected in certificate chains. A certificate chain traces a path of certificates from a branch in the hierarchy to the root of the hierarchy.

The following illustration shows a CA hierarchy with a certificate chain leading from an entity certificate through two subordinate CA certificates (CA6 and CA3) to the CA certificate for the root CA.



Verifying a certificate chain is the process of ensuring that a specific certificate chain is valid, correctly signed, and trustworthy. The following procedure verifies a certificate chain, beginning with the certificate that is presented for authentication −

- A client whose authenticity is being verified supplies his certificate, generally along with the chain of certificates up to Root CA.
- Verifier takes the certificate and validates by using public key of issuer. The issuer's public key is found in the issuer's certificate which is in the chain next to client's certificate.
- Now if the higher CA who has signed the issuer's certificate, is trusted by the verifier, verification is successful and stops here.
- Else, the issuer's certificate is verified in a similar manner as done for client in above steps. This process continues till either trusted CA is found in between or else it continues till Root CA.

Digital Signatures

Digital Signatures

Requirements

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.

For example, suppose that John sends an authenticated message to Mary, using one of the schemes of Figure 11.4. Consider the following disputes that could arise:

1. Mary may forge a different message and claim that it came from John. Mary would simply have to create a message and append an authentication code using the key that

John and Mary share.

2. John can deny sending the message. Because it is possible for Mary to forge a message, there is no way to prove that John 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.

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 is analogous to the handwritten signature. It must have the following properties:

- It must verify the author and the date and time of the signature.
- It must to 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.

On the basis of these properties, 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 11.5c or d, satisfies these requirements.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

Direct Digital Signature

The direct digital signature involves only the communicating parties (source, destination). It is assumed that the destination knows the public key of the source. A digital signature may be formed by encrypting the entire message with the sender's private key (Figure 11.1c) or by encrypting a hash code of the message with the sender's private key (Figure 11.5c).

[Page 380]

Confidentiality can be provided by further encrypting the entire message plus signature with either the receiver's public key (public-key encryption) or a shared secret key (symmetric encryption); for example, see Figures 11.1d and 11.5d. 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.

All direct schemes described so far share a common weakness. The validity of the scheme 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.

Another threat is that some private key might actually be stolen from X at time T. The opponent

can then send a message signed with X's signature and stamped with a time before or equal to T.

Arbitrated Digital Signature

The problems associated with direct digital signatures can be addressed by using an arbiter.

As with direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as follows. Every signed message from a sender X to a receiver Y goes first to an arbiter A, who subjects the message and its signature to a number of tests to check its origin and content. The message is then dated and sent to Y with an indication that it has been verified to the satisfaction of the arbiter. The presence of A solves the problem faced by direct signature schemes: that X might disown the message.

The arbiter plays a sensitive and crucial role in this sort of scheme, and all parties must have a great deal of trust that the arbitration mechanism is working properly. The use of a trusted system, described in Chapter 20, might satisfy this requirement.

Table 13.1, based on scenarios described in [AKL83] and [MITC92], gives several examples of arbitrated digital signatures.[1] In the first, symmetric encryption is used. It is assumed that the sender X and the arbiter A share a secret key K_{xa} and that A and Y share secret key K_{ay}. X constructs a message M and computes its hash value H(M). Then X transmits the message plus a signature to A. The signature consists of an identifier ID_X of X plus the hash value, all encrypted using K_{xa}. A decrypts the signature and checks the hash value to validate the message. Then A transmits a message to Y, encrypted with K_{ay}. The message includes ID_X, the original message from X, the signature, and a timestamp. Y can decrypt this to recover the message and the signature. The timestamp informs Y that this message is timely and not a replay. Y can store M and the signature. In case of dispute, Y, who claims to have received Mfrom X, sends the following message to A:

[1] The following format is used. A communication step in which P sends a message M to Q is represented as P $\longrightarrow$ Q:

$E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])])$

Table 13.1. Arbitrated Digital Signature Techniques

(1) X ⟶ A: K_{xa}, $[ID_X||H(M)]$)	
(2) A ⟶ Y: $E(ID_X||M||E(K_{xa}, [ID_X||H(M)])||T])$	
(a) Conventional Encryption, Arbiter Sees Message	
(1) X ⟶ A: K_{xy}, $M)||E(K_{xa}, [ID_X||H(E(K_{xy}, M))])$	
(2) A ⟶ Y: $E(ID_X||E(K_{xy}, M)])||E(K_{xa}, [ID_X||H(E(K_{xy}, M))||T])$	
(b) Conventional Encryption, Arbiter Does Not See Message	
(1) X ⟶ A: PR_x, $[ID_X||E(PU_y, E(PR_x, M))])$	
(2) A ⟶ Y: $E(ID_X||E(PU_y, E(PR_x, M))||T])$	
(c) Public-Key Encryption, Arbiter Does Not See Message	
Notation:	
X	= sender
Y	= recipient
A	= Arbiter
M	= message
T	= timestamp

The arbiter uses K_{ay} to recover ID_X, M, and the signature, and then uses K_{xa} to decrypt the signature and verify the hash code. In this scheme, Y cannot directly check X's signature; the signature is there solely to settle disputes. Y considers the message from X authentic because it comes through A. In this scenario, both sides must have a high degree of trust in A:

- X must trust A not to reveal K_{xa} and not to generate false signatures of the form $E(K_{xa}, [ID_X||H(M)])$.
- Y must trust A to send $E(K_{ay}, [ID_X||M||E(K_{xa}, [ID_X||H(M)])||T])$ only if the hash value is correct and the signature was generated by X.
- Both sides must trust A to resolve disputes fairly.

If the arbiter does live up to this trust, then X is assured that no one can forge his signature and Y

is assured that X cannot disavow his signature.

The preceding scenario also implies that A is able to read messages from X to Y and, indeed, that any eavesdropper is able to do so. Table 13.1b shows a scenario that provides the arbitration as before but also assures confidentiality. In this case it is assumed that X and Y share the secret key K_{xy}. Now, X transmits an identifier, a copy of the message encrypted with K_{xy}, and a signature to A. The signature consists of the identifier plus the hash value of the encrypted message, all encrypted using K_{xa}. As before, A decrypts the signature and checks the hash value to validate the message. In this case, A is working only with the encrypted version of the message and is prevented from reading it. A then transmits everything that it received from X, plus a timestamp, all encrypted with K_{ay}, to Y.

[Page 382]

Although unable to read the message, the arbiter is still in a position to prevent fraud on the part of either X or Y. A remaining problem, one shared with the first scenario, is that the arbiter could form an alliance with the sender to deny a signed message, or with the receiver to forge the sender's signature.

All the problems just discussed can be resolved by going to a public-key scheme, one version of which is shown in Table 13.1c. In this case, X double encrypts a message Mfirst with X's private key, PR_x and then with Y's public key, PU_y. This is a signed, secret version of the message. This signed message, together with X's identifier, is encrypted again with PR_x and, together with ID_X, is sent to A. The inner, double-encrypted message is secure from the arbiter (and everyone else except Y). However, A can decrypt the outer encryption to assure that the message must have come from X (because only X has PR_x). A checks to make sure that X's private/public key pair is still valid and, if so, verifies the message. Then A transmits a message to Y, encrypted with PR_a. The message includes ID_X, the double-encrypted message, and a timestamp.

This scheme has a number of advantages over the preceding two schemes. First, no information is shared among the parties before communication, preventing alliances to defraud. Second, no incorrectly dated message can be sent, even if PR_x is compromised, assuming that PR_a is not

compromised. Finally, the content of the message from X to Y is secret from A and anyone else. However, this final scheme involves encryption of the message twice with a public-key algorithm. We discuss more practical approaches subsequently.

Authentication Protocols

Authentication Protocols

In this section, we focus on two general areas (mutual authentication and one-way authentication) and examine some of the implications of authentication techniques in both.

Mutual Authentication

An important application area is that of mutual authentication protocols. Such protocols enable communicating parties to satisfy themselves mutually about each other's identity and to exchange session keys. There, the focus was key distribution.

Central to the problem of authenticated key exchange are two issues: confidentiality and timeliness. To prevent masquerade and to prevent compromise of session keys, essential identification and session key information must be communicated in encrypted form. This requires the prior existence of secret or public keys that can be used for this purpose. The second issue, timeliness, is important because of the threat of message replays. Such replays, at worst, could allow an opponent to compromise a session key or successfully impersonate another party. At minimum, a successful replay can disrupt operations by presenting parties with messages that appear genuine but are not.

---

the following examples are of replay attacks:

- Simple replay: The opponent simply copies a message and replays it later.
- Repetition that can be logged: An opponent can replay a timestamped message within the valid time window.

- Repetition that cannot be detected: This situation could arise because the original message could have been suppressed and thus did not arrive at its destination; only the replay message arrives.
- Backward replay without modification: This is a replay back to the message sender. This attack is possible if symmetric encryption is used and the sender cannot easily recognize the difference between messages sent and messages received on the basis of content.

One approach to coping with replay attacks is to attach a sequence number to each message used in an authentication exchange. A new message is accepted only if its sequence number is in the proper order. The difficulty with this approach is that it requires each party to keep track of the last sequence number for each claimant it has dealt with. Because of this overhead, sequence numbers are generally not used for authentication and key exchange. Instead, one of the following two general approaches is used:

- Timestamps: Party A accepts a message as fresh only if the message contains a timestamp that, in A's judgment, is close enough to A's knowledge of current time. This approach requires that clocks among the various participants be synchronized.
- Challenge/response: Party A, expecting a fresh message from B, first sends B a nonce(challenge) and requires that the subsequent message (response) received from B contain the correct nonce value.

Symmetric Encryption Approaches

A a two-level hierarchy of symmetric encryption keys can be used to provide confidentiality for communication in a distributed environment. In general, this strategy involves the use of a trusted key distribution center (KDC). Each party in the network shares a secret key, known as a master key, with the KDC. The KDC is responsible for generating keys to be used for a short time over a connection between two parties, known as session keys, and for distributing those keys using the master keys to protect the distribution.

The following Figure illustrates a proposal initially put forth by Needham and Schroeder for secret key distribution using a KDC , includes authentication features. The protocol can be summarized as follows:

1.A $\longrightarrow$ KDC:$ID_A||ID_B||N_1$

2.KDC $\longrightarrow$ A:$E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$

3.A $\longrightarrow$ B: $E(K_b, [K_s||ID_A])$

4.A $\longrightarrow$ A: $E(K_s, N_2)$

5.A $\longrightarrow$ B: $E(K_s, f(N_2))$

Secret keys K_a and K_b are shared between A and the KDC and B and the KDC, respectively. The purpose of the protocol is to distribute securely a session key K_s to A and B. A securely acquires a new session key in step 2. The message in step 3 can be decrypted, and hence understood, only by B. Step 4 reflects B's knowledge of K_s, and step 5 assures B of A's knowledge of K_s and assures B that this is a fresh message because of the use of the nonce N_2. Recall from our discussion in Chapter 7 that the purpose of steps 4 and 5 is to prevent a certain type of replay attack. In particular, if an opponent is able to capture the message in step 3 and replay it, this might in some fashion disrupt operations at B.

Despite the handshake of steps 4 and 5, the protocol is still vulnerable to a form of replay attack. Suppose that an opponent, X, has been able to compromise an old session key. Admittedly, this is a much more unlikely occurrence than that an opponent has simply observed and recorded step 3. Nevertheless, it is a potential security risk. X can impersonate A and trick B into using the old key by simply replaying step 3. Unless B remembers indefinitely all previous session keys used with A, B will be unable to determine that this is a replay. If X can intercept the handshake message, step 4, then it can impersonate A's response, step 5. From this point on, X can send bogus messages to B that appear to B to come from A using an authenticated session key.

Denning proposes to overcome this weakness by a modification to the Needham/Schroeder protocol that includes the addition of a timestamp to steps 2 and 3. Her proposal assumes that the master keys, $K_a$and K_b are secure, and it consists of the following steps:

1.A $\longrightarrow$ KDC:$ID_A||ID_B$

2.KDC ⟶ A:$E(K_a, [K_s||ID_B||T||E(K_b, [K_s||ID_A||T])])$

3.A ⟶ B: $E(K_b, [K_s||ID_A||T])$

4.B ⟶ A: $E(K_s, N_1)$

5.A ⟶ B: $E(K_s, f(N_1))$

T is a timestamp that assures A and B that the session key has only just been generated. Thus, both A and B know that the key distribution is a fresh exchange. A and B can verify timeliness by checking that

$|\text{Clock } T| < \Delta t_1 + \Delta t_2$

where Δt_1 is the estimated normal discrepancy between the KDC's clock and the local clock (at A or B) and Δt_2 is the expected network delay time. Each node can set its clock against some standard reference source. Because the timestamp T is encrypted using the secure master keys, an opponent, even with knowledge of an old session key, cannot succeed because a replay of step 3 will be detected by B as untimely.

A final point: Steps 4 and 5 were not included in the original presentation [DENN81] but were added later [DENN82]. These steps confirm the receipt of the session key at B.

The Denning protocol seems to provide an increased degree of security compared to the Needham/Schroeder protocol. However, a new concern is raised: namely, that this new scheme requires reliance on clocks that are synchronized throughout the network. [GONG92] points out a risk involved. The risk is based on the fact that the distributed clocks can become unsynchronized as a result of sabotage on or faults in the clocks or the synchronization mechanism.[2] The problem occurs when a sender's clock is ahead of the intended recipient's clock. In this case, an opponent can intercept a message from the sender and replay it later when the timestamp in the message becomes current at the recipient's site. This replay could cause unexpected results. Gong refers to such attacks as **suppress-replay attacks**.

One way to counter suppress-replay attacks is to enforce the requirement that parties regularly check their clocks against the KDC's clock. The other alternative, which avoids the need for

clock synchronization, is to rely on handshaking protocols using nonces. This latter alternative is not vulnerable to a suppress-replay attack because the nonces the recipient will choose in the future are unpredictable to the sender. The Needham/Schroeder protocol relies on nonces only but, as we have seen, has other vulnerabilities.

An attempt is made to respond to the concerns about suppress-replay attacks and at the same time fix the problems in the Needham/Schroeder protocol. The protocol is as follows:

[3] It really is hard to get these things right.

1.A $\longrightarrow$ B: $ID_A||N_a$

2.B $\longrightarrow$ KDC: $ID_B||N_b||E(K_b, [ID_A||N_a||T_b])$

3.KDC $\longrightarrow$ A: $E(K_a, [ID_B||N_a||K_s||T_b])||E(K_b,[ID_A||K_s||T_b])||N_b$

4.A $\longrightarrow$ B: $E(K_b, [ID_A||K_s||T_b])||E(K_s, N_b)$

[Page 386]

Let us follow this exchange step by step.

1. A initiates the authentication exchange by generating a nonce, N_a, and sending that plus its identifier to B in plaintext. This nonce will be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

2. B alerts the KDC that a session key is needed. Its message to the KDC includes its identifier and a nonce, N_b This nonce will be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B's message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A; the block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A.

3. The KDC passes on to A B's nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a "ticket" that can be used by A for subsequent

authentications, as will be seen. The KDC also sends to A a block encrypted with the secret key shared by A and the KDC. This block verifies that B has received A's initial message (ID_B) and that this is a timely message and not a replay (N_a) and it provides A with a session key (K_s) and the time limit on its use (T_b).

4. A transmits the ticket to B, together with the B's nonce, the latter encrypted with the session key. The ticket provides B with the secret key that is used to decrypt $E(K_s, N_b)$ to recover the nonce. The fact that B's nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

This protocol provides an effective, secure means for A and B to establish a session with a secure session key. Furthermore, the protocol leaves A in possession of a key that can be used for subsequent authentication to B, avoiding the need to contact the authentication server repeatedly. Suppose that A and B establish a session using the aforementioned protocol and then conclude that session. Subsequently, but within the time limit established by the protocol, A desires a new session with B. The following protocol ensues:

1.A $\longrightarrow$ B:$E(K_b, [ID_A||K_s||T_b])||N'_a$
2.B $\longrightarrow$ A:$N'_b||E(K_s, N'_a)$
3.A $\longrightarrow$ B:$E(K_s, N'_b)$

When B receives the message in step 1, it verifies that the ticket has not expired. The newly generated nonces N'_a and N'_b assure each party that there is no replay attack.

In all the foregoing, the time specified in T_b is a time relative to B's clock. Thus, this timestamp does not require synchronized clocks because B checks only self-generated timestamps.

Public-Key Encryption Approaches

We presented one approach to the use of public-key encryption for the purpose of session key distribution. This protocol assumes that each of the two parties is in possession of the current public key of the other. It may not be practical to require this assumption.

A protocol using timestamps is provided in:

1.A $\longrightarrow$ AS:$ID_A||ID_B$
2.AS $\longrightarrow$ A:$E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
3.A $\longrightarrow$ B: $E(PR_{as}, [ID_A||PU_a||T])||E(PR_{as}, [ID_B||PU_b||T])$
$||E(PU_b, E(PR_a, [K_s||T]))$

In this case, the central system is referred to as an authentication server (AS), because it is not actually responsible for secret key distribution. Rather, the AS provides public-key certificates. The session key is chosen and encrypted by A; hence, there is no risk of exposure by the AS. The timestamps protect against replays of compromised keys.

This protocol is compact but, as before, requires synchronization of clocks. Another approach, proposed by Woo and Lam makes use of nonces. The protocol consists of the following steps:

1.A $\longrightarrow$ KDC:$ID_A||ID_B$
2.KDC $\longrightarrow$ A:$E(PR_{auth}, [ID_B||PU_b])$
3.A $\longrightarrow$ B: $E(PU_b, [N_a||ID_A])$
4.B $\longrightarrow$ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$
5.KDC $\longrightarrow$ B:$E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_B]))$
6.B $\longrightarrow$ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_B)||N_b]))$
7.A $\longrightarrow$ B: $E(K_s, N_b)$

In step 1, A informs the KDC of its intention to establish a secure connection with B. The KDC returns to A a copy of B's public-key certificate (step 2). Using B's public key, A informs B of its desire to communicate and sends a nonce N_a(step 3). In step 4, B asks the KDC for A's public-key certificate and requests a session key; B includes A's nonce so that the KDC can stamp the session key with that nonce. The nonce is protected using the KDC's public key. In step 5, the KDC returns to B a copy of A's public-key certificate, plus the information $\{N_a, K_s, ID_B\}$. This information basically says that K_s is a secret key generated by the KDC on behalf of B and tied

to N_a; the binding of K_s and N_a will assure A that K_s is fresh. This triple is encrypted, using the KDC's private key, to allow B to verify that the triple is in fact from the KDC. It is also encrypted using B's public key, so that no other entity may use the triple in an attempt to establish a fraudulent connection with A. In step 6, the triple $\{N_a, K_s, ID_B\}$, still encrypted with the KDC's private key, is relayed to A, together with a nonce N_b generated by B. All the foregoing are encrypted using A's public key. A retrieves the session key K_s and uses it to encrypt N_b and return it to B. This last message assures B of A's knowledge of the session key.

This seems to be a secure protocol that takes into account the various attacks. However, the authors themselves spotted a flaw and submitted a revised version of the algorithm in [WOO92b]:

1.A $\longrightarrow$ KDC: $ID_A||ID_B$

2.KDC $\longrightarrow$ A: $E(PR_{auth}, [ID_B||PU_b])$

3.A $\longrightarrow$ B: $E(PU_b, [N_a||ID_A])$

4.B $\longrightarrow$ KDC: $ID_A||ID_B||E(PU_{auth}, N_a)$

5.KDC $\longrightarrow$ B: $E(PR_{auth}, [ID_A||PU_a])||E(PU_b, E(PR_{auth}, [N_a||K_s||ID_A||ID_B])$

6.B $\longrightarrow$ A: $E(PU_a, E(PR_{auth}, [(N_a||K_s||ID_A||ID_B)||N_b]))$

7.A $\longrightarrow$ B: $E(K_s, N_b)$

The identifier of A, ID_A, is added to the set of items encrypted with the KDC's private key in steps 5 and 6. This binds the session key K_s to the identities of the two parties that will be engaged in the session. This inclusion of ID_Aaccounts for the fact that the nonce value N_a is considered unique only among all nonces generated by A, not among all nonces generated by all parties. Thus, it is the pair $\{ID_A, N_a\}$ that uniquely identifies the connection request of A.

In both this example and the protocols described earlier, protocols that appeared secure were revised after additional analysis. These examples highlight the difficulty of getting things right in the area of authentication.

One-Way Authentication

One application for which encryption is growing in popularity is electronic mail (e-mail). The

very nature of electronic mail, and its chief benefit, is that it is not necessary for the sender and receiver to be online at the same time. Instead, the e-mail message is forwarded to the receiver's electronic mailbox, where it is buffered until the receiver is available to read it.

The "envelope" or header of the e-mail message must be in the clear, so that the message can be handled by the store-and-forward e-mail protocol, such as the Simple Mail Transfer Protocol (SMTP) or X.400. However, it is often desirable that the mail-handling protocol not require access to the plaintext form of the message, because that would require trusting the mail-handling mechanism. Accordingly, the e-mail message should be encrypted such that the mail-handling system is not in possession of the decryption key.

A second requirement is that of authentication. Typically, the recipient wants some assurance that the message is from the alleged sender.

Symmetric Encryption Approach

Using symmetric encryption, the decentralized key distribution scenario illustrated in Figure 7.11 is impractical. This scheme requires the sender to issue a request to the intended recipient, await a response that includes a session key, and only then send the message.

With some refinement, the KDC strategy illustrated in Figure 7.9 is a candidate for encrypted electronic mail. Because we wish to avoid requiring that the recipient (B) be on line at the same time as the sender (A), steps 4 and 5 must be eliminated. For a message with content M, the sequence is as follows:

1.A $\longrightarrow$ KDC: $ID_A||ID_B||N_1$

2.KDC $\longrightarrow$ A: $E(K_a, [K_s||ID_B||N_1||E(K_b, [K_s||ID_A])])$

3.A $\longrightarrow$ B: $E(K_b, [K_s||ID_A])||E(K_s, M)$

This approach guarantees that only the intended recipient of a message will be able to read it. It also provides a level of authentication that the sender is A. As specified, the protocol does not protect against replays. Some measure of defense could be provided by including a timestamp with the message. However, because of the potential delays in the e-mail process, such

timestamps may have limited usefulness.

[Page 389]

Public-Key Encryption Approaches

We have already presented public-key encryption approaches that are suited to electronic mail, including the straightforward encryption of the entire message for confidentiality (Figure 11.1b), authentication (Figure 11.1c), or both (Figure 11.1d). These approaches require that either the sender know the recipient's public key (confidentiality) or the recipient know the sender's public key (authentication) or both (confidentiality plus authentication). In addition, the public-key algorithm must be applied once or twice to what may be a long message.

If confidentiality is the primary concern, then the following may be more efficient:

$A \longrightarrow B: E(K_s, M)$

In this case, the message is encrypted with a one-time secret key. A also encrypts this one-time key with B's public key. Only B will be able to use the corresponding private key to recover the one-time key and then use that key to decrypt the message. This scheme is more efficient than simply encrypting the entire message with B's public key.

If authentication is the primary concern, then a digital signature may suffice, as was illustrated in Figure 11.5c:

$A \longrightarrow B:PR_a, H(M))$

This method guarantees that A cannot later deny having sent the message. However, this technique is open to another kind of fraud. Bob composes a message to his boss Alice that contains an idea that will save the company money. He appends his digital signature and sends it into the e-mail system. Eventually, the message will get delivered to Alice's mailbox. But suppose that Max has heard of Bob's idea and gains access to the mail queue before delivery. He finds Bob's message, strips off his signature, appends his, and requeues the message to be delivered to Alice. Max gets credit for Bob's idea.

To counter such a scheme, both the message and signature can be encrypted with the recipient's public key:

$A \longrightarrow B: E(M||E(PR_a, H(M))])$

The latter two schemes require that B know A's public key and be convinced that it is timely. An effective way to provide this assurance is the digital certificate, described in Chapter 10. Now we have

$A \longrightarrow B:PR_a, H(M))||E(PR_{as}, [T||ID_A||PU_a])$

In addition to the message, A sends B the signature, encrypted with A's private key, and A's certificate, encrypted with the private key of the authentication server. The recipient of the message first uses the certificate to obtain the sender's public key and verify that it is authentic and then uses the public key to verify the message itself. If confidentiality is required, then the entire message can be encrypted with B's public key. Alternatively, the entire message can be encrypted with a one-time secret key; the secret key is also transmitted, encrypted with B's public key. This approach is explored in Chapter 15.

PREVIOUS PAGE

Digital Signature Standard

[Page 390]

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) described in Chapter 12 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

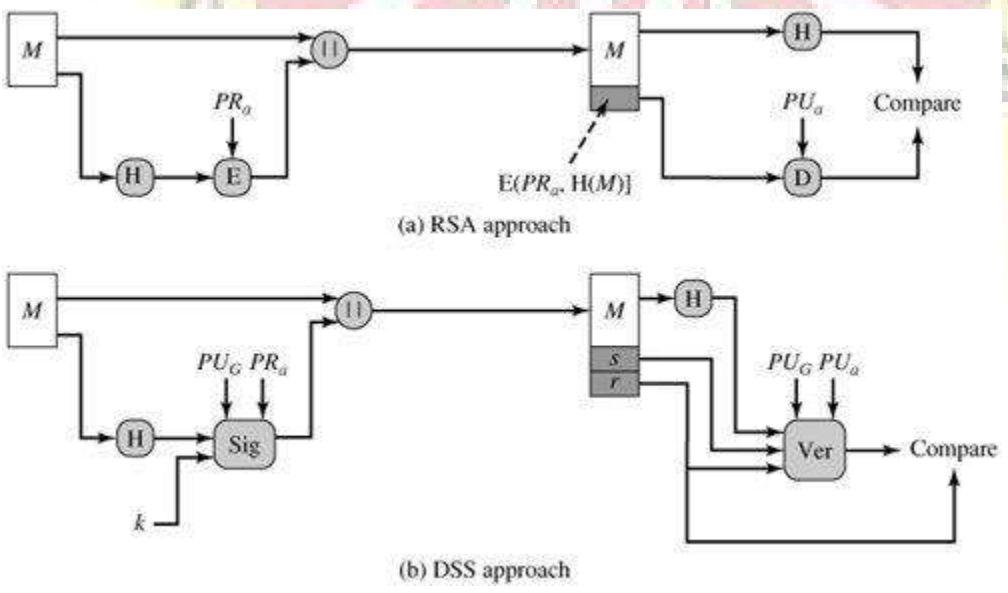
issued as FIPS 186-2. This latest version also incorporates digital signature algorithms based on RSA and on elliptic curve cryptography. In this section, we discuss the original DSS algorithm.

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 13.1 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.

Figure 13.1. Two Approaches to Digital Signatures



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 kgenerated 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).[4] The result is a signature consisting of two components, labeled s and r.

[4] It is also possible to allow these additional parameters to vary with each user so that they are a part of a user's public key. In practice, it is more likely that a global public key will be used that is separate from each user's public key.

[Page 391]

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 (PU_a), 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.

We turn now to the details of the algorithm.

The Digital Signature Algorithm

The DSA is based on the difficulty of computing discrete logarithms (see Chapter 8) and is based on schemes originally presented by ElGamal [ELGA85] and Schnorr [SCHN91].

Figure 13.2 summarizes the algorithm. There are three parameters that are public and can be common to a group of users. A 160-bit prime number qis chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides (p 1). Finally, g is chosen to be of the form $h^{(p1)/q}$ mod p where h is an integer between 1 and (p 1) with the restriction that g must be greater than 1.[5]

[5] In number-theoretic terms, g is of order q mod p; see Chapter 8.

[Page 392]

Figure 13.2. The Digital Signature Algorithm (DSA)

(This item is displayed on page 391 in the print version)

Global Public-Key Components		
p	prime number where $2^{L-1} < p < 2^L$ for 512 ≤ L 1024 and	prime divisor of (p1), where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits
q		
g	= $h^{(p-1)/q}$ mod p, where h is any integer with 1 < h < (p 1) such that $h^{(p-1)/q}$ mod p > 1	
User's Private Key		
x	random or pseudorandom integer with 0 < x < q	
User's Public Key		
y	= g^x mod p	
User's Per-Message Secret Number		
k	= random or pseudorandom integer with 0 < k < q	
Signing		
r	= (g^k mod p) mod q	
s	= [k^{-1} (H(M) + xr)] mod q	
Signature = (r, s)		
Verifying		
w	= $(s')^{-1}$ mod q	
u1	= [H(M')w] mod q	
u2	=(r')w mod q	
v	= [(g^{u1} y^{u2}) mod p] mod q	
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	

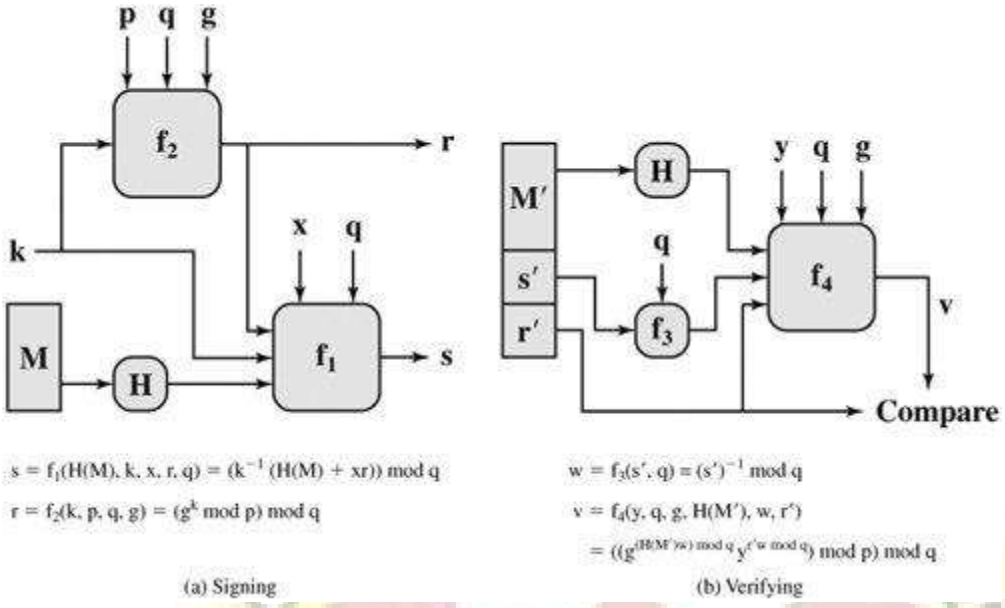
With these numbers in hand, each user selects a private key and generates a public key. The private key x must be a number from 1 to (q 1) and should be chosen randomly or pseudorandomly. The public key is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key y, it is believed to be computationally infeasible to determine x, which is the discrete logarithm of y to the base g, mod p (see Chapter 8).

To create a signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and an additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

At the receiving end, verification is performed using the formulas shown in Figure 13.2. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and the hash code of the incoming message. If this quantity matches the r component of the signature, then the signature is validated.

Figure 13.3 depicts the functions of signing and verifying.

Figure 13.3. DSS Signing and Verifying



(a) Signing (b) Verifying

The structure of the algorithm, as revealed in Figure 13.3, is quite interesting. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global public-key components. The multiplicative inverse of k (mod q) is passedto a function that also has as inputs the message hash code and the user's private key. The structure of this function is such that the receiver can recover r using the incoming message and signature, the public key of the user, and the global public key. It is certainly not obvious from Figure 13.2 or Figure 13.3 that such a scheme would work. A proof is provided at this book's Web site.

[Page 393]

Given the difficulty of taking discrete logarithms, it is infeasible for an opponent to recover k from r or to recover x from s.

Another point worth noting is that the only computationally demanding task in signature generation is the exponential calculation g^k mod p. Because this value does not depend on the message to be signed, it can be computed ahead of time. Indeed, a user could precalculate a number of values of r to be used to sign documents as needed. The only other somewhat demanding task is the determination of a multiplicative inverse, K^1. Again, a number of these values can be precalculated.

*****