Unit-2

1. Caesar Cipher:

- The encryption rule is simple; replace each letter of the alphabet with the letter standing 3 places further down the alphabet.
- The alphabet is wrapped around so that Z follows A.
- Generally Plain text is in lower case and Cipher text is Upper Case.

The following table is used in the Caesar Cipher encryption:

a	b	c	d	e	f	g	h	i	j
0	1	2	3	4	5	6	7	8	9

k	l	m	n	o	p	q	r	s	t
10	11	12	13	14	15	16	17	18	19

u	v	w	x	y	z
20	21	22	23	24	25

- Mathematically, starting from a=0, b=1, c=2, d=3, and so on, Caesar cipher can be written as:
  E(P)=(P+K)mod(26) (i.e., for encryption)
  D(C)=(C-K)mod(26) (i.e., for decryption),
  Where P= plain text, K=security key, C=cypher text

Ex: Plain text: "meet" and security key: K=3
Sol: First we find the cipher text of the word "meet" by using the key K=3.
Let the encryption be E(P)=(P+K)mod(26)
i.e., E(P)=(P+3)mod(26)

Serial number of the letter in the word "meet"	Encryption
1	M=E(M)=(12+3)mod(26)=15=p
2	E=E(E)=(4+3)mod(26)=7=H
3	E=E(E)=(4+3)mod(26)=7=H
4	T=E(T)=(19+3)mod(26)=21=V

Therefore, the ciphertext of the word "meet" is "PHHV"
Now, we find the plaintext of the ciphertext of the word "PHHV"by using the key K=3.
Let the decryption be D(P)=(C-K)mod(26)
i.e., D(P)=(C-3)mod(26)

Serial number of the letter in the word "meet"	Decryption
1	D(p)=(15-3)mod(26)=12=m
2	D(H)=(7-3)mod(26)=4=e
3	D(H)=(7-3)mod(26)=4=e
4	D(V)=(21-3)mod(26)=19=t

Ex: Find ciphertext for the plain text: "meet" and use security key: K=4, and hence by using decryption obtain plaintext for the obtained ciphertext.

Ex: Find ciphertext for the plain text: "meet" and use security key: K=5, and hence by using decryption obtain plaintext for the obtained ciphertext.

Ex: Find ciphertext for the plain text: "meet" and use security key: K=6, and hence by using decryption obtain plaintext for the obtained ciphertext.

Ex: Find ciphertext for the plain text: "hello" and use security key: K=4, and hence by using decryption obtain plaintext for the obtained ciphertext.

Ex: Plaintext: meet me after the party, use security key K=3
Ciphertext: PHHW PH DIWHU WKH SDUWB.

- This cipher can be broken
  - If we know one plaintext-cipher text pair since the difference will be same.
  - By applying Brute Force attack as there are only 26 possible keys.

2. **Monoalphabetic Substitution Cipher**

Instead of shifting alphabets by fixed amount as in Caesar cipher, any random permutation is assigned to the alphabets. This type of encryption is called monoalphabetic substitution cipher.

For example let us take following keys:

Key number 1:

A	b	c	d	e	f	g	h	i	j
D	C	A	B	H	G	F	E	M	K

K	l	m	n	o	p	q	r	s	t
I	N	J	L	S	Q	R	O	P	W

U	v	w	x	y	z

X	U	Z	Y	T	V

Or
Key number 2:

A	b	c	d	e	f	g	h	i	j
J	I	H	G	F	E	D	C	B	A

K	l	m	n	o	p	q	r	s	T
T	S	R	Q	P	O	N	M	L	K

U	v	w	x	y	z
Z	Y	X	W	V	U

Or
Key number 3:

A	b	c	d	e	f	g	h	i	j
T	U	S	R	V	Q	P	W	O	N

K	L	m	n	o	p	q	r	s	t
X	M	L	Y	K	J	I	Z	H	G

U	v	w	x	y	Z
F	E	D	C	B	A

And so on (i.e. if we continue to write all possible keys we get total 26! Keys). Therefore, it will be comparatively stronger than Caesar cipher. Thus, Brute Force attack is impractical in this case.

However, another attack is possible. Human languages are redundant i.e. certain characters are used more frequently than others. This fact can be exploited.

- In English ‘e’ is the most common letter followed by ‘t’, ‘r’, ‘n’, ’o’, ‘a’ etc. Letters like ‘q’, ‘x’, ‘j’ are less frequently used.
- Moreover, digrams like ‘th’ and trigrams like ‘the’ are also more frequent.
- Tables of frequency of these letters exist. These can be used to guess the plaintext if the plaintext is in uncompressed English language.
- The most common two letter combinations are called as **digrams**. e.g. th, in, er, re and an.
- The most common three letter combinations are called as **trigrams.** e.g. the, ing, and, and ion.

Ex: Write ciphertext for the word “**meet**” by using key number 1 (defined above).

Ans: Ciphertext: **JHHW**

Ex: Write plaintext for the ciphertext "**JHHW**" by using key number 1 (defined above).

Ans: Ciphertext: **meet**

3. Playfair Cipher

- In this technique multiple (2) letters are encrypted at a time.
- This technique uses a 5 X 5 matrix which is also called key matrix.
  For example:
  1. By using the word "KRISHNA", we define a key (i.e., a 5 X 5 matrix)

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

2. By using the word "ABHI", we define a key (i.e., a 5 X 5 matrix)

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

3. By using the word "MONARCHY", we define a key (i.e., a 5 X 5 matrix)

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

The plaintext is encrypted **two letters at a time**:

- Break the plaintext into pairs of two consecutive letters.
- If a pair is a repeated letter, insert a filler like 'X' in the plaintext, eg. "Balloon" is treated as "ba lx lo on".
- If both letters fall in the same row of the key matrix, replace each with the letter to its right (wrapping back to start from end), eg. "AR" encrypts as "RM" (by using "MONARCHY" key).
- If both letters fall in the same column, replace each with the letter below it (again wrapping to top from bottom), eg. "MU" encrypts to "CM" (by using "MONARCHY" key).

- Otherwise each letter is replaced by the one in its row in the column of the other letter of the pair, eg. “HS" encrypts to "BP", and “EA" to "IM" or "JM" (by using “MONARCHY” key).
- If the letter is standing alone in the process of pairing then add “Z” with the letter.

Strength of Playfair cipher is a great advance over simple mono alphabetic ciphers. Since there are 26 letters, 26x26 = 676 diagrams are possible, so identification of individual diagram is more difficult.

However, it can be broken even if a few hundred letters are known as much of plaintext structure is retained in cipher text.

Example: **PlainText**: "instruments", use keyword: monarchy
After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'
cipher text : ga tl mz cl rq tx

Hill Cipher:

The **Hill cipher** is a polygraphic substitution cipher built on concepts from Linear Algebra. The Hill cipher makes use of modulo arithmetic, matrix multiplication, and matrix inverses; hence, it is a more mathematical cipher than others. The Hill cipher is also a block cipher, so, theoretically, it can work on arbitrary sized blocks. Hill Cipher in cryptography was invented and developed in 1929 by Lester S. Hill, a renowned American mathematician.

- Polygraphic substitution is a uniform substitution where a block of letters is substituted by a word, character, number, etc.

Generally, the below-mentioned structure of numbers and letters are used in the Hill Cipher Encryption, but this can be modified as per requirement.

Letter	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
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

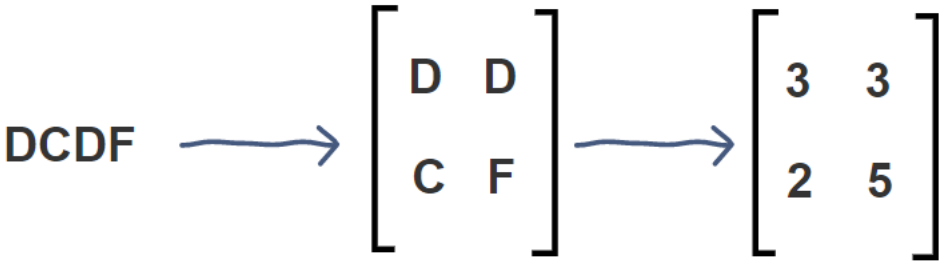
Encryption

Encrypting with the Hill cipher is built on the following operation:

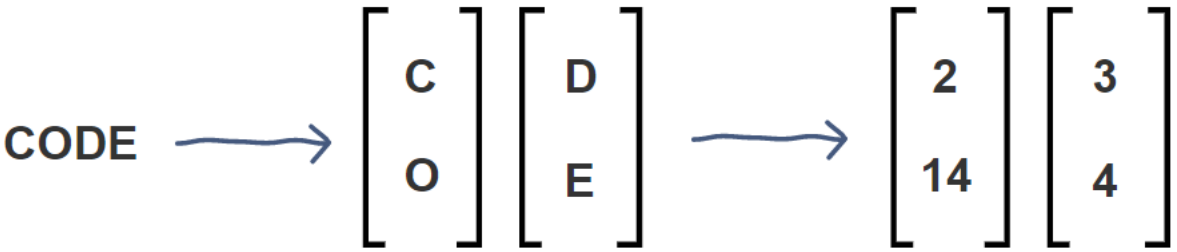
$$E(K, P) = (K*P) \bmod 26$$

Where K is our key matrix and P is the plaintext in vector form. Matrix multiplying these two terms produces the encrypted ciphertext. Let's do so step by step:

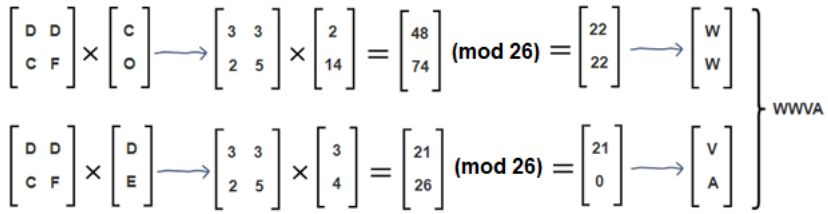
1. Pick a keyword to encrypt your plaintext message. Let's work with the random keyword "DCDF". Convert this keyword to matrix form using your substitution scheme to convert it to a numerical 2x2 key matrix.



2. Next, we will convert our plaintext message to vector form. Since our key matrix is 2x2, the vector needs to be 2x1 for matrix multiplication to be possible. In our case, our message is four letters long so we can split it into blocks of two and then substitute to get our plaintext vectors.



3. Now, we can matrix multiply the key matrix with each 2x1 plaintext vector, take the moduli of the resulting 2x1 vectors by 26, and concatenate the results to get "WWVA", the final ciphertext.



Decryption

Decrypting with the Hill cipher is built on the following operation:

$$D(K, C) = (K^{-1} * C) \bmod 26$$

Where K is our key matrix and C is the ciphertext in vector form. Matrix multiplying the inverse of the key matrix with the ciphertext produces the decrypted plaintext. Let's do this step by step with our ciphertext, "WWVA":

1. First, we calculate the inverse of the key matrix. In doing so, we must keep the result between 0-25 using modulo 26. For this reason, the Extended Euclidean algorithm is used to find the modular multiplicative inverse of the key matrix determinant.

$$K^{-1} \pmod{26} = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}^{-1} \pmod{26} = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

2. Next, we will multiply 2x1 blocks of the ciphertext with the inverse of the key matrix to get our original plaintext message, "CODE," back.

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 22 \\ 22 \end{bmatrix} = \begin{bmatrix} 704 \\ 638 \end{bmatrix} \pmod{26} = \begin{bmatrix} 2 \\ 14 \end{bmatrix} \longrightarrow \begin{bmatrix} C \\ O \end{bmatrix}$$

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 21 \\ 0 \end{bmatrix} = \begin{bmatrix} 315 \\ 420 \end{bmatrix} \pmod{26} = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} D \\ E \end{bmatrix}$$

CODE

Polyalphabetic ciphers

- In a polyalphabetic cipher, multiple "alphabets" are used to encipher.
- If two letters are the same in the ciphertext it does not mean they must decipher to the same plaintext letter.

  Polyalphabetic ciphers are 1. Vigenere Cipher, 2.Vernam Cipher

The Vigenère cipher :

- This is a type of polyalphabetic substitution cipher (includes multiple substitutions depending on the key). In this type of cipher, the key determines which particular substitution is to be used.
- To encrypt a message, a key is needed that is as long as the message. Usually, the key is a repeating keyword.

  For example, if the keyword is *deceptive*, the message "we are discovered save yourself" is encrypted as follows:

  Key: *deceptivedecept*

  Plaintext: wearediscovered

  Ciphertext: ZICVTWQNGRZGVTW
- Encryption can be done by looking in the Vigenere Table where ciphertext is the letter key's row and plaintext's column.

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

- Decryption is equally simple. The key letter again identifies the row. The position of the cipher text letter in that row determines the column, and the plaintext letter is at the top of that column.
- The strength of this cipher is that there are multiple ciphertext letters for each plaintext letter, one for each unique letter of the keyword.

Example: Plaintext: MICHIGAN TECHNOLOGICAL UNIVERSITY
Keyword: HOUGHTON

Sol: Plaintext: MICHIGAN TECHNOLOGICAL UNIVERSITY
Key HOUGH TONHO UGHTO NHOUG HTONH OUGHT O
Ciphertext: TWWNP ZOAAS WNUHZ BNWWG SNBVC SLYPM M

Vernam Cipher :

The Vernam Cipher is an algorithm invented in 1917 to encrypt teletype (TTY) messages by Gilbert Sandford Vernam, it is a symmetric cipher.
The technique can be expressed as follows:
For encryption: $C = P + K$
Where P = plaintext, K = key, C = ciphertext.
For decryption $P = C$ – K

(or)

- The technique can also be expressed as follows:

Ci = Pi $\oplus$ Ki
Where Pi = ith binary digit of plaintext.
Ki = ith binary digit of key.
Ci = ith binary digit of ciphertext.
$\oplus$ = exclusive-or (XOR) operation

- Thus, the ciphertext is generated by performing the bitwise XOR of the plaintext and the key.
- Decryption simply involves the same bitwise operation:
  Pi = Ci $\oplus$ Ki

One-Time Pad

- In this scheme, a random key that is as long as the message is used.
- The key is used to encrypt and decrypt a single message, and then is discarded. Each new message requires a new key of the same length as the new message.
- This scheme is unbreakable.
- It produces random output that bears no statistical relationship to the plaintext.
- Because the ciphertext contains no information whatsoever about the plaintext, there is simply no way to break the code.
- For any plaintext of equal length to the ciphertext, there is a key that produces that plaintext.
- Therefore, if you did an exhaustive search of all possible keys, you would end up with many legible plaintexts, with no way of knowing which the intended plaintext was.
- Therefore, the code is unbreakable
- The security of the one-time pad is entirely due to the randomness of the key.
- The one-time pad offers complete security but, in practice, has two fundamental difficulties:
  - There is the practical problem of making large quantities of random keys. Any heavily used system might require millions of random characters on a regular basis. Supplying truly random characters in this volume is a significant task.
  - Another problem is that of key distribution and protection. For every message to be sent, a key of equal length is needed by both sender and receiver.
- Because of these difficulties, the one-time pad is used where very high security is required.
- The one-time pad is the only cryptosystem that exhibits **perfect secrecy .**

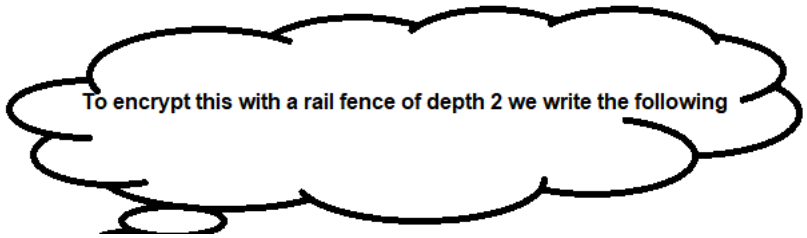
Transposition Techniques

- No replacement of character.
- We will rearrange the character's position i.e., we will apply some sort of permutation on the plain text letters. (i.e., it reorders the symbols. i.e., that is rearrangement of the letters of the plain text).
- The word "NAME" CAN BE ARRANGED IN 4! WAYS
  That is, ANEM, AMEN, NEAM and so on (these are ciphertexts).

Rail Fence Technique:

In this, the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

Ex: Plaintext: All the best for your exams



Encrypt: A L H B S F R X M
L T E E T O E A S

Ciphertext: ALHBSFRXMLTEETOEAS

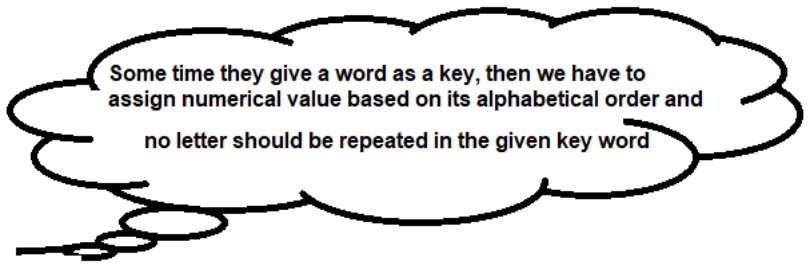
Note:

- Used for short messages
- Easy to break by the attackers

Row transposition cipher (or) Columnar transposition:

We write the message in a rectangle, row by row and read the message off, column by column, but permute the order of column.

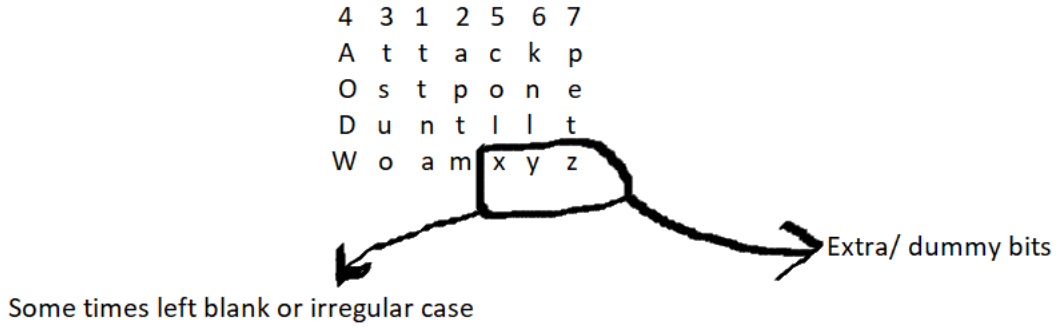
Key: 45312, 4321 etc.



EXAMPLE: 3 1 4 2 5 and 1 4 6 3 5 2
M A N G O C R Y P T O

Example:
Plaintext: Attack postponed until two am, Key: 4312567

Sol:



Ciphertext: TTNA APTM TSUO AODW COLX PETZ

Explanation:

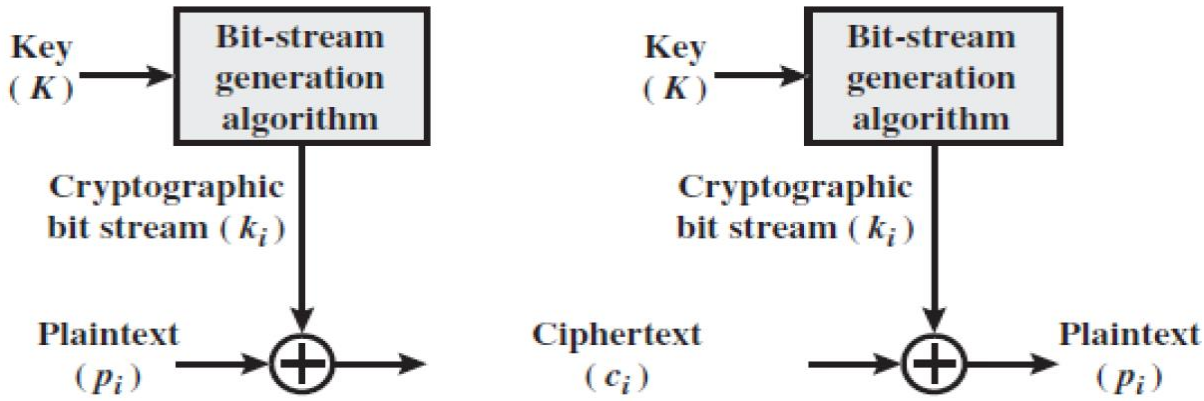
- To encrypt, start with the column label as 1 i.e., in our example, column three is labelled as 1, write down all the letters of the column.
- Now, proceed column four (i.e., assign number 2 for column four) then 2, 1, 5, 6 and 7^{th} column.

Unit-3

Block Ciphers

- Basically cryptographic algorithm is used for transformation of plain text into cipher text.
- There are basically two method on which cryptographic algorithm is work:
  1. Stream Cipher 2. Block Cipher

1. Stream Cipher: A stream cipher is an encryption technique that works byte by byte (or bit by bit or character by character) to transform plain text into code that's unreadable to anyone without the proper key. Stream cipher is a symmetric cipher model.
Example of classical stream cipher is the One time pad cipher (i.e., Vernam cipher).



(a) Stream cipher using algorithmic bit-stream generator

p	q	$p \oplus q$
1	1	0
1	0	1
0	1	1

0	0	0

E

xclusive-or Logical operator: Exclusive-or (XOR) is a logical operator which results true when either of the operands are true (one is true and the other one is false) but both are not true and both are not false. The Exclusive-or truth table is as follows:

- If P=Plaintext, C=Cipher text and K=Key, then

  To encrypt: $C = P \oplus K$

  To decrypt: $P = C \oplus K$

  Example: P= 1 0 0 1 1 1 0 1

  K= 0 0 1 1 1 0 1 0

  For encryption C= 1 0 1 0 0 1 1 1

  For Decryption: C= 1 0 1 0 0 1 1 1

  K= 0 0 1 1 1 0 1 0

  P= 1 0 0 1 1 1 0 1

  Advantages:

- High speed transformation
- Low error propagation

  Disadvantages:

- Low diffusion

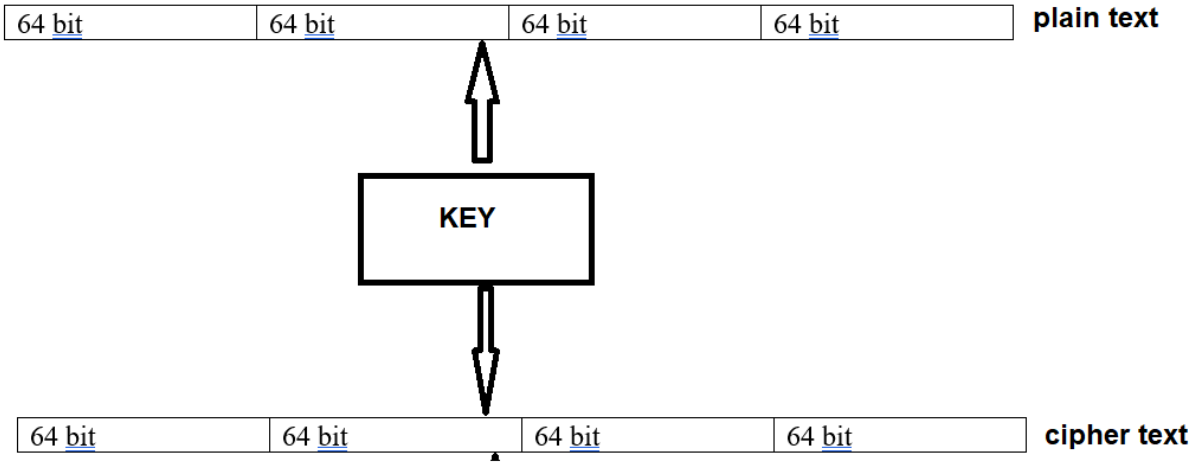
  (Diffusion means if a single symbol in the plaintext is changed, several or all symbol in the ciphertext will also be changed.

  Confusion means If a single bit in the key is changed, most or all bits in the ciphertext will also be changed. More details about Diffusion and Confusion are provided at the end of this notes)
- Less secure

2. Block Cipher: In Block Cipher, plain text message divide into fixed size blocks and encrypt each block with some fixed size of key. Divide each plain text message into block size of 64, 128, 256 bits and apply key size of 40, 56, 64, 80, 128, 168, 192 and 256 bits which generate cipher text block same as size of plain text block. It is also a symmetric cipher. A receiver side decrypt message with same key to generate plain text.

- Block cipher are used chaining mode, this is because for repeating text pattern, the same cipher block will be generated which can give clue to cryptanalyst regarding what is the original plain text.

  For example: let us assume that the given plain text is divided into four blocks of containing 64 bit each. Given key will be applied for each block.



Advantages:

- High diffusion
- More secure

Disadvantages:

- Encryption process is slow
- Error propagation

Difference between Block and Stream Ciphers

Stream Cipher	Block Cipher
1. Key and algorithm applied on each binary bit. 2. Less time consuming in compared to block cipher. 3. Only one bit encrypted at a time, it is faster than block cipher. 4. It does not use chaining mode 5. Hardware implementation is easy to use stream cipher. 6. One time pad is best example for Stream cipher. 7. Less secure in compared to block cipher.	1. Key and algorithm applied on block of data. 2. More time consuming in compared to stream cipher. 3. Block of data is encrypted at a time; it is slower than stream cipher. 4. It uses chaining mode. 5. Software implementation is easy to use block cipher. 6. DES is the best example for block cipher. 7. More secure in compared to stream cipher.

Confusion and Diffusion

- The terms Confusion and Diffusion were introduced by Claude Shannon. Shannon's concern was to prevent cryptanalysis, based on statistical analysis, The reason is as follows:

  Assume attacker has some knowledge of the statistical characteristics of the plain text (For example, in a message, the frequency distribution of the various letters may be known. If these statistics are in any way reflected in the ciphertext, the cryptanalyst i.e. the attacker may be able to deduce the encryption key.)

  Thus, Shannon suggested 2 methods for frustrating the attackers:

  1. Confusion
  2. Diffusion

  } **properties for creating a secure cipher**

  Diffusion: If a single symbol in the plaintext is changed, several or all symbol in the ciphertext will also be changed.

  The idea of diffusion is to hides the relationship between the **ciphertext** and **plaintext.**

  Diffusion implies that each symbol in the ciphertext is dependent on some or all the symbols in the plaintext.

  Confusion: If a single bit in the key is changed, most or all bits in the ciphertext will also be changed.

Feistel cipher: Named after the IBM cryptographer Horst Feistel and first implemented in the Lucifer cipher by Horst Feistel and Don Coppersmith.

The process of encryption **Feistel Cipher** takes place as follows,

1. In this Cipher, the plain text is divided into two equal parts. The left part is denoted as *L* and the Right part is denoted as *R*.
2. Every round has an encryption function that is applied to the plain text. (It is applied only to one of the two divisions of the plain text, that is to the left one.)
3. The encryption function is applied on the left part of the plain text and the right part goes unchanged in every round.
4. The encryption function has two parameters: Encryption key and Right part of the plain text (i.e., $f(RE_{i-1}, K_i), i = 1, 2, \ldots, 16$).
5. XOR operation is performed between the Left part and the encryption function.
6. The Right part becomes the Left part of the next round and the output of the XOR operation becomes the Right part of the next round. It means that the substituted right part and unchanged right part are swapped for the next round.
7. Each round has a different encryption key, or we can say that the key is round dependent, i.e. the key for every round is generated in advance.

The process round-I is shown below:

$LE_0 = RE_1$,
$RE_1 = LE_0 \oplus f(RE_0, K_1)$

Where f is a Feistel function, $K_1 =$ Round-I key, $E_0 =$ Intial round encrytion , and $E_1 =$ Round-I encrytion .

8. The process shown above is of a single round. The number of rounds depends upon the algorithm of the process.

The difficult part of this algorithm is designing the round function because it must be applied in every round until the final ciphertext is received. The more the number of rounds, the more secure the data becomes.

The decryption process of Feistel Cipher is given below:

The decryption process of **Feistel Cipher** is almost the same as the encryption process. Just like we entered the plain text in the Feistel block, we have to do the same with the ciphertext. The ciphertext will be divided into two parts just like the plain text. The only difference is that the keys will be used in reverse order.

The output of the first round of the decryption is as follows:

- By last round of the encryption, we have $LE_{16} = RE_{15}$, and $RE_{16} = LE_{15} \oplus f(RE_{15}, K_{16})$
- Where f is a Feistel function, $K_{16} =$ Round-16 key, $E_{15} =$ Round-15 encrytion , and $E_{16} =$ Round-16 encrytion .
- On the decryption side $LD_1 = RD_0 = LE_{16} = RE_{15}$, where $D_0 =$ Intial round decrytion , $D_1 =$ Round-I decrytion and $RD_1 = LD_0 \oplus f(RD_0, K_{16})$ (see the block diagram of Feistel cipher)
  $= RE_{16} \oplus f(RE_{15}, K_{16})$, where $RE_{16} = LD_0$
  $= [LE_{15} \oplus f(RE_{15}, K_{16})] \oplus f(RE_{15}, K_{16})$, where $RE_{16} = LE_{15} \oplus f(RE_{15}, K_{16})$
  $= LE_{15} \oplus [f(RE_{15}, K_{16}) \oplus f(RE_{15}, K_{16})]$, since $[A \oplus B] \oplus C = A \oplus [B \oplus C]$
  $= LE_{15} \oplus [0]$, Since $A \oplus A = 0$
  $= LE_{15}$, Since $A \oplus 0 = A$

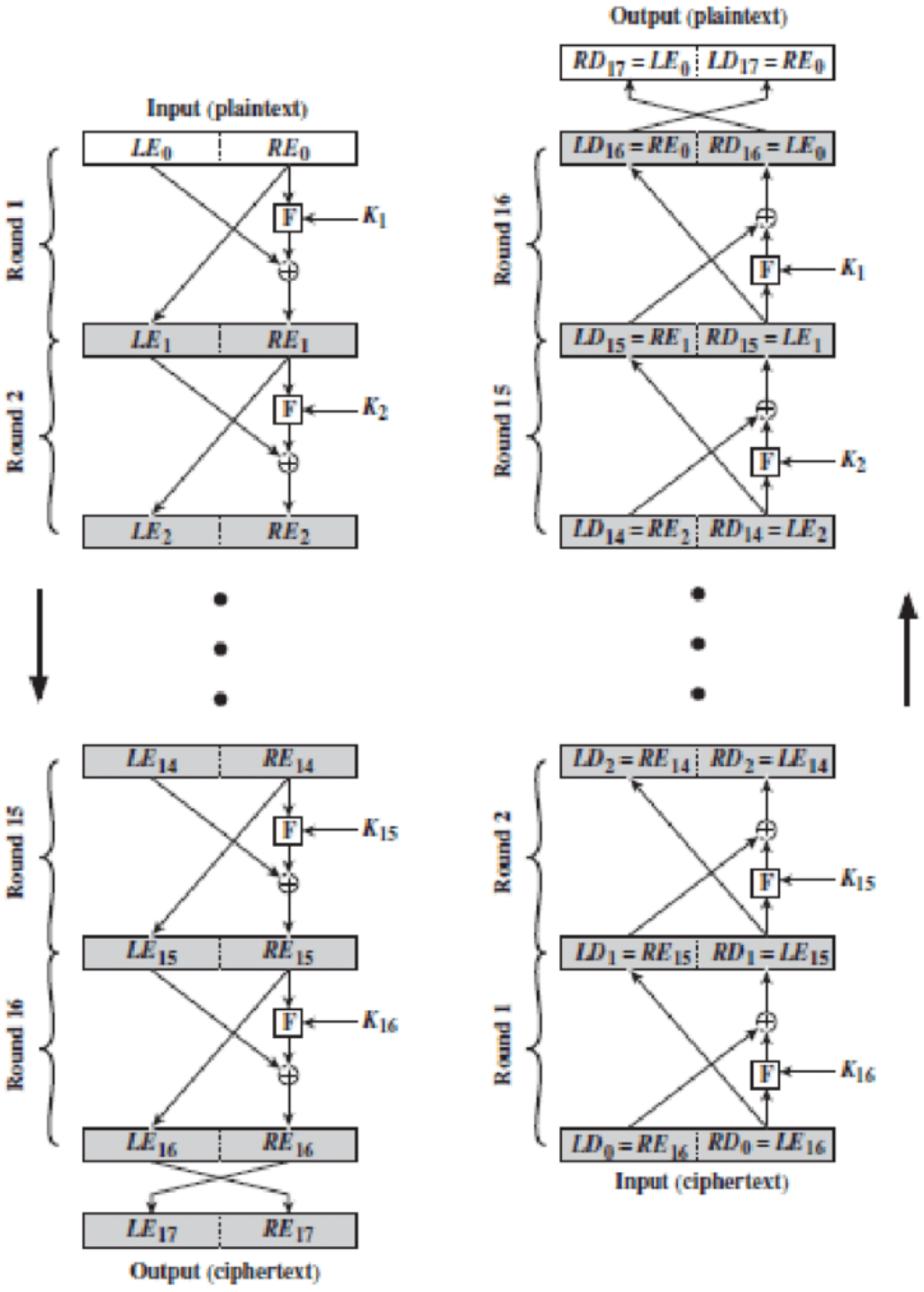
Thus, we have $LD_1 = RE_{15}$ and $RD_1 = LE_{15}$

Number of rounds:

The number of rounds depends upon how much security you want. Security is directly proportional to the number of rounds. But simultaneously it slows down the speed of

encryption and decryption. The larger the number of rounds is, the creation of ciphertext from plain text and plain text from ciphertext will be slow.

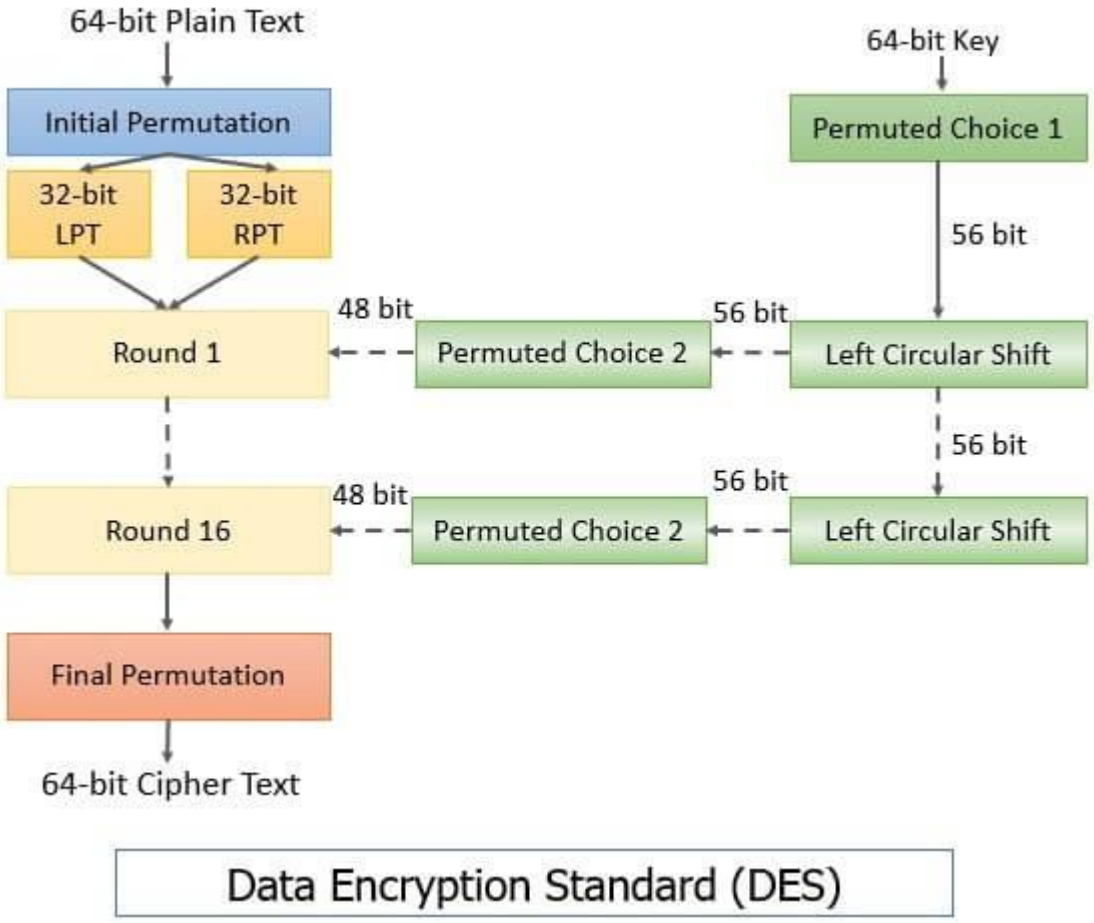
The block diagram of the Feistel cipher is as follows:



DES: THE DATA ENCRYPTION STANDARD

The 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 :



Data Encryption Standard (DES)

The encryption process in DES is made of two permutations (P-boxes), which we call initial and final permutations, and sixteen Feistel rounds. Each round uses a different 48-bit round key (Actually, we have 64 bit key which go as a input to permuted choice-I and we get out as 56 bit key, finally this 56 bit key then converted into 48 bit key by using permuted choice-II) generated from the cipher key according to a predefined algorithm.

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 Permutation(IP) and Final Permutation/Inverse Initial Permutation (IP^{-1}):

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

Initial Permutation(IP) table:

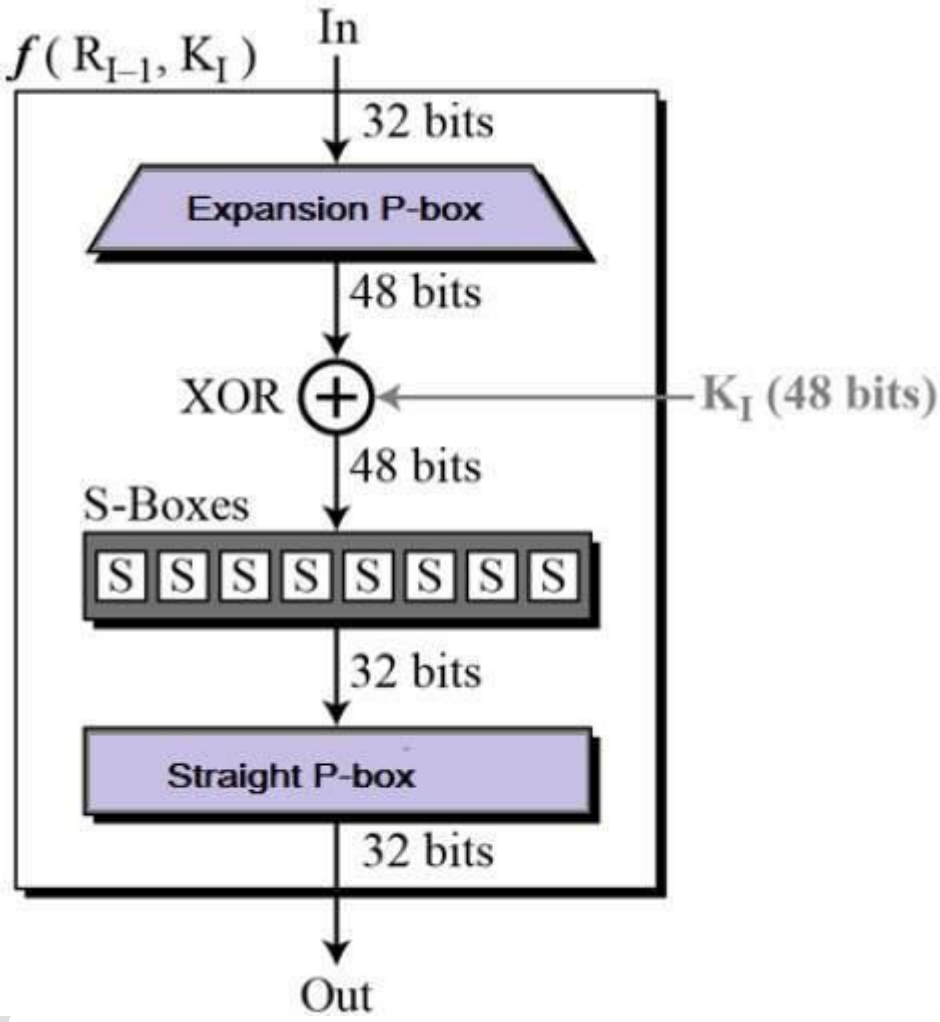
58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Inverse Initial Permutation (IP^{-1}):

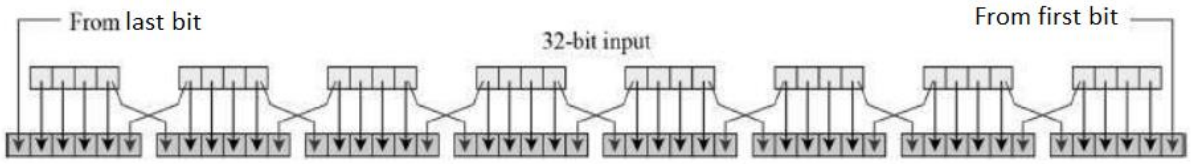
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

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. The single round function of DES (i.e., Round-I) is as follows:



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 –

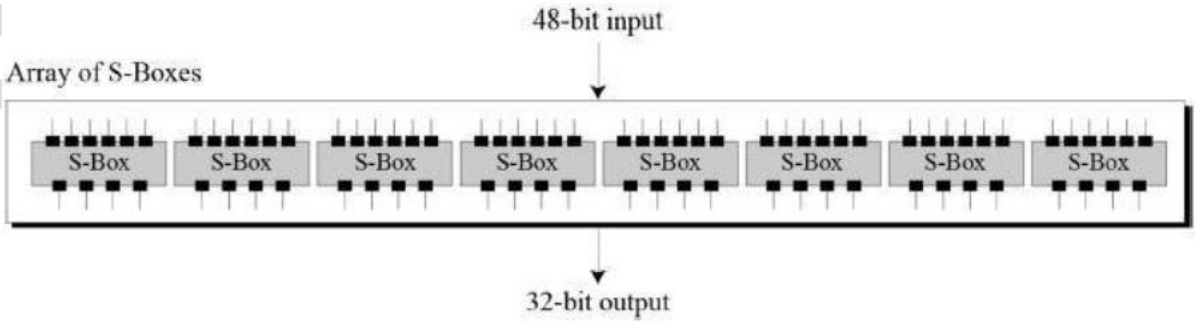


The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown

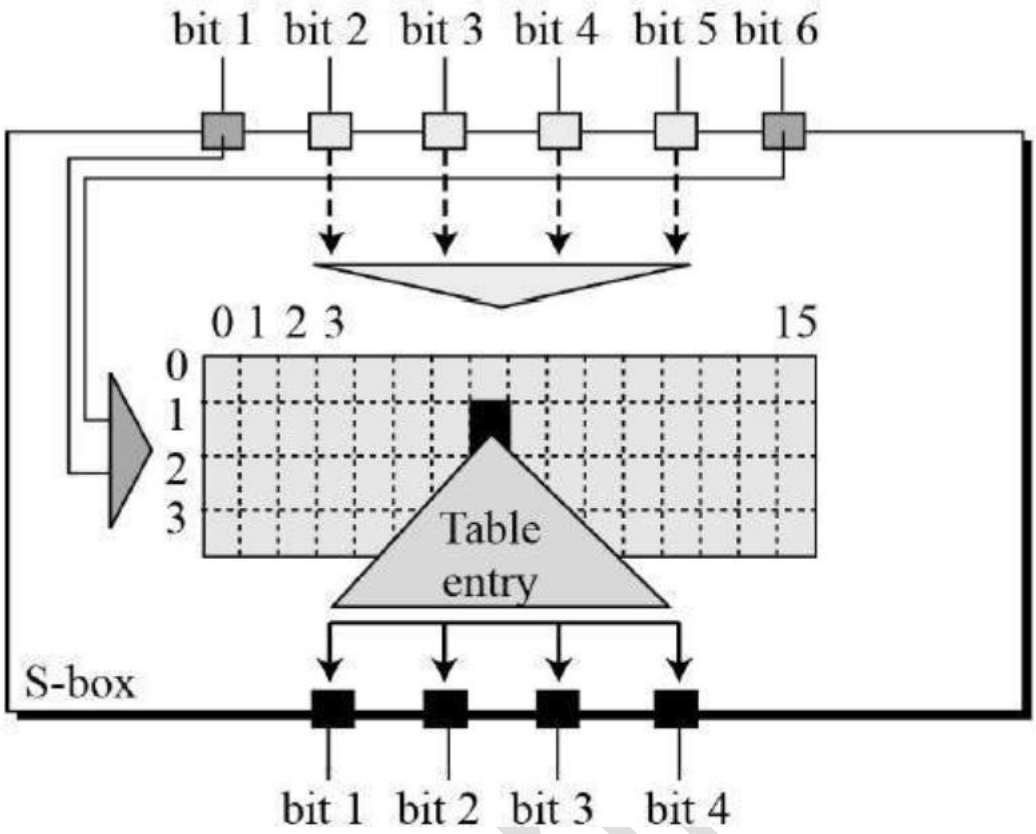
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

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.
- That is, each S-box has its own table. The values of the inputs (row number and column number) and the values of the outputs are given as decimal numbers to save space. These need to be changed to binary

S-box 1

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	14	04	13	01	02	15	11	08	03	10	06	12	05	09	00	07
1	00	15	07	04	14	02	13	10	03	06	12	11	09	05	03	08
2	04	01	14	08	13	06	02	11	15	12	09	07	03	10	05	00
3	15	12	08	02	04	09	01	07	05	11	03	14	10	00	06	13

S-box 2

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	15	01	08	14	06	11	03	04	09	07	02	13	12	00	05	10
1	03	13	04	07	15	02	08	14	12	00	01	10	06	09	11	05
2	00	14	07	11	10	04	13	01	05	08	12	06	09	03	02	15
3	13	08	10	01	03	15	04	02	11	06	07	12	00	05	14	09

S-box 3

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	10	00	09	14	06	03	15	05	01	13	12	07	11	04	02	08
1	13	07	00	09	03	04	06	10	02	08	05	14	12	11	15	01
2	13	06	04	09	08	15	03	00	11	01	02	12	05	10	14	07
3	01	10	13	00	06	09	08	07	04	15	14	03	11	05	02	12

S-box 4

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	07	13	14	03	00	6	09	10	1	02	08	05	11	12	04	15
1	13	08	11	05	06	15	00	03	04	07	02	12	01	10	14	09
2	10	06	09	00	12	11	07	13	15	01	03	14	05	02	08	04
3	03	15	00	06	10	01	13	08	09	04	05	11	12	07	02	14

S-box 5

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	02	12	04	01	07	10	11	06	08	05	03	15	13	00	14	09
1	14	11	02	12	04	07	13	01	05	00	15	10	03	09	08	06
2	04	02	01	11	10	13	07	08	15	09	12	05	06	03	00	14
3	11	08	12	07	01	14	02	13	06	15	00	09	10	04	05	03

S-box 6

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	12	01	10	15	09	02	06	08	00	13	03	04	14	07	05	11
1	10	15	04	02	07	12	09	05	06	01	13	14	00	11	03	08
2	09	14	15	05	02	08	12	03	07	00	04	10	01	13	11	06
3	04	03	02	12	09	05	15	10	11	14	01	07	10	00	08	13

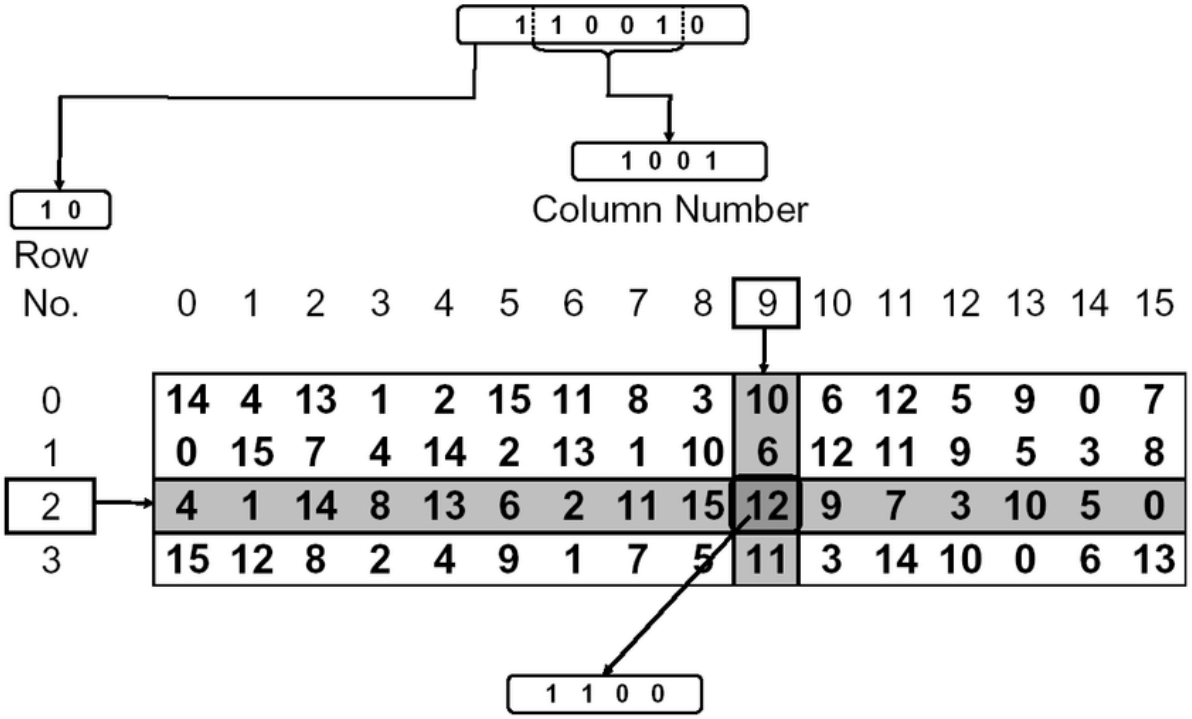
S-box 7

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	4	11	2	14	15	00	08	13	03	12	09	07	05	10	06	01
1	13	00	11	07	04	09	01	10	14	03	05	12	02	15	08	06
2	01	04	11	13	12	03	07	14	10	15	06	08	00	05	09	02
3	06	11	13	08	01	04	10	07	09	05	00	15	14	02	03	12

S-box 8

	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
0	13	02	08	04	06	15	11	01	10	09	03	14	05	00	12	07
1	01	15	13	08	10	03	07	04	12	05	06	11	10	14	09	02
2	07	11	04	01	09	12	14	02	00	06	10	10	15	03	05	08
3	02	01	14	07	04	10	8	13	15	12	09	09	03	05	06	11

For example:



Example

The input to S-box 1 is **1**00001**1**. What is the output?

Solution

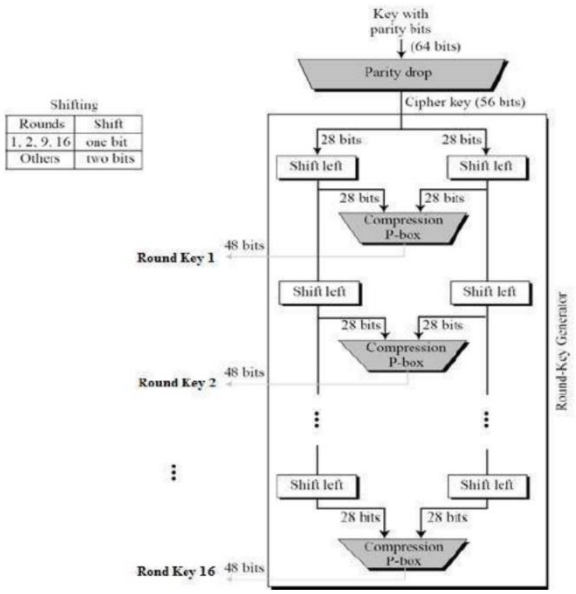
If we write the first and the sixth bits together, we get 11 in binary, which is 3 in decimal. The remaining bits are 0001 in binary, which is 1 in decimal. We look for the value in row 3, column 1, in Table 6.3 (S-box 1). The result is 12 in decimal, which in binary is 1100. So the input 100011 yields the output 1100.

Straight Permutation (Permutation function) − The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. However, the cipher key is normally given as a 64-bit key in which 8 extra bits are the parity bits, which are dropped before the actual key-generation process, as shown in the following figure:



Parity Drop (Permuted Choice-I):

The pre-process before key expansion is a compression permutation that we call parity bit drop. It drops the parity bits (bits 8, 16, 24, 32, …, 64) from the 64-bit key and permutes the rest of the bits according to the following parity-bit drop table. The remaining 56-bit value is the actual cipher key which is used to generate round keys. The parity drop permutation (a compression P-box) is shown in the following parity-bit drop table.

Parity-bit drop table

57	49	41	33	25	17	09	01
58	50	42	34	26	18	10	02
59	51	43	35	27	19	11	03
60	52	44	36	63	55	47	39
31	23	15	07	62	54	46	38
30	22	14	06	61	53	45	37
29	21	13	05	28	20	12	04

Shift Left

After the straight permutation, the key is divided into two 28-bit parts. Each part is shifted left (circular shift) one or two bits. In rounds 1, 2, 9, and 16, shifting is one bit; in the other rounds, it is two bits. The two parts are then combined to form a 56-bit part. The following table shows the number of shifts for each round.

Number of bit shifts

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bit shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Compression Permutation (Permuted Choice-II):

The compression permutation (P-box) changes the 58 bits to 48 bits, which are used as a key for a round. The compression permutation is shown in the following table.

Key-compression table

14	17	11	24	01	05	03	28
15	06	21	10	23	19	12	04
26	08	16	07	27	20	13	02
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

DES ANALYSIS:

Critics have used a strong magnifier to analyse DES. Tests have been done to measure the strength of some desired properties in a block cipher. The elements of DES have gone

through scrutinies to see if they have met the established criteria. We discuss some of these in this section.

Properties:

Two desired properties of a block cipher are the avalanche effect and the completeness.

Avalanche Effect

Avalanche effect means a small change in the plaintext (or key) should create a significant change in the ciphertext. DES has been proved to be strong with regard to this property.

Example:

To check the avalanche effect in DES, let us encrypt two plaintext blocks (with the same key) that differ only in one bit and observe the differences in the number of bits in each round.

Plaintext: 0000000000000000 Key: 22234512987ABB23
Ciphertext: 4789FD476E82A5F1

Plaintext: 0000000000000001 Key: 22234512987ABB23
Ciphertext: 0A4ED5C15A63FEA3

Although the two plaintext blocks differ only in the rightmost bit, the ciphertext blocks differ in 29 bits. This means that changing approximately 1.5 percent of the plaintext creates a change of approximately 45 percent in the ciphertext.

Completeness effect

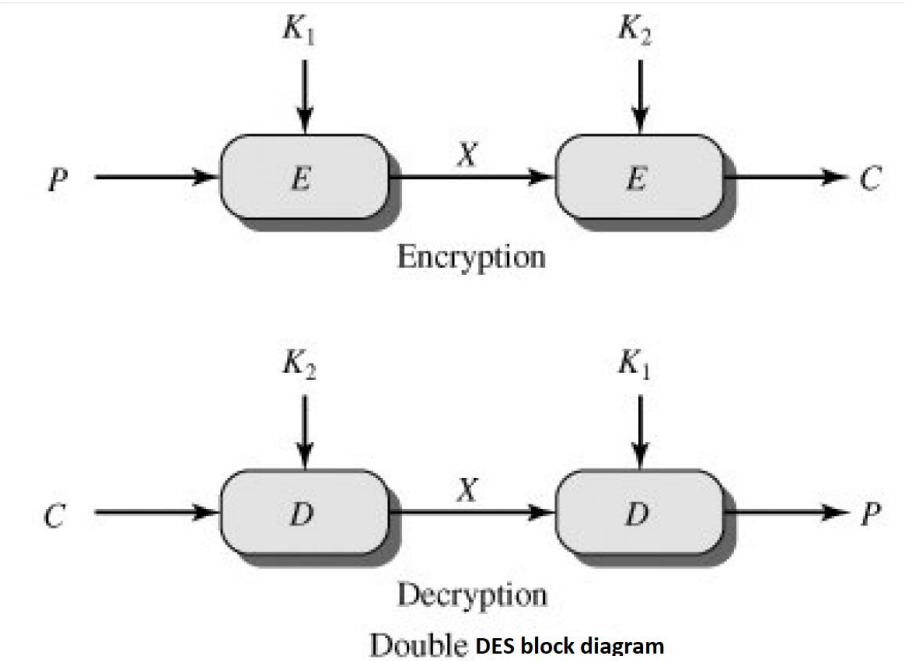
Completeness effect means that each bit of the ciphertext needs to depend on many bits on the plaintext. The diffusion and confusion produced by P-boxes and S-boxes in DES, show a very strong completeness effect.

MULTIPLE DES

Since DES attacks were vulnerable to brute-force attack, variations of DES called multiple DES were introduced.

Double DES

In this approach, we use two instances of DES ciphers for encryption and two instances of reverse ciphers for decryption. Each instance uses a different key, which means that the size of the key is now doubled (i.e., 56+56=112 bits). The simplest form of multiple encryptions has two encryption stages and two keys. Given a plaintext P and two encryption keys $K1$ and $K2$, ciphertext C is generated as $C = E(K_2, E(K_1, P))$, Where E means encryption. Similarly, Decryption requires that the keys be applied in reverse order i.e., $P = D(K_1, D(K_2, C))$, where D means decryption.



Double **DES block diagram**

However, double DES is vulnerable to a known-plain text attack, as discussed in the next definition.

Meet-in-the-Middle attack:

This attack involves encryption from one end and decryption from the other end and then "Matching the result in the middle" and hence the name.

This attack requires knowing same plaintext/ciphertext pairs. Let us assume plaintext=*P*, ciphertext=*C*. The attack process is as follows:

- Encrypt *P* for all 2^{56} possible values of K_1 and store the result in a table and sort it.
- Now, decrypt *C* using all 2^{56} possible values of K_2. As each decryption result is produced, check against the table for a match.
- When there is a match, we have located a possibly correct pair of keys.

Note: More than one pair of keys may result in a match, but these number of pairs will be small. We should try each possible pair of keys. So, it takes twice as long to break double DES using brute-force (because in double DES, we are using 2 keys).

Triple DES

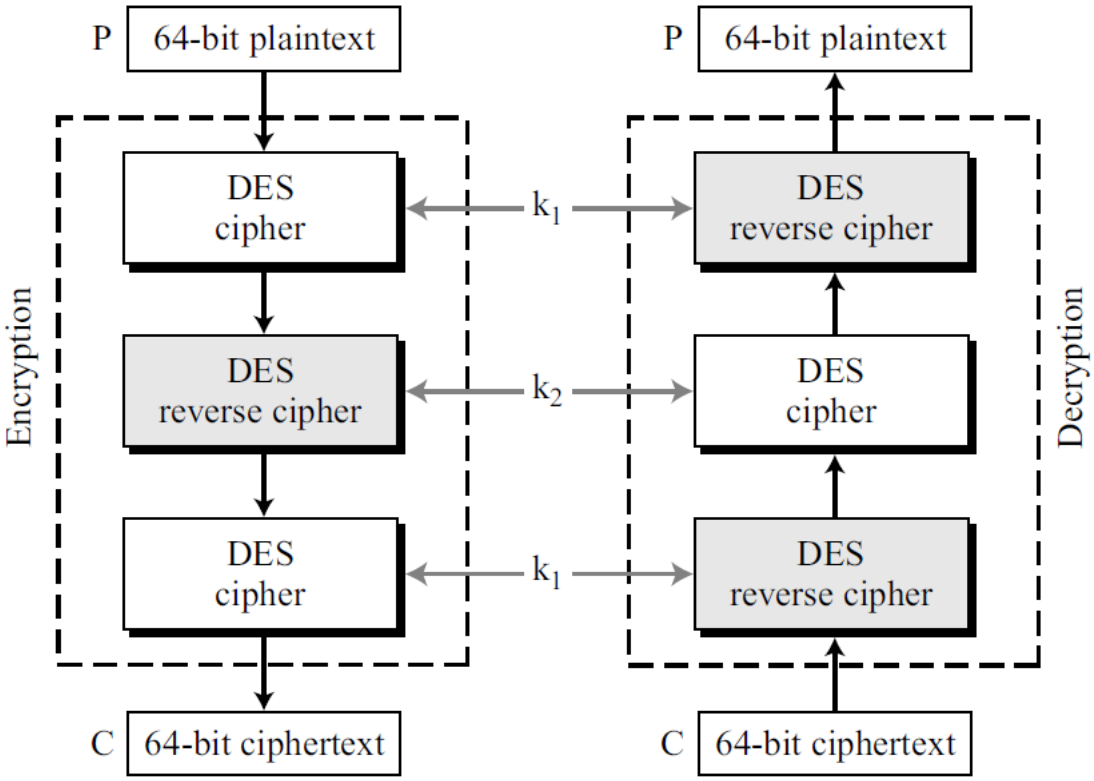
To improve the security of DES, triple DES (3DES) was proposed. This uses three stages of DES for encryption and decryption. Two versions of triple DES are in use today: triple DES with two keys and triple DES with three keys.

Triple DES with Two Keys

In triple DES with two keys, there are only two keys: K_1 and K_2. The first and the third stages use K_1; the second stage uses K_2. To make triple DES compatible with single DES,

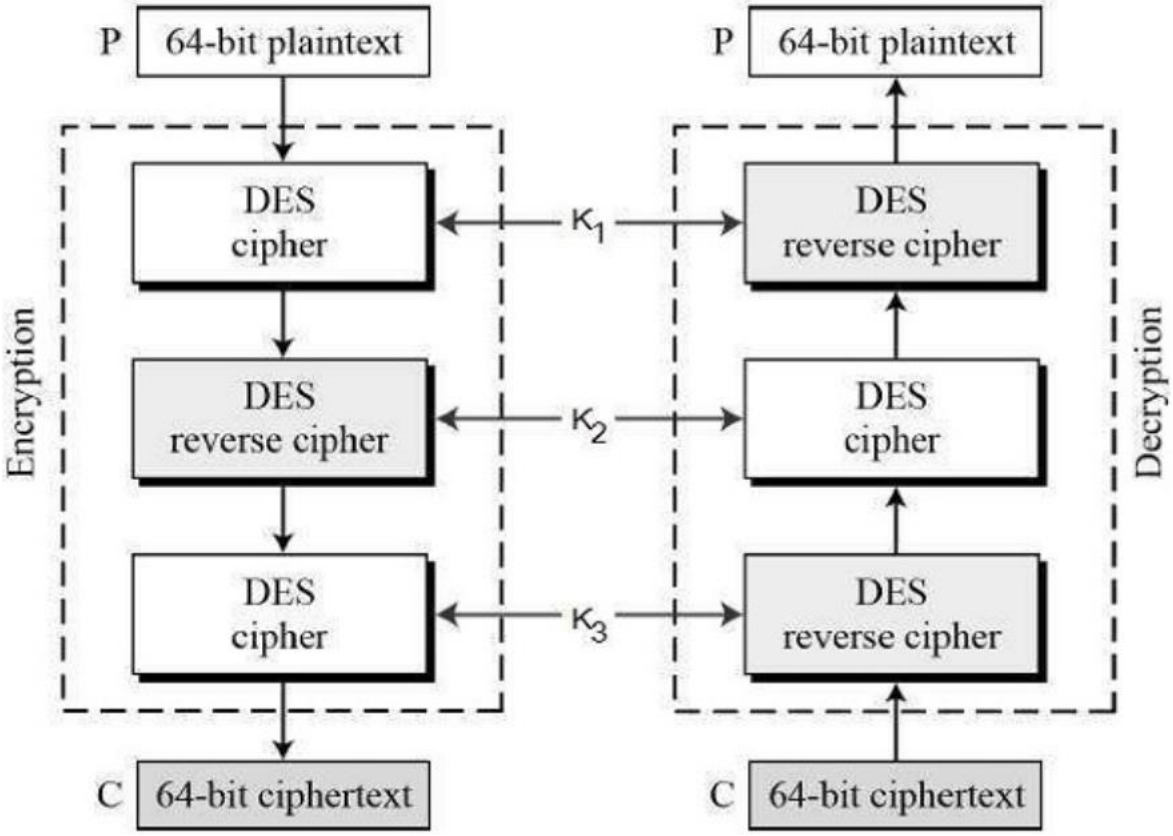
the middle stage uses decryption (reverse cipher) in the encryption site and encryption (cipher) in the decryption site. In this way, a message encrypted with single DES with key k can be decrypted with triple DES if $K_1 = K_2 = K$. Although triple DES with two keys is also vulnerable to a known-plaintext attack, it is much stronger than double DES. It has been adopted by the banking industry.

Triple DES with two keys



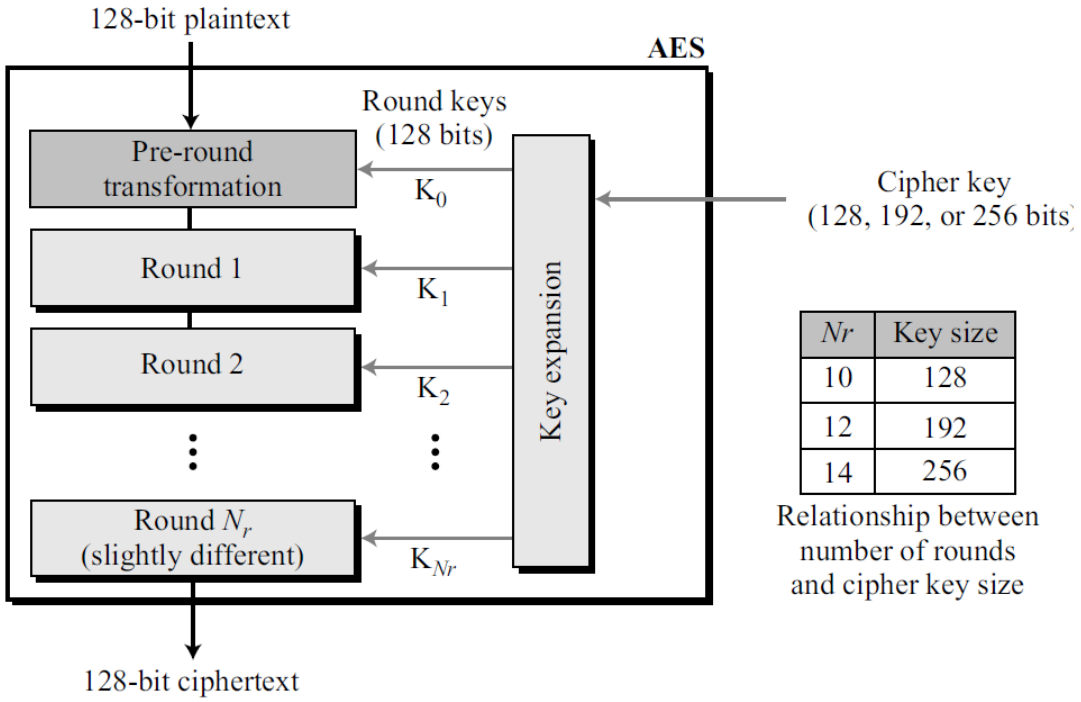
Triple DES with Three Keys

The possibility of known-plaintext attacks on triple DES with two keys has enticed some applications to use triple DES with three keys. Although the algorithm can use three DES cipher stages at the encryption site and three reverse cipher stages at the decryption site, to be compatible with single DES, the encryption site uses EDE and the decryption site uses DED (E stands for encryption and D stands for decryption). Compatibility with single DES is provided by letting $K_1 = K$ and setting K_2 and K_3 to the same arbitrary key chosen by the receiver.


P
64-bit plaintext
DES
cipher
DES
reverse cipher
DES
cipher
K1
K2
K3
Encryption
C
64-bit ciphertext
P
64-bit plaintext
DES
reverse cipher
DES
cipher
DES
reverse cipher
Decryption
C
64-bit ciphertext

Advanced Encryption Standard (AES)

General design of AES encryption cipher



Nr	Key size
10	128
12	192
14	256

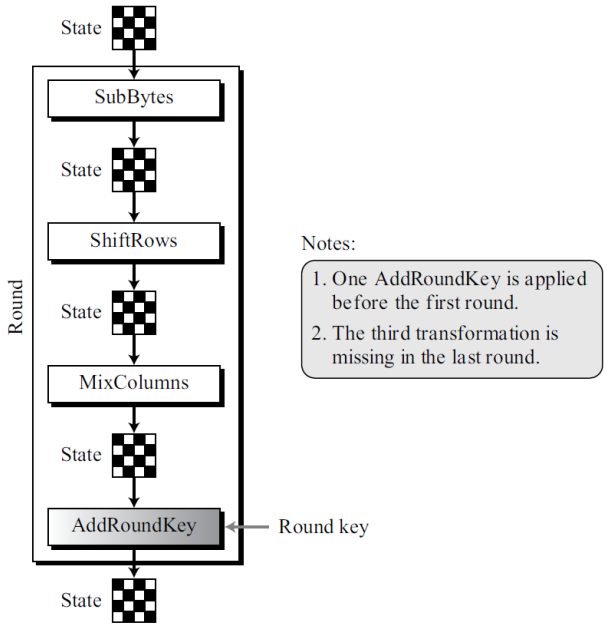
Relationship between number of rounds and cipher key size

TRANSFORMATIONS:

To provide security, AES uses four types of transformations: 1. substitution, 2. Permutation (Shift rows), 3. mixing, and 4. key-adding (Add round key). We will discuss each here.

Structure of Each Round:

- The figure below shows the structure of each round at the encryption side. Each round, except the last, uses four transformations that are invertible. The last round has only three transformations.
- As figure (given in below) shows, each transformation takes a state and creates another state to be used for the next transformation or the next round. The pre-round section uses only one transformation (AddRoundKey); the last round uses only three transformations (MixColumns transformation is missing).
- At the decryption site, the inverse transformations are used: InvSubByte, InvShiftRows, InvMixColumns, and AddRoundKey (this one is self-invertible).



Structure of each round at the encryption site

1. Substitution:

AES, like DES, uses substitution. However, the mechanism is different. First, the substitution is done for each byte. Second, only one table is used for transformation of every byte, which means that if two bytes are the same, the transformation is also the same. Third, the transformation is defined by either a table lookup process.

SubBytes:

The first transformation, SubBytes, is used at the encryption site. To substitute a byte, we interpret the byte as two hexadecimal digits. The left digit defines the row and the right digit defines the column of the substitution table. The two hexadecimal digits at the junction of the row and the column are the new byte. In the SubBytes transformation, the state is treated as a 4×4 matrix of bytes. Transformation is done one byte at a time. The contents of each byte is changed, but the arrangement of the bytes in the matrix remains the same. In the process, each byte is transformed independently. There are sixteen distinct byte-to-byte transformations.

SubBytes transformation table

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8

SubBytes transformation table (continued)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Inverse SubBytes transformation table:

Inverse SubBytes is the inverse of SubBytes. The transformation is done using following table.
We can easily check that the two transformations are inverse of each other.

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

2. Permutation:

Another transformation found in a round is shifting, which permutes the bytes. Unlike DES, in which permutation is done at the bit level, shifting transformation in AES is done at the byte level; the order of the bits in the byte is not changed.

ShiftRows:

In the encryption, the transformation is called ShiftRows and the shifting is to the left.The number of shifts depends on the row number (0, 1, 2, or 3) of the state matrix. This means the row 0 is not shifted at all and the last row is shifted three bytes.

Inverse ShiftRows:

In the decryption, the transformation is called InvShiftRows and the shifting is to the right. The number of shifts is the same as the row number (0, 1, 2, and 3) of the state matrix.

3. Mixing:

The substitution provided by the SubBytes transformation changes the value of the byte based only on original value and an entry in the table; the process does not include the neighboring bytes. We can say that SubBytes is an intrabyte transformation. The permutation provided by the ShiftRows transformation exchanges bytes without permuting the bits inside the bytes

MixColumns:

The MixColumns transformation operates at the column level; it transforms each column of the state to a new column. The transformation is actually the matrix multiplication of a state column by a constant square matrix.

Inverse MixColumns:

The InvMixColumns transformation is basically the same as the MixColumns transformation. If the two constant matrices are inverses of each other, it is easy to prove that the two transformations are inverses of each other.

Constant matrices used by MixColumns and InvMixColumns

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{Inverse} \quad \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}$$

$$C \qquad\qquad C^{-1}$$

4. Key Adding:

Probably the most important transformation is the one that includes the cipher key. All previous transformations use known algorithms that are invertible. If the cipher key is not added to the state at each round, it is very easy for the adversary to find the plaintext, given the ciphertext.

AddRoundKey:

AddRoundKey also proceeds one column at a time. It is similar to MixColumns in this respect. MixColumns multiplies a constant square matrix by each state column; AddRoundKey adds a round key word with each state column matrix. The operation in MixColumns is matrix multiplication; the operation in AddRoundKey is matrix addition.

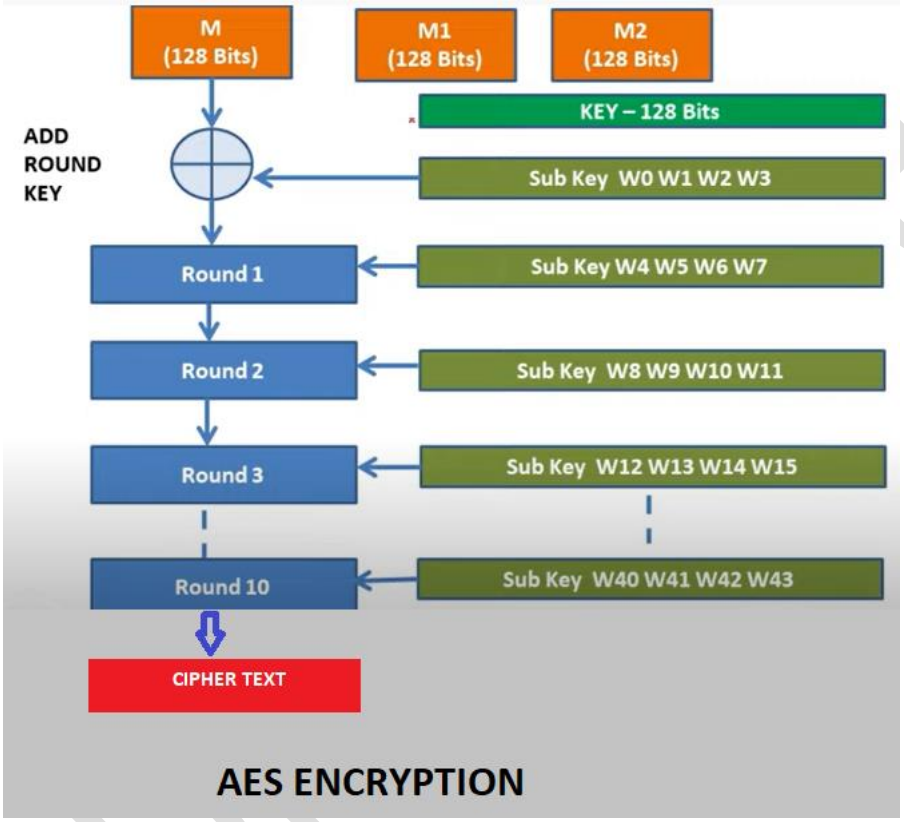
KEY EXPANSION:

To create round key for each round, AES uses a key-expansion process. If the number of rounds is Nr , the key-expansion routine creates (Nr + 1) 128-bit round keys from one single 128-bit cipher key. The first round key is used for pre-round transformation (AddRoundKey); the remaining round keys are used for the last transformation (AddRoundKey) at the end of each round. The key-expansion routine creates round keys word by word, where a word is an array of four bytes. The routine creates 4 × (Nr +1) words that are called $w_0, w_1, w_2, \ldots, w_{4(N_r+1)-1}$

In other words, in the AES-128 version (10 rounds), there are 44 words; in the AES- 192 version (12 rounds), there are 52 words; and in the AES-256 version (with 14 rounds), there are 60 words. Each round key is made of four words. Below table shows the relationship between rounds and words.

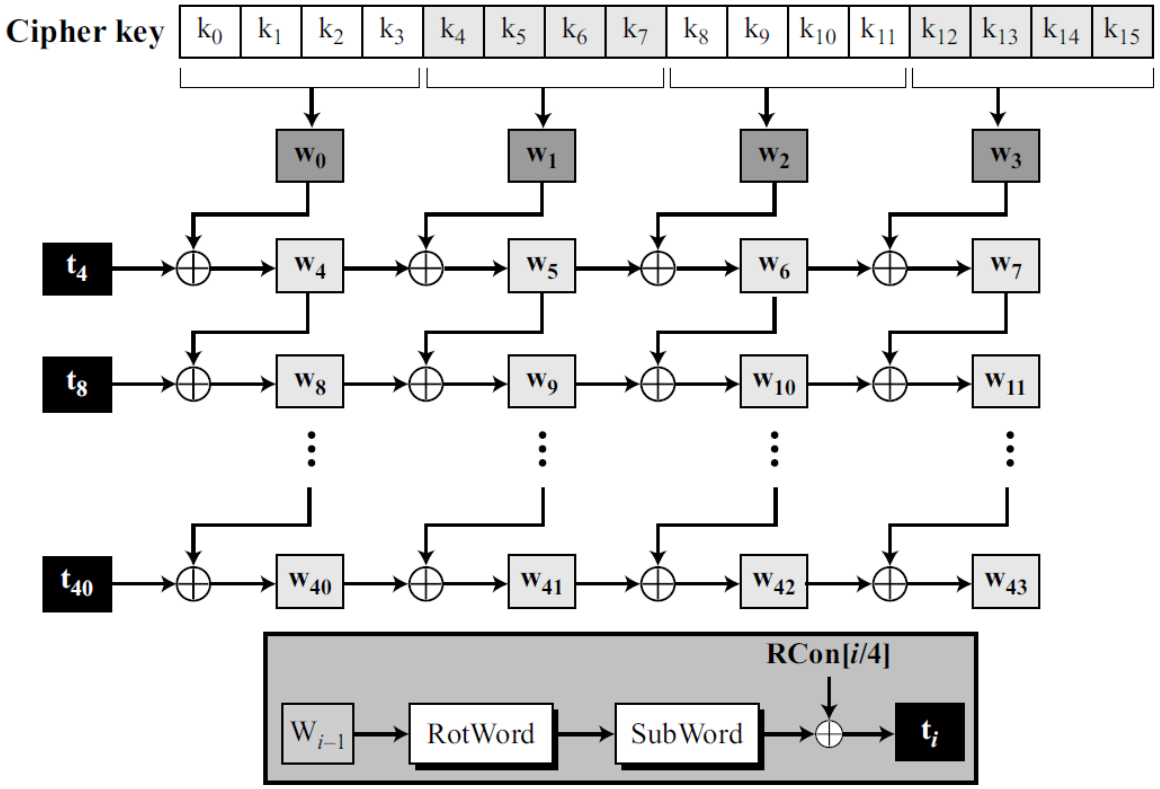
Table: *Words for each round*

Round	*Words*			
Pre-round	$\mathbf{w}_0$	$\mathbf{w}_1$	$\mathbf{w}_2$	$\mathbf{w}_3$
1	$\mathbf{w}_4$	$\mathbf{w}_5$	$\mathbf{w}_6$	$\mathbf{w}_7$
2	$\mathbf{w}_8$	$\mathbf{w}_9$	$\mathbf{w}_{10}$	$\mathbf{w}_{11}$
. . .	. . .			
N_r	$\mathbf{w}_{4N_r}$	$\mathbf{w}_{4N_r+1}$	$\mathbf{w}_{4N_r+2}$	$\mathbf{w}_{4N_r+3}$



Key Expansion in AES-128:

Let us show the creation of words for the AES-128 version; the processes for the other two versions are the same with some slight changes. The figure shows how 44 words are made from the original key.



Making of t_i (temporary) words $i = 4\ N_r$

The process is as follows:

1. The first four words (w_0, w_1, w_2, w_3) are made from the cipher key. The cipher key is thought of as an array of 16 bytes (k0 to k15). The first four bytes (k0 to k3) become w_0; the next four bytes (k4 to k7) become w_1; and so on. In other words, the concatenation of the words in this group replicates the cipher key.
2. The rest of the words (w_i for i = 4 to 43) are made as follows:

a. If $(i \bmod 4) \neq 0$, $W_i = W_{i-1} \oplus W_{i-4}$. Referring to above figure, this means each word is made from the one at the left and the one at the top.

b. If $(i \bmod 4) = 0$, $W_i = t \oplus W_{i-4}$. Here *t*, a temporary word, is the result of applying two routines, SubWord and RotWord, on wi−1 and XORing the result with a round constants, RCon. In other words, we have,

$$\mathbf{t} = \text{SubWord}\ (\text{RotWord}\ (\mathbf{w}_{i-1}))\ \oplus\ \text{RCon}_{i/4}$$

RotWord:

The RotWord (rotate word) routine is similar to the ShiftRows transformation, but it is applied to only one row. The routine takes a word as an array of four bytes and shifts each byte to the left with wrapping.

SubWord

The SubWord (substitute word) routine is similar to the SubBytes transformation, but it is applied only to four bytes. The routine takes each byte in the word and substitutes another byte for it.

Round Constants

Each round constant, RCon, is a 4-byte value in which the rightmost three bytes are always zero. Table below shows the values for AES-128 version (with 10 rounds).

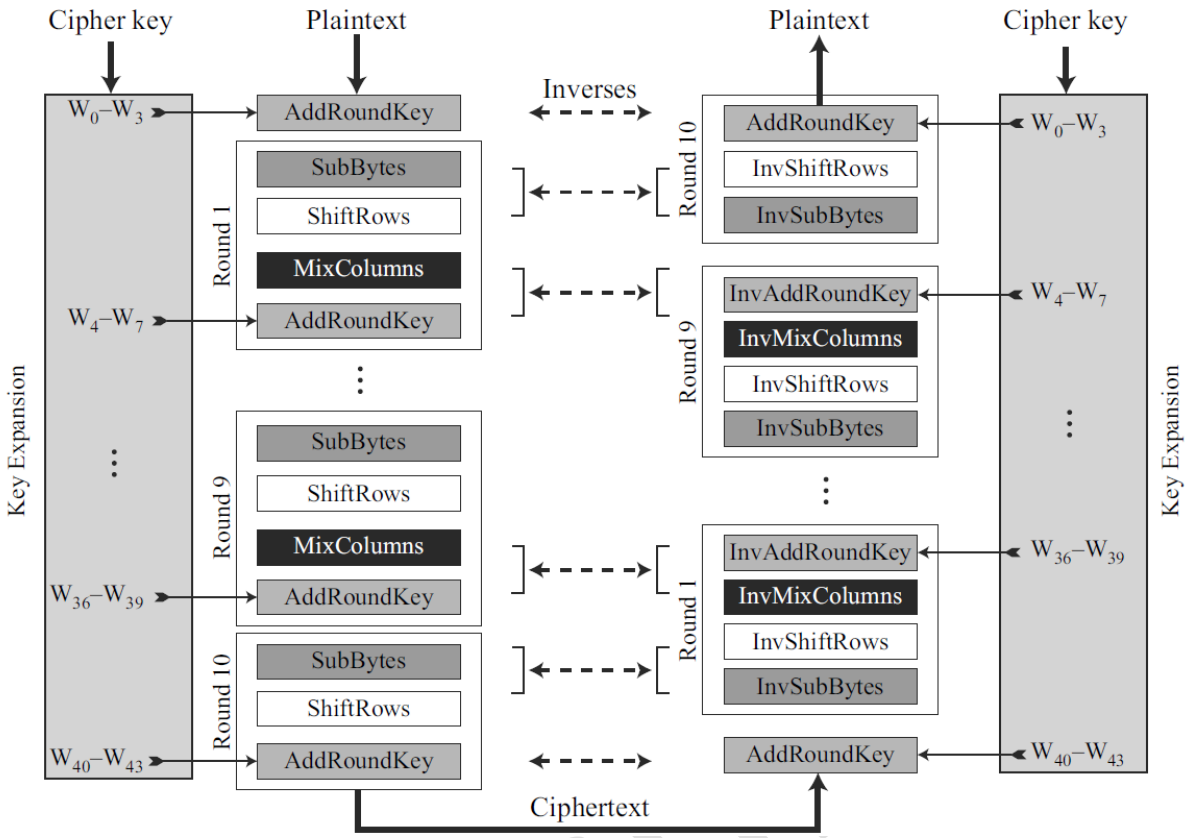
RCon constants

Round	*Constant (RCon)*	*Round*	*Constant (RCon)*
1	$(\underline{\mathbf{01}}\ 00\ 00\ 00)_{16}$	6	$(\underline{\mathbf{20}}\ 00\ 00\ 00)_{16}$
2	$(\underline{\mathbf{02}}\ 00\ 00\ 00)_{16}$	7	$(\underline{\mathbf{40}}\ 00\ 00\ 00)_{16}$
3	$(\underline{\mathbf{04}}\ 00\ 00\ 00)_{16}$	8	$(\underline{\mathbf{80}}\ 00\ 00\ 00)_{16}$
4	$(\underline{\mathbf{08}}\ 00\ 00\ 00)_{16}$	9	$(\underline{\mathbf{1B}}\ 00\ 00\ 00)_{16}$
5	$(\underline{\mathbf{10}}\ 00\ 00\ 00)_{16}$	10	$(\underline{\mathbf{36}}\ 00\ 00\ 00)_{16}$

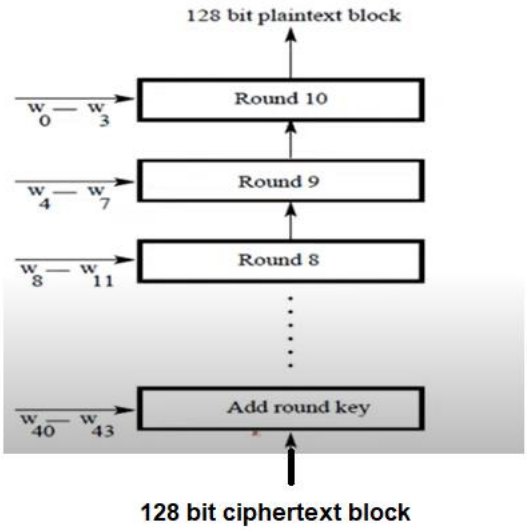
The key-expansion routine can either use the above table when calculating the words or use the GF(28) field to calculate the leftmost byte dynamically, as shown below (prime is the irreducible polynomial):

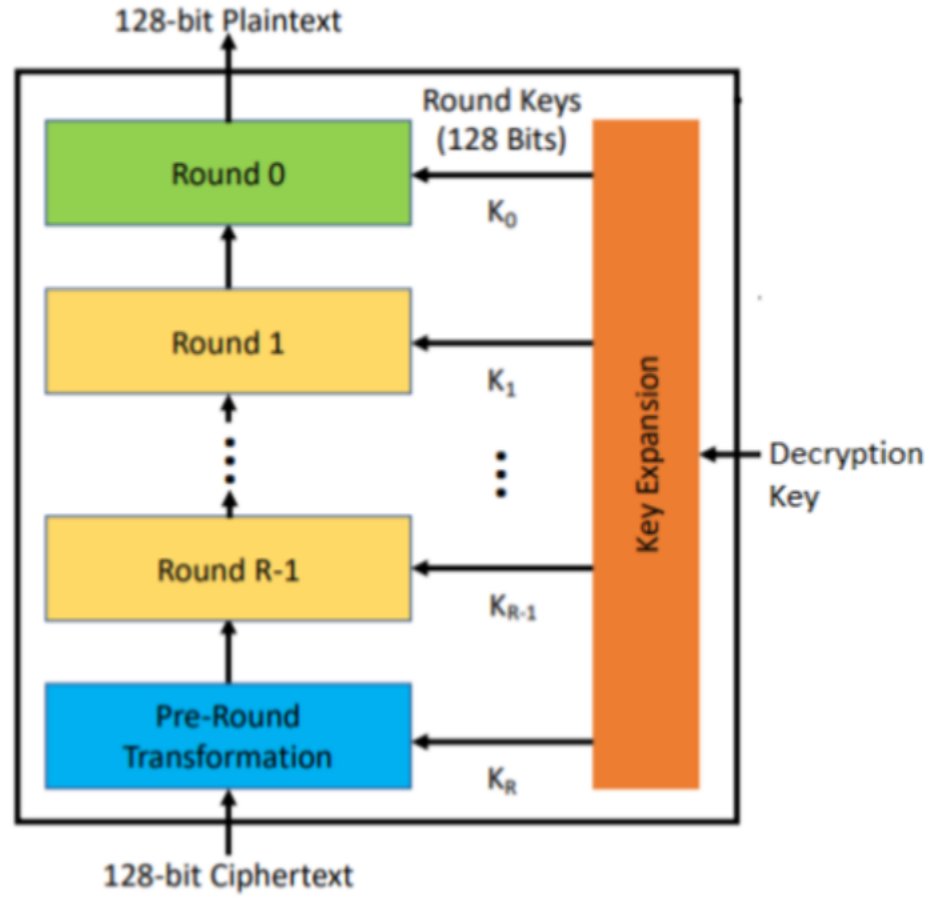
$$
\begin{array}{llllll}
RC_1 & \rightarrow x^{1-1} & = x^0 \bmod prime & = 1 & \rightarrow 00000001 & \rightarrow 01_{16} \\
RC_2 & \rightarrow x^{2-1} & = x^1 \bmod prime & = x & \rightarrow 00000010 & \rightarrow 02_{16} \\
RC_3 & \rightarrow x^{3-1} & = x^2 \bmod prime & = x^2 & \rightarrow 00000100 & \rightarrow 04_{16} \\
RC_4 & \rightarrow x^{4-1} & = x^3 \bmod prime & = x^3 & \rightarrow 00001000 & \rightarrow 08_{16} \\
RC_5 & \rightarrow x^{5-1} & = x^4 \bmod prime & = x^4 & \rightarrow 00010000 & \rightarrow 10_{16} \\
RC_6 & \rightarrow x^{6-1} & = x^5 \bmod prime & = x^5 & \rightarrow 00100000 & \rightarrow 20_{16} \\
RC_7 & \rightarrow x^{7-1} & = x^6 \bmod prime & = x^6 & \rightarrow 01000000 & \rightarrow 40_{16} \\
RC_8 & \rightarrow x^{8-1} & = x^7 \bmod prime & = x^7 & \rightarrow 10000000 & \rightarrow 80_{16} \\
RC_9 & \rightarrow x^{9-1} & = x^8 \bmod prime & = x^4 + x^3 + x + 1 & \rightarrow 00011011 & \rightarrow 1B_{16} \\
RC_{10} & \rightarrow x^{10-1} & = x^9 \bmod prime & = x^5 + x^4 + x^2 + x & \rightarrow 00110110 & \rightarrow 36_{16}
\end{array}
$$

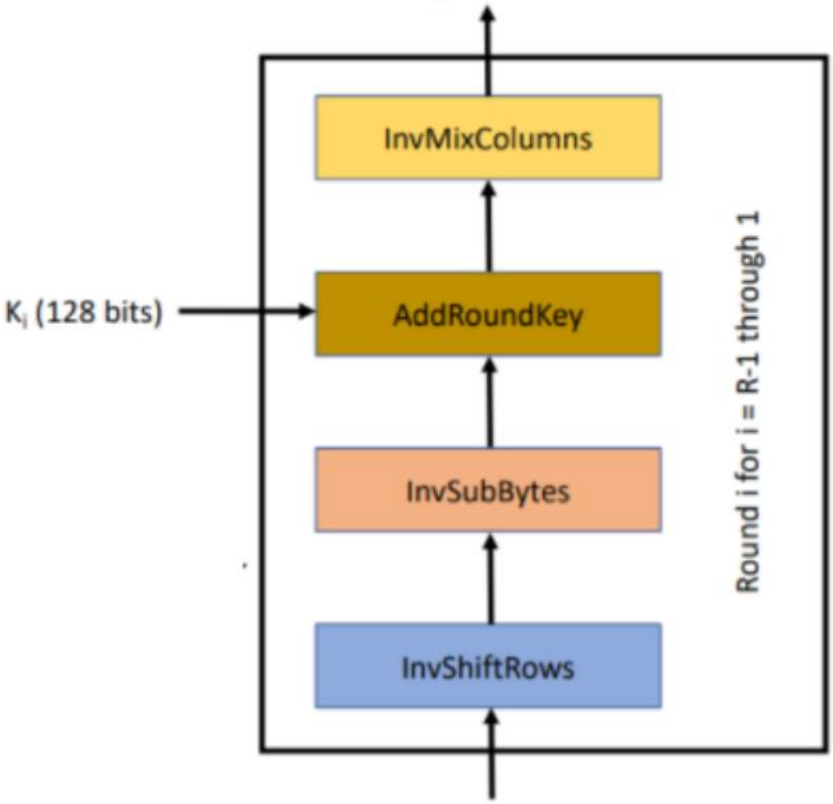
The following diagrams shows the process of decryption in AES:


Cipher key
Plaintext
Plaintext
Cipher key
Inverses
Key Expansion
Key Expansion
W0–W3
W4–W7
W36–W39
W40–W43
AddRoundKey
SubBytes
ShiftRows
MixColumns
Round 1
Round 9
Round 10
InvAddRoundKey
InvMixColumns
InvShiftRows
InvSubBytes
Ciphertext

Decryption


128 bit plaintext block
Round 10
Round 9
Round 8
Add round key
w0 — w3
w4 — w7
w8 — w11
w40 — w43
128 bit ciphertext block


128-bit Plaintext
Round Keys
(128 Bits)
Round 0
K_0
Round 1
K_1
Round R-1
K_{R-1}
Pre-Round
Transformation
K_R
Key Expansion
Decryption
Key
128-bit Ciphertext


InvMixColumns
K_i (128 bits)
AddRoundKey
InvSubBytes
InvShiftRows
Round i for i = R-1 through 1

Notes on AES (One can use either above notes or below one)

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 −

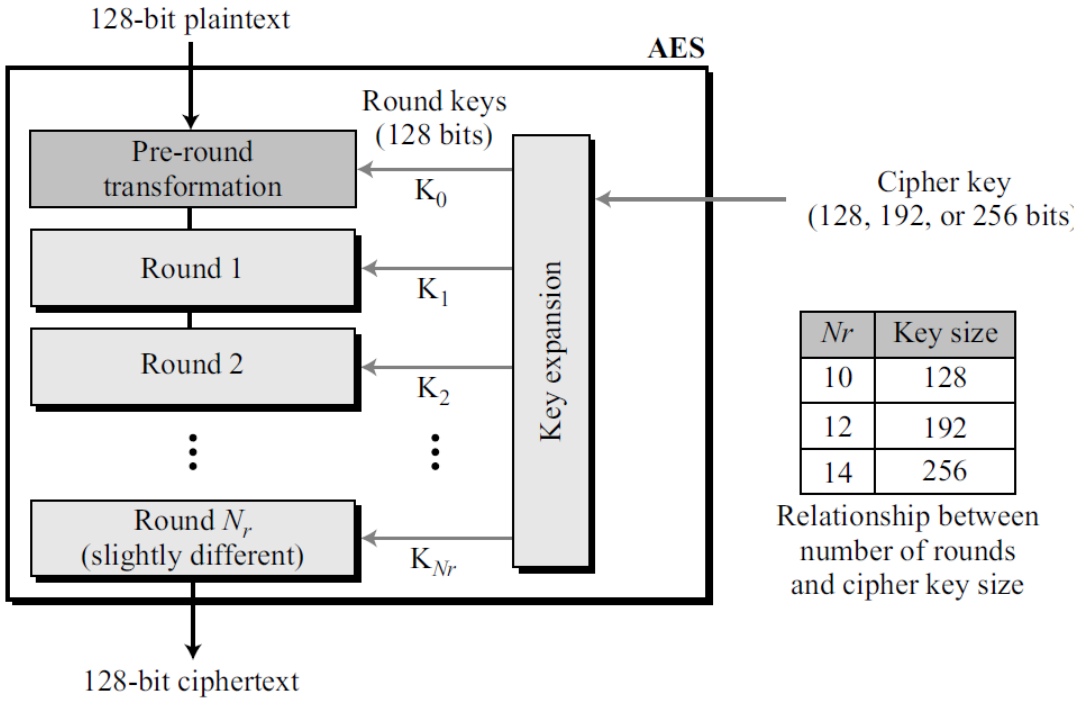
- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

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 –

General design of AES encryption cipher

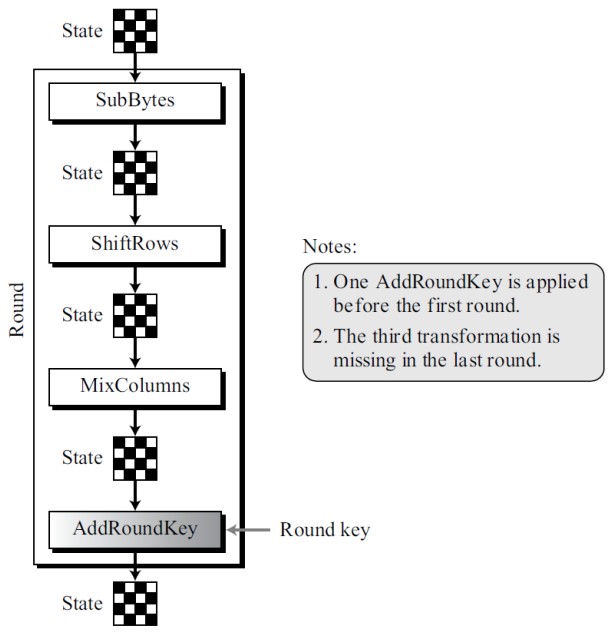


Nr	Key size
10	128
12	192
14	256

Relationship between number of rounds 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 –



Structure of each round at the encryption site

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.

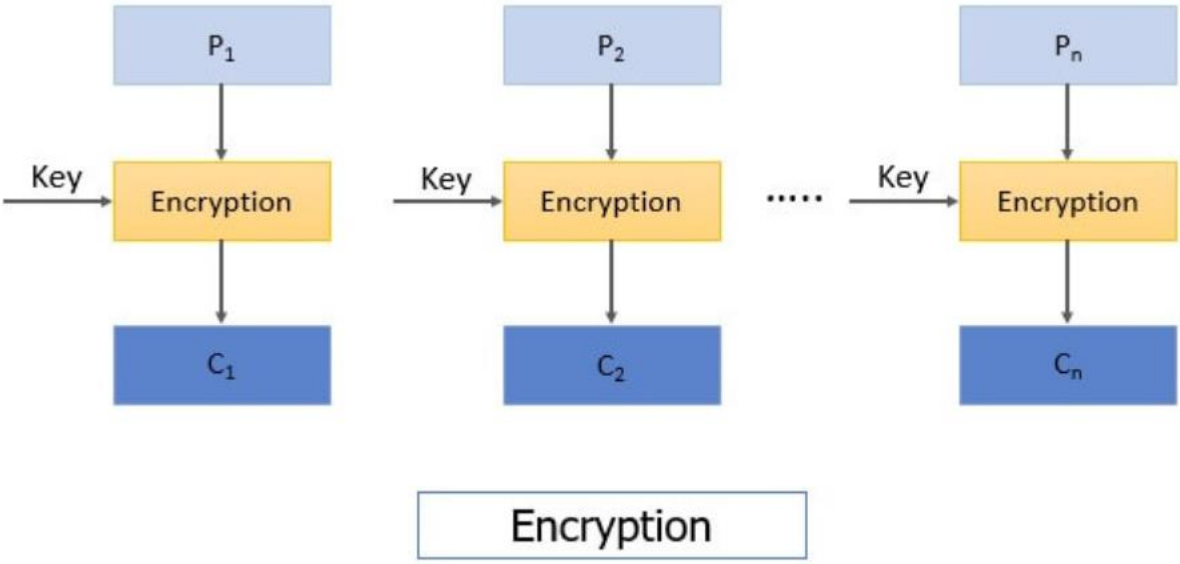
However, just as for DES, the AES security is assured only if it is correctly implemented and good key management is employed.

Block Ciphers Modes of operation:

There are five important block cipher modes of operation defined by NIST (National Institute of Standards and Technology).

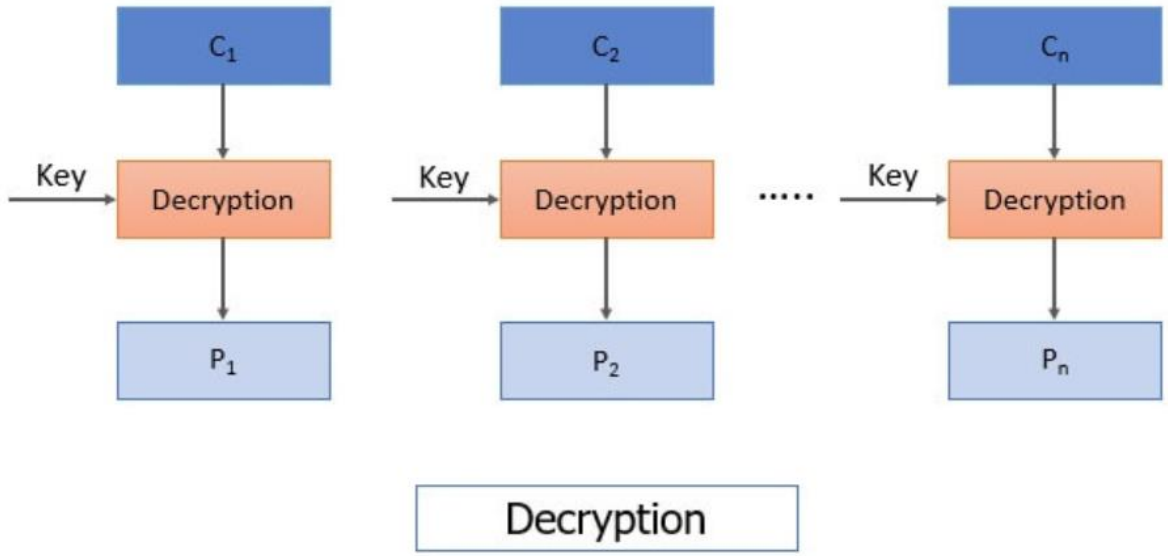
1. Electronic Code Book Mode
2. Block Chaining Mode
3. Cipher Feedback Mode
4. Output Feedback Mode
5. Counter Mode

1. **Electronic Code Book Mode(ECB):** This is considered to be the easiest block cipher mode of operation. In electronic codebook mode (ECB) the plain text is divided into the blocks, each of 64-bit. Each block is encrypted one at a time to produce the cipher block. The same key is used to encrypt each block.



When the receiver receives the message i.e. ciphertext. This ciphertext is again divided into blocks, each of 64-bit and each block is decrypted independently one at a time to obtain the corresponding plain text block. Here also the same key is used to decrypt each block which was used to encrypt each block.

As the same key used to encrypt each block of plain text there arises an issue that for a repeating plain text block it would generate the same cipher and will ease the cryptanalysis to crack the algorithm. Hence, ECB is considered for encrypting the small messages which have a rare possibility of repeating text.



2. **Cipher Block Chaining Mode (CBC)**: To overcome the limitation of ECB i.e. the repeating block in plain text produces the same ciphertext, a new technique was required which is Cipher Block Chaining (CBC) Mode. CBC confirms that even if the plain text has repeating blocks its encryption won't produce same cipher block.

   To achieve totally different cipher blocks for two same plain text blocks **chaining** has been added to the block cipher. For this, the result obtained from the encryption of the first plain text block is fed to the encryption of the next plaintext box.

   In this way, each ciphertext block obtained is dependent on its corresponding current plain text block input and all the previous plain text blocks. But during the encryption of first plain text block, no previous plain text block is available so a random block of text is generated called **Initialization vector**.

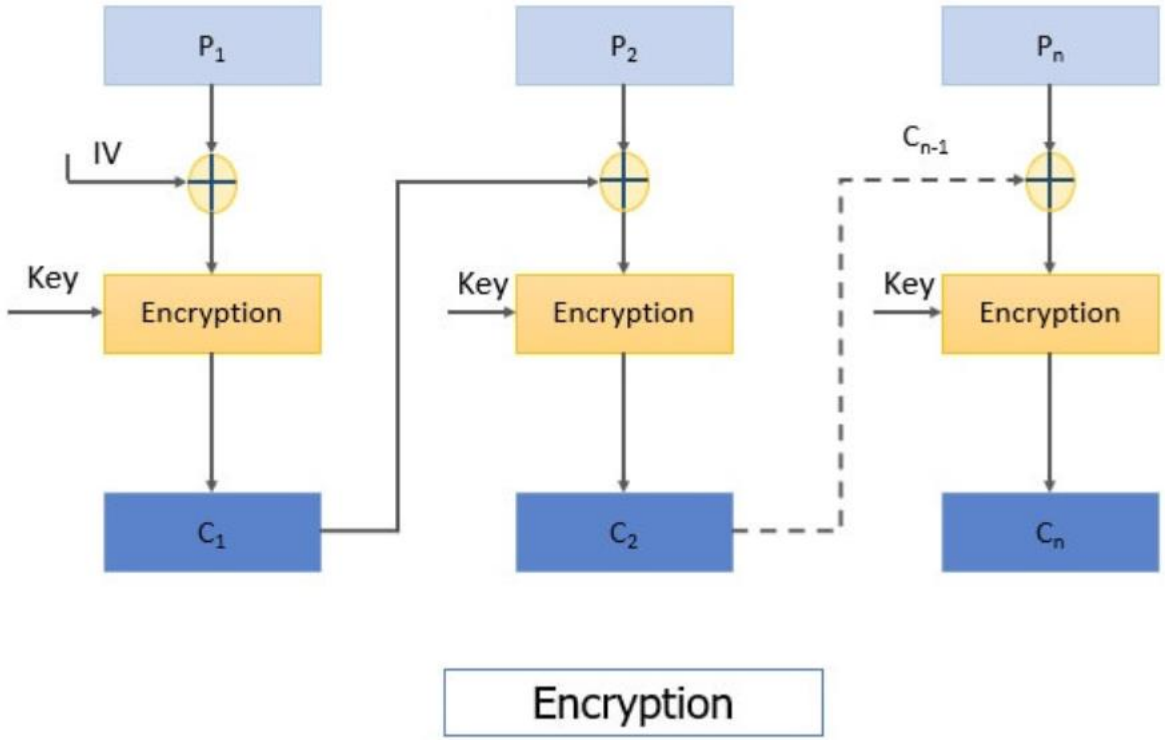
   Now let's discuss the encryption steps of CBC

   Step 1: The initialization vector and first plain text block are XORed and the result of XOR is then encrypted using the key to obtain the first ciphertext block.

   Step 2: The first ciphertext block is fed to the encryption of the second plain text block. For the encryption of second plain text block, first ciphertext block and second plain text block is XORed and the result of XOR is encrypted using the same key in step 1 to obtain the second ciphertext block.

   Similarly, the result of encryption of second plain text block i.e. the second ciphertext block is fed to the encryption of third plain text block to obtain third ciphertext block. And the process continues to obtain all the ciphertext blocks.

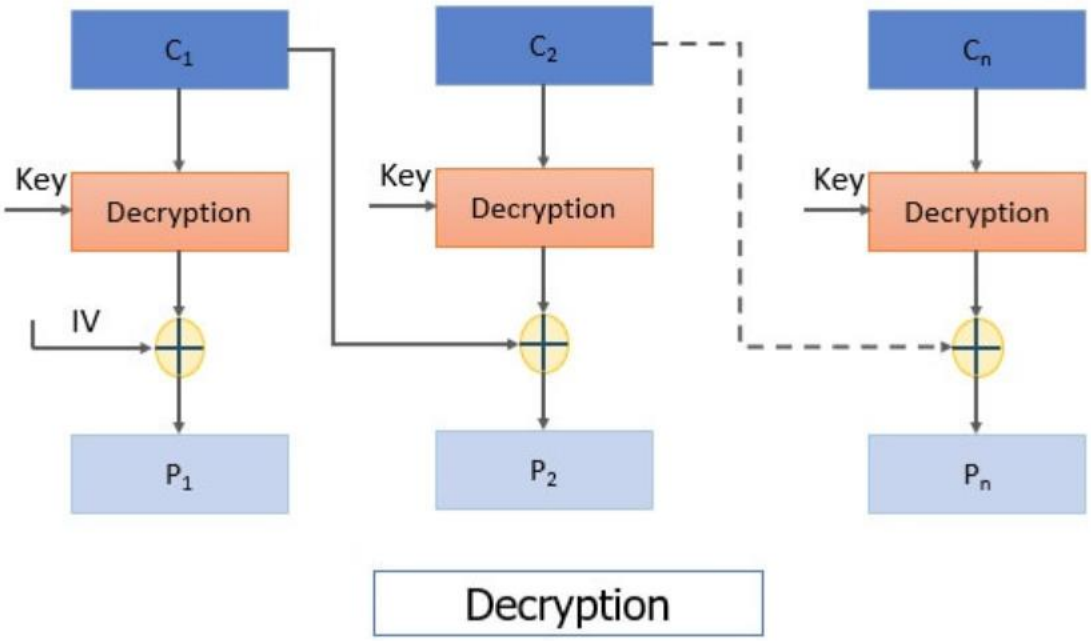
You can see the steps of CBC in the figure below:



Decryption steps of CBC:

Step 1: The first ciphertext block is decrypted using the same key that was used for encrypting all plain text blocks. The result of decryption is then XORed with the initialization vector (IV) to obtain the first plain text block.

Step 2: The second ciphertext block is decrypted and the result of decryption is XORed with the first ciphertext block to obtain the second plain text block. And the process continues till all plain text blocks are retrieved.



There was a limitation in CBC that if we have two identical messages and if we use the same IV for both the identical message it would generate the same ciphertext block.

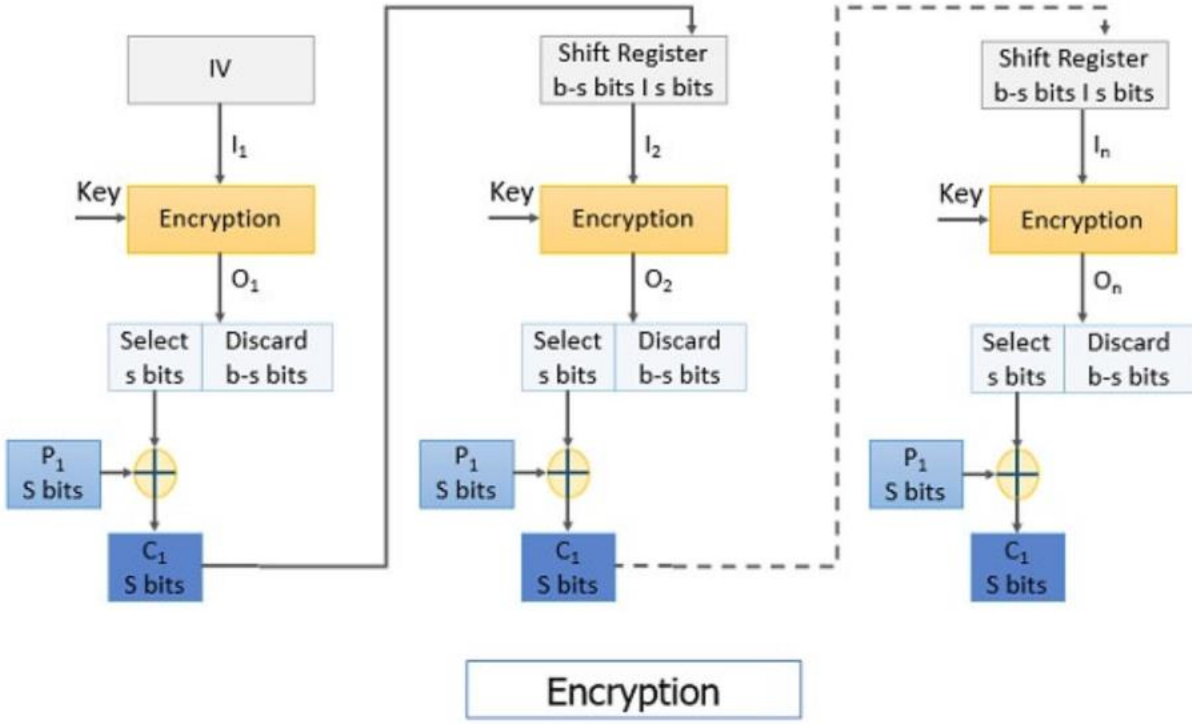
3. **Cipher Feedback Mode (CFB):** All applications may not be designed to operate on the blocks of data, some may be **character or bit-oriented**. Cipher feedback mode is used to operate on smaller units than blocks.
   Let us discuss the encryption steps in cipher feedback mode:

   Step 1: Here also we use initialization vector, IV is kept in the shift register and it is encrypted using the key.

   Step 2: The left most s bits of the encrypted IV is then XORed with the first fragment of the plain text of s bits. It produces the first ciphertext C1 of s bits.

   Step 3: Now the shift register containing initialization vector performs left shift by s bits and s bits C1 replaces the rightmost s bits of the initialization vector.

   Then again, the encryption is performed on IV and the leftmost s bit of encrypted IV is XORed with the second fragment of plain text to obtain s bit ciphertext C2.



The process continues to obtain all ciphertext fragments.

Decryption Steps:

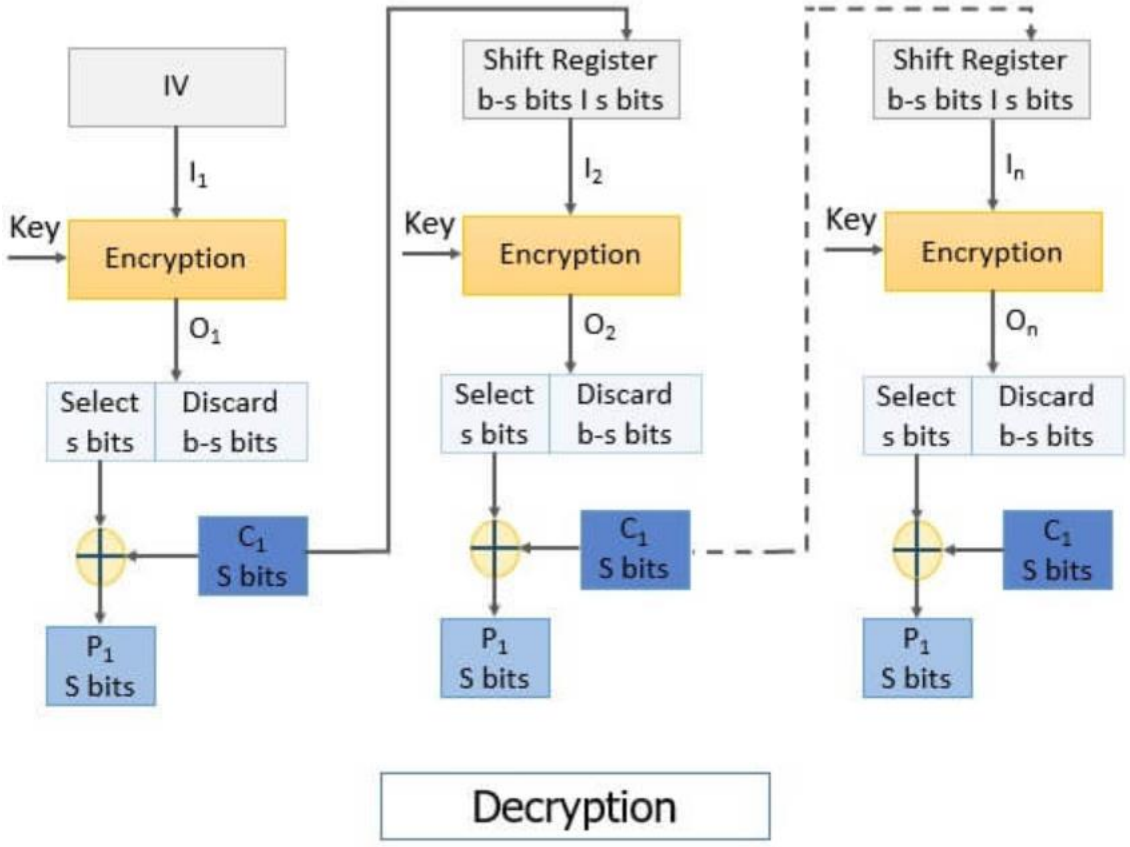
Step 1: The initialization vector is placed in the shift register. It is encrypted using the same key.

Keep a note that even in the **decryption process** the **encryption** algorithm is implemented instead of the decryption algorithm.

Then from the encrypted IV s bits are XORed with the s bits ciphertext C1 to retrieve s bits plain text P1.

Step 2: The IV in the shift register is left-shifted by s bits and the s bits C1 replaces the rightmost s bits of IV.

The process continues until all plain text fragments are retrieved.



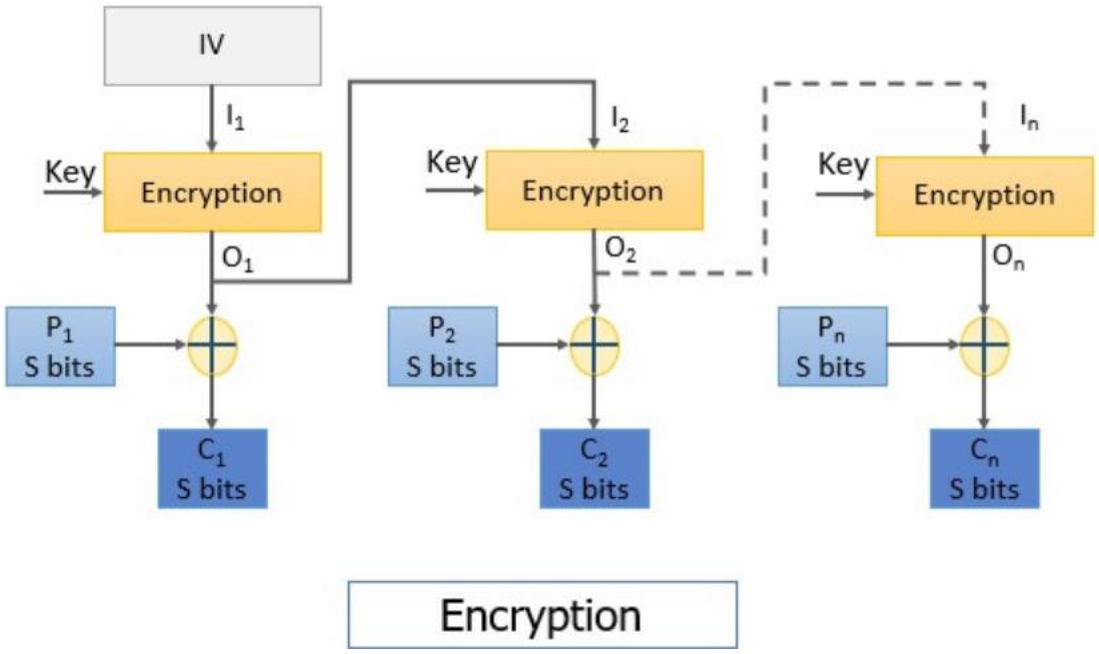
It has a limitation that if there occur a bit error in any ciphertext C_i it would affect all the subsequent ciphertext units as C_i is fed to the encryption of next P_{i+1} to obtain C_{i+1} . In this way, bit error would propagate.

4. **Output Feedback Mode (OFB):** The output feedback (OFB) mode is almost similar to the CFB. The difference between CFB and OFB is that unlike CFB, in OFB the encrypted IV is fed to the encryption of next plain text block. The other difference is that CFB operates on a stream of bits whereas OFB operates on the block of bits.

Steps for encryption:

Step 1: The initialization vector is encrypted using the key.

Step 2: The encrypted IV is then XORed with the plain text block to obtain the ciphertext block.
The encrypted IV is fed to the encryption of next plain text block as you can see in the image below.



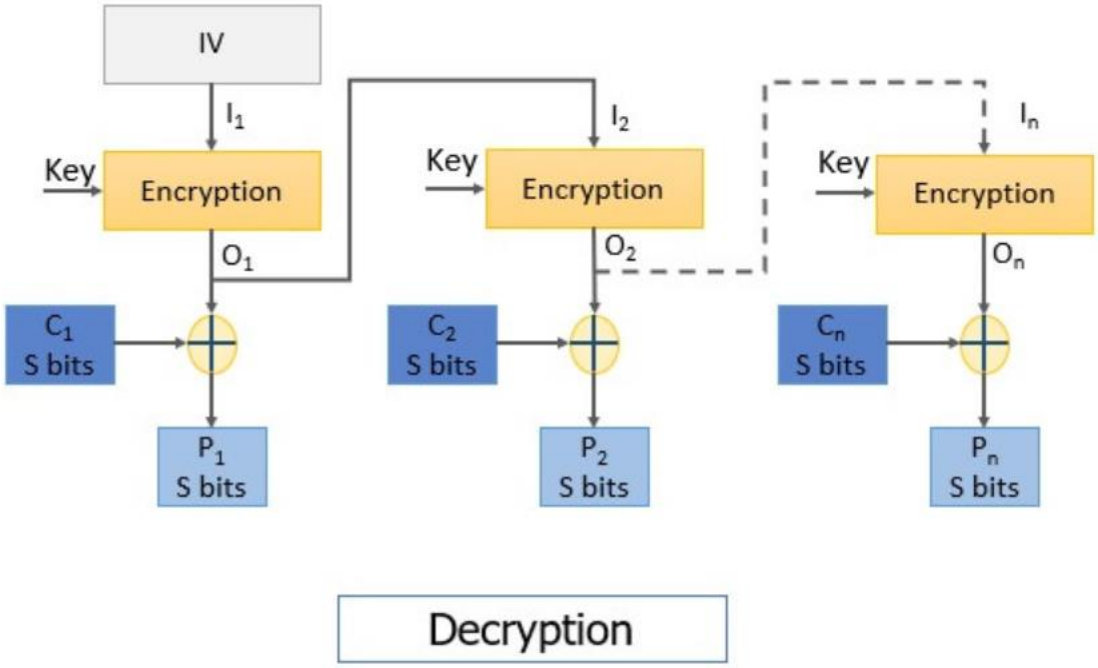
Steps for decryption:

Step 1: The initialization vector is encrypted using the same key used for encrypting all plain text blocks.

Note: In the **decryption process** also the **encryption** function is implemented.

Step2: The encrypted IV is then XORed with the ciphertext block to retrieve the plain text block.
The encrypted IV is also fed to the decryption process of the next ciphertext block.

The process continues until all the plain text blocks are retrieved.



The advantage of OFB is that it protects the propagation of bit error means the if there occurs a bit error in C_1 it will only affect the retrieval of P_1 and won't affect the retrieval of subsequent plain text blocks.

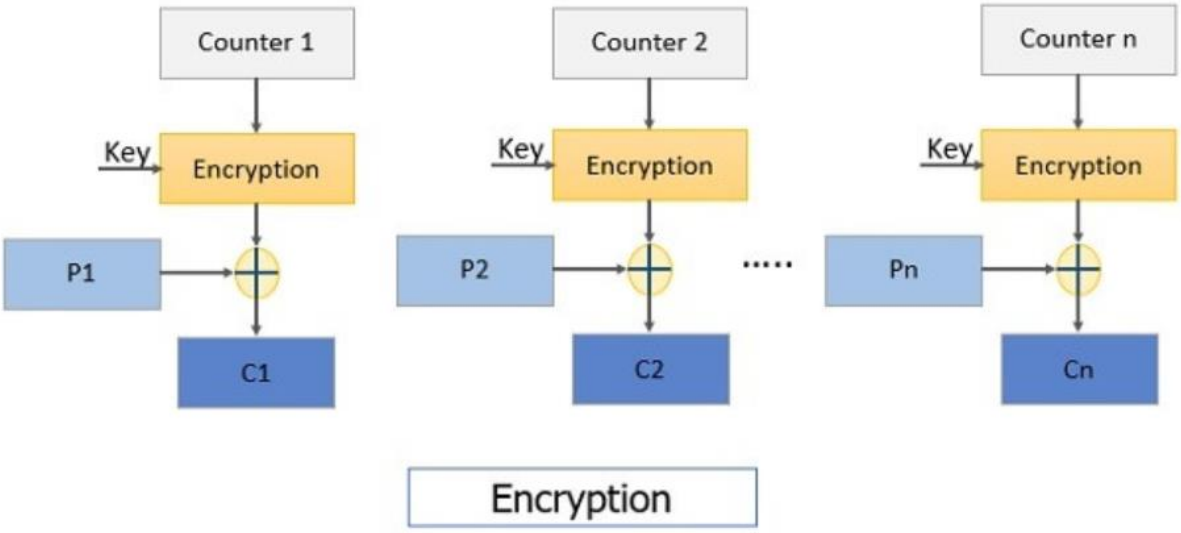
5. **Counter Mode(CTR):** It is similar to OFB but there is no feedback mechanism in counter mode. Nothing is being fed from the previous step to the next step instead it uses a sequence of number which is termed as a **counter** which is input to the encryption function along with the key. After a plain text block is encrypted the counter value increments by 1.

   Steps of encryption:

   Step1: The counter value is encrypted using a key.

   Step 2: The encrypted counter value is XORed with the plain text block to obtain a ciphertext block.

   To encrypt the next subsequent plain text block the counter value is incremented by 1 and step 1 and 2 are repeated to obtain the corresponding ciphertext.



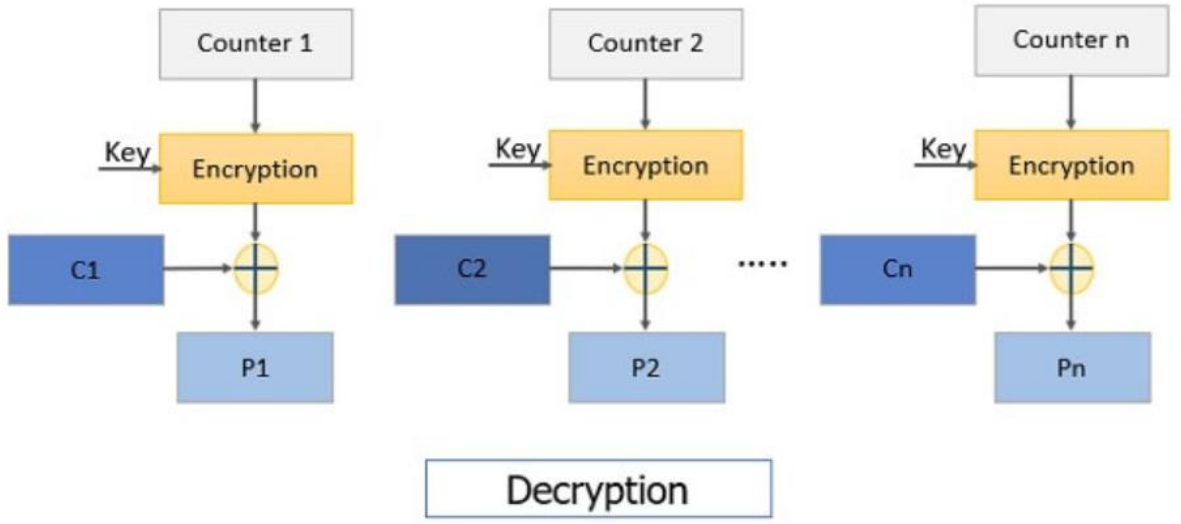
The process continues until all plain text block is encrypted.

Steps for decryption:

Step1: The counter value is encrypted using a key.

Note: Encryption function is used in the decryption process. The same counter values are used for decryption as used while encryption.

Step 2: The encrypted counter value is XORed with the ciphertext block to obtain a plain text block.



To decrypt the next subsequent ciphertext block the counter value is incremented by 1 and step 1 and 2 are repeated to obtain corresponding plain text.

The process continues until all ciphertext block is decrypted.
So, this is all about Block cipher, its designing principles and its mode of operation.

RC4 algorithm

RC stands for Ron's Cipher after Ron Rivest of MIT. You should not think that the RC4 cipher is a prior version of the block ciphers RC5 and RC6. It is in fact a very, very fast stream cipher. It is very easy to remember since it is surprisingly simple.

Given an array S (State vector) indexed from 0 to 255 consisting of the integers 0, 1, 2, . . . , 255, permuted in some key-dependent way, the output of the RC4 algorithm is a keystream of bytes K (encryption key) which is XORed with the plaintext byte by byte. Since the algorithm works on bytes and not bits, and uses very simple operations it is particularly fast in software. We start by letting i = 0 and j = 0, we then repeat the steps as in the following Algorithm:

Algorithm : RC4 Algorithm

$i = (i + 1) \mod 256$

$j = (j + S_i) \mod 256$

$\text{swap}(S_i, S_j)$

$t = (S_i + S_j) \mod 256$

$K = S_t$

Where t= temporary vector. The initial state of the array S is determined from the key using the method described by the following Algorithm:

Algorithm : RC4 Key Schedule

for $i = 0$ **to** 255 **do** $S_i = i$

Initialise K_i, for $i = 0, \ldots, 255$, with the key, repeating if neccesary

$j = 0$

for $i = 0$ **to** 255 **do**

  $j = (j + S_i + K_i) \mod 256$

  $\text{swap}(S_i, S_j)$

end

Where $K_i = S_t$.

Although RC4 is very fast for a software-based stream cipher, there are some issues with its use. In particular both the key schedule and the main algorithm do not produce as random a stream as one might wish. Hence, it should be used with care.

Pseudo Random Number Generators:

Principles of Pseudo Random Number Generators:

While generating pseudo random numbers we need to consider the following three properties namely (1) Uniform Distribution (2) Independence and (3) Unpredictability.

(1) Uniform Distribution:

The Distribution of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same.

That is, every value of any random number used in a cryptographic system must be equally likely to appear. This requirement is of utmost importance since a biased probability distribution would open the door to an attacker e.g. by making frequency attacks possible.

(2) **Independence**: No one value in the sequence can be inferred from the others. You should not be able to derive a subsequence from other numbers. There are some tests to check uniform distribution but not for independence. That is, there are well-defined tests for determining that a sequence of numbers matches a particular distribution, such as the uniform distribution, there is no such test to "prove" independence. Rather, a number of tests can be applied to demonstrate if a sequence does not exhibit independence. The general strategy is to apply a number of such tests until the confidence that independence exists is sufficiently strong.

(3) **Unpredictability:** In applications such as reciprocal authenticate (or) stream ciphers, numbers should not only be statistically random but also successive numbers of the sequence must be unpredictable. **(or)**

That is, Random numbers, especially those used for secret parameters such as keys, must be unpredictable to prevent an attacker from being able to compute future or preceding values from the already generated and captured data.

Randomness and entropy:

As we've seen in the previous sections, cryptography often requires randomness. For example, symmetric keys are usually randomly-generated bits, and random IVs (Initial Vectors) and nonces are required to build secure block cipher chaining modes.

In cryptography, when we say "random," we usually mean "random and unpredictable." For example, flipping a biased coin that comes up heads 99% of the time is random, but you can predict a pattern–for a given coin toss, if you guess heads, it's very likely you're correct. A better source of randomness for cryptographic purposes would be flipping a fair coin, because the outcome is harder to predict than the outcome of the biased coin flip. Consider generating a random symmetric key: you would want to use outcomes of the fair coin to generate the key, because that makes it harder for the attacker to guess your key than if you had used outcomes of the biased coin to generate the key.

We can formalize this concept of unpredictability by defining *entropy*, a measure of uncertainty or surprise, for any random event. The biased coin has low entropy because you expect a given outcome most of the time. The fair coin has high entropy because you are very uncertain about the outcome. The specifics of entropy are beyond the scope of this class, but an important note is that the uniform distribution (all outcomes equally likely) produces the greatest entropy. In cryptography, we generally want randomness with the most entropy, so ideally, any randomness should be bits drawn from a uniform distribution (i.e. The distribution of numbers in the sequence should be uniform; that is, the frequency of occurrence of each of the numbers should be approximately the same. E.g. the outcomes of fair coin tosses).

Instead of using expensive true randomness each time a cryptographic algorithm requires randomness, we instead use *pseudo-randomness*. Pseudorandom numbers are generated deterministically using an algorithm, but they look random.

Pseudorandom Numbers: Cryptographic applications typically make use of algorithmic techniques for random number generation. These algorithms are deterministic and therefore produce sequences of numbers that are not statistically random. However, if the algorithm is good, the resulting sequences will pass many reasonable tests of randomness. Such numbers are referred to as **pseudorandom numbers**.

Deterministic / Pseudorandom Number Generators (pRNGs):

A *pseudorandom number generator (pRNG)* is an algorithm that takes a small amount of truly random bits as input and outputs a long sequence of pseudorandom bits. The initial truly random input is called the *seed*.

The pRNG algorithm is deterministic, so anyone who runs the pRNG with the same seed will see the same pseudorandom output. However, to an attacker who doesn't know the seed, the output of a secure pRNG is computationally indistinguishable from true random bits. A pRNG is not completely indistinguishable from true random bits–given infinite computational time and power, an attacker can distinguish pRNG output from truly random output. If the pRNG takes in an n-bit seed as input, the attacker just has to input all 2^n possible seeds and see if any of the 2^n outputs matches the bitstring they received. However, when restricted to any practical computation limit, an attacker has no way of distinguishing pRNG output from truly random output. Output sequence of a good pRNG is perfectly uniformly distributed. pRNGs achieve high output bit rates

It would be very inefficient if a pRNG only outputted a fixed number of pseudorandom bits for each truly random input. If this were the case, we would have to generate more true randomness each time the pRNG output has all been used. Ideally, we would like the pRNG to take in an initial seed and then be available to generate as many pseudorandom bits as needed on demand. To achieve this, the pRNG maintains some internal state and updates the state any time the pRNG generates new bits or receives a seed as input.

Formally, a pRNG is defined by the following three functions:

- *Seed(entropy)*: Take in some initial truly random entropy and initialize the pRNG's internal state.
- *Reseed(entropy)*: Take in some additional truly random entropy, updating the pRNG's internal state as needed.
- *Generate(n)*: Generate nn pseudorandom bits, updating the internal state as needed. Some pRNGs also support adding additional entropy directly during this step.

True random number generators (TRNGs): TRNGs are not algorithmic, but instead they are systems, which extract randomness from non-algorithmic random phenomena. These phenomena may be temperature fluctuations, radioactive decay, ambient radio noise, hard disk access times, or user interactions with the PC. Since the phenomena used are

intrinsically unpredictable, TRNGs produce real random data instead of just randomlooking periodic sequences. The behavior of a TRNG is not defined by a mathematical formula, which is the case of DRNGs. Since the quality of generated random sequence depends on physical properties, the output sequence may exhibit some statistical defects such as bias. TRNGs are in general slower than DRNGs.‘ Based on the source used, they can be further divided to: — Physical TRNG (PTRNG) uses physical noise on electron level present in all semiconductors. A PTRNG is a physical device and uses physical noise. — Non-physical TRNG (NPTRNG) may not be a physical device, but instead a piece of software. It uses non-physical randomness source such as user interactions with an operating system. Unpredictability of deterministic random number generators is guaranteed computationally, while unpredictability of truly random generators is guaranteed by random physical phenomena and characterized by the entropy rate at generator output.

In particular, a good pseudorandom number algorithm generates bits that are *computationally indistinguishable* from true random bits–there is no efficient algorithm that would let an attacker distinguish between pseudorandom bits and truly random bits.

Pseudo randomness vs Randomness:

Pseudo randomness:

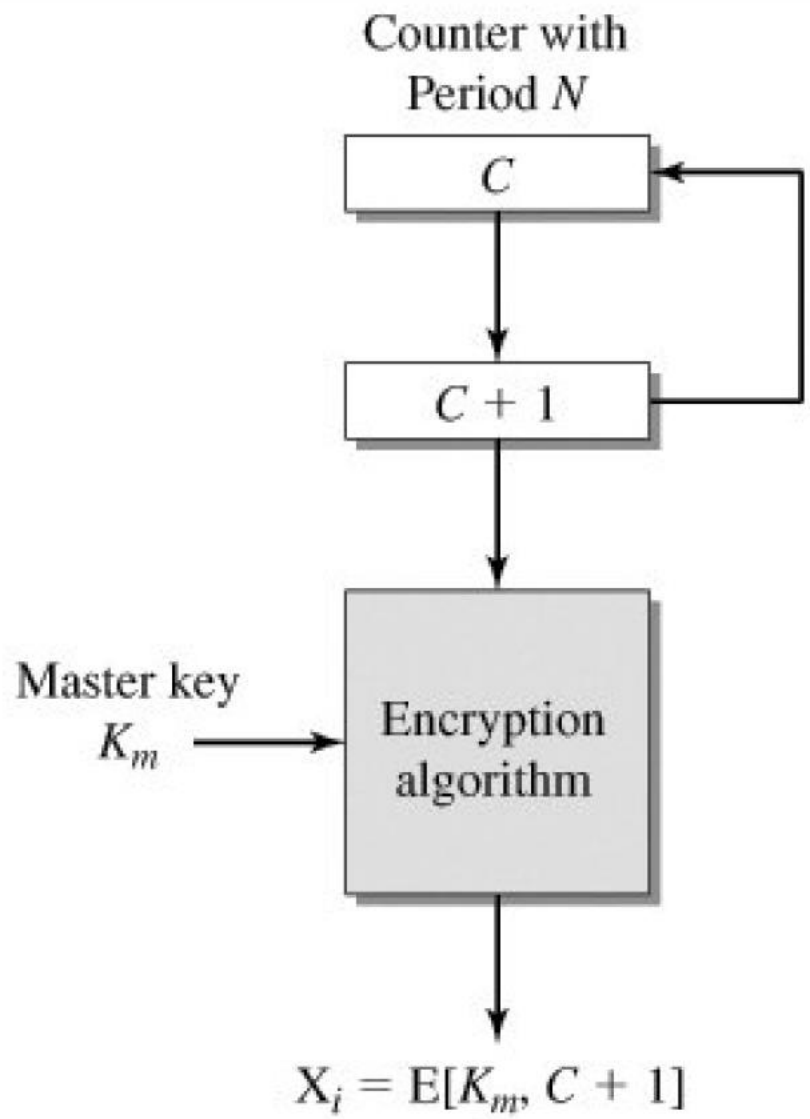
Produced by an algorithm that generates a series of bits that appear to be unpredictable:

- Usually maintain large internal state
- State changes on each output
- Initial state called seed

Randomness: A series of bits that are unpredictable

- Not produced by deterministic algorithm.

Following figure shows generation of a Pseudorandom Number from a Counter:



To strengthen the algorithm further, the input could be the output of a full-period PRNG rather than a simple counter.

A Sample Generator:

- Let generator function be $x_n = f\left(x_{n-1}, x_{n-2}, \ldots\right)$
- For example: Let $x_n = 5x_{n-1} + 1 \mod 16$
- Starting with $x_0 = 5$:

  $x_1 = 5(5) + 1 \bmod 16 = 26 \bmod 16 = 10$
- The first 32 numbers obtained by the above procedure 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5 10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.

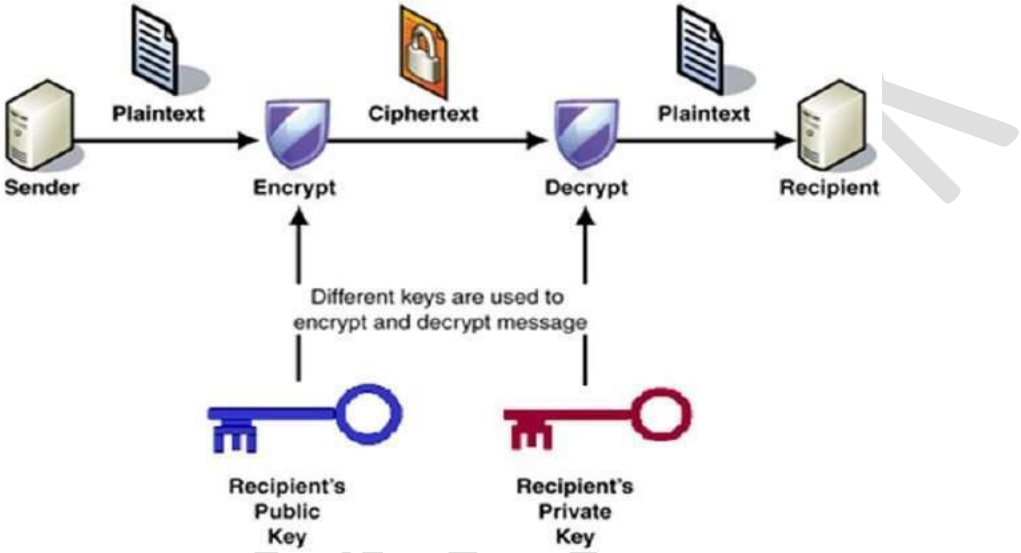
Unit-IV

Public Key Cryptography: Principles of Public-Key Cryptosystems:

Public-key cryptography is also known as *asymmetric-key* Cryptography. In this encryption and decryption are carried out using two different keys. The two keys in such a key pair are referred to as the public key and the private key.

The symmetric key was found to be non-practical due to challenges it faced for key management (i.e. key sharing). 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.

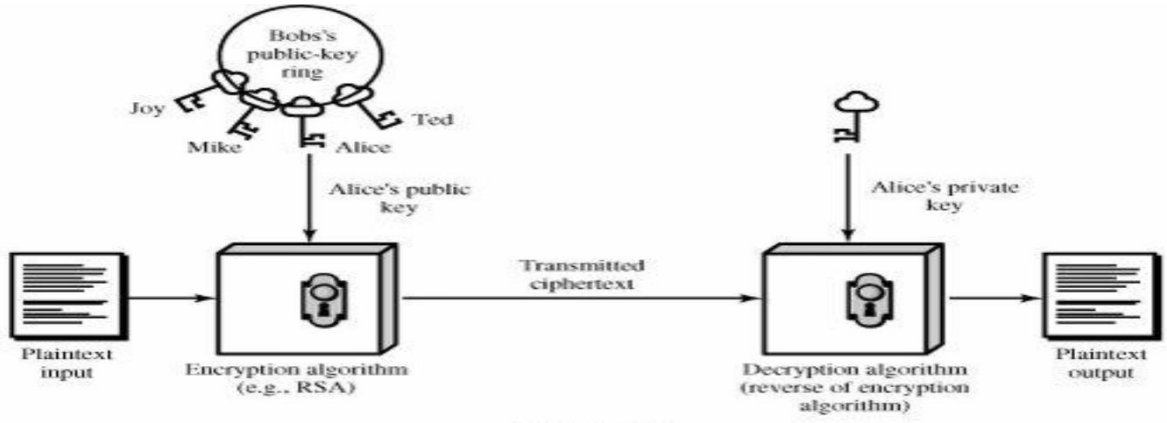
Though private and public keys are related mathematically, it is not be feasible to calculate the private key from the public key. In fact, intelligent part of any public-key cryptosystem is in designing a relationship between two keys.

A public-key encryption scheme has six ingredients

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations

performed by the algorithm depend on the public or private key that is provided as input.

- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.



Q). What is the difference between Symmetric Key Cryptographic and Asymmetric Key Cryptography?

Ans: <u>Symmetric Key Cryptography:</u>

In symmetric key cryptography, an individual key is used for both encryption and decryption. The sender needs the key to encrypt the plaintext and sends the cipher document to the receiver. The receiver used the similar key (or ruleset) to decrypt the message and recover the plaintext. Because an individual key is used for both functions, symmetric key cryptography is also known as symmetric encryption.

Symmetric key cryptography schemes are usually categorized such as stream ciphers or block ciphers. Stream ciphers works on a single bit (byte or computer word) at a time and execute some form of feedback structure so that the key is constantly changing.

<u>Asymmetric cryptography:</u>

Asymmetric cryptography uses two keys for encryption and decryption. It depends on the technique of public and private keys. A public key, which is interchanged between higher than one user. Data is decrypted by a private key, which is not transformed. It is slower but more secure. The public key used in this encryption technique is applicable to everyone, but the private key used in it is not revealed.

In asymmetric encryption, a message that is encrypted utilizing a public key can be decrypted by a private key, while if the message is encrypted by a private key can be decrypted by

utilizing the public key. Asymmetric encryption is broadly used in day to- day communication channels, particularly on the internet.

Let us see the comparison between Symmetric Key Cryptography and Asymmetric Key Cryptography.

Symmetric Key Cryptography	**Asymmetric Key Cryptography**
There is only one key (symmetric key) is used, and the similar key can be used to encrypt and decrypt the message.	There are two different cryptographic keys (asymmetric keys), known as the public and the private keys, are used for encryption and decryption.
It is effective as this technique is recommended for high amounts of text.	It is inefficient as this approach is used only for short messages.
Symmetric encryption is generally used to transmit bulk information.	It is generally used in smaller transactions. It is used for making a secure connection channel before transferring the actual information.
Symmetric key cryptography is also known as secret-key cryptography or private key cryptography.	Asymmetric key cryptography is also known as public-key cryptography or a conventional cryptographic system.
Symmetric key cryptography uses fewer resources as compared to asymmetric key cryptography.	Asymmetric key cryptography uses more resources as compared to symmetric key cryptography.
The length of the keys used is frequently 128 or 256 bits, based on the security need.	The length of the keys is much higher, such as the recommended RSA key size is 2048 bits or higher.

RSA Algorithm:

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 co-prime.
- **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 $n = pq = 7 \times 13 = 91$.
- Select $e = 5$, which is a valid choice since there is no number that is common factor of 5 and $(p-1)(q-1) = 6 \times 12 = 72$, 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*.

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

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

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

Key Generation	
Select p, q	p and q both prime, $p \neq q$
Calculate $n = p \times q$	
Calculate $\phi(n) = (p - 1)(q - 1)$	
Select integer e	$\gcd(\phi(n), e) = 1; 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \pmod{\phi(n)}$
Public key	$PU = \{e, n\}$
Private key	$PR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \bmod n$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \bmod n$

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.

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.

DIFFIE- HELLMAN KEY EXCHANGE:

The first published public-key algorithm appeared in the seminal paper by Diffie and Hellman that defined public-key cryptography and is generally referred to as Diffie-Hellman key exchange. The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent encryption of messages. The algorithm itself is limited to the exchange of secret values.

- It is not an encryption algorithm
- It Exchange Secret/Symmetric key between end users
- It use Asymmetric encryption

Global Public Elements	
q	prime number
α	$\alpha < q$ *and* α a primitive root of q

User A Key Generation	
Select private X_A	$X_A < q$
Calculate public Y_A	$Y_A = \alpha^{X_A} \bmod q$

User B Key Generation	
Select private X_B	$X_B < q$
Calculate public Y_B	$Y_B = \alpha^{X_B} \bmod q$

Calculation of Secret Key by User A

$$K = (Y_B)^{X_A} \bmod q$$

Calculation of Secret Key by User B

$$K = (Y_A)^{X_B} \bmod q$$

i. Assume X_A (Private Key of user A) $X_A < q$

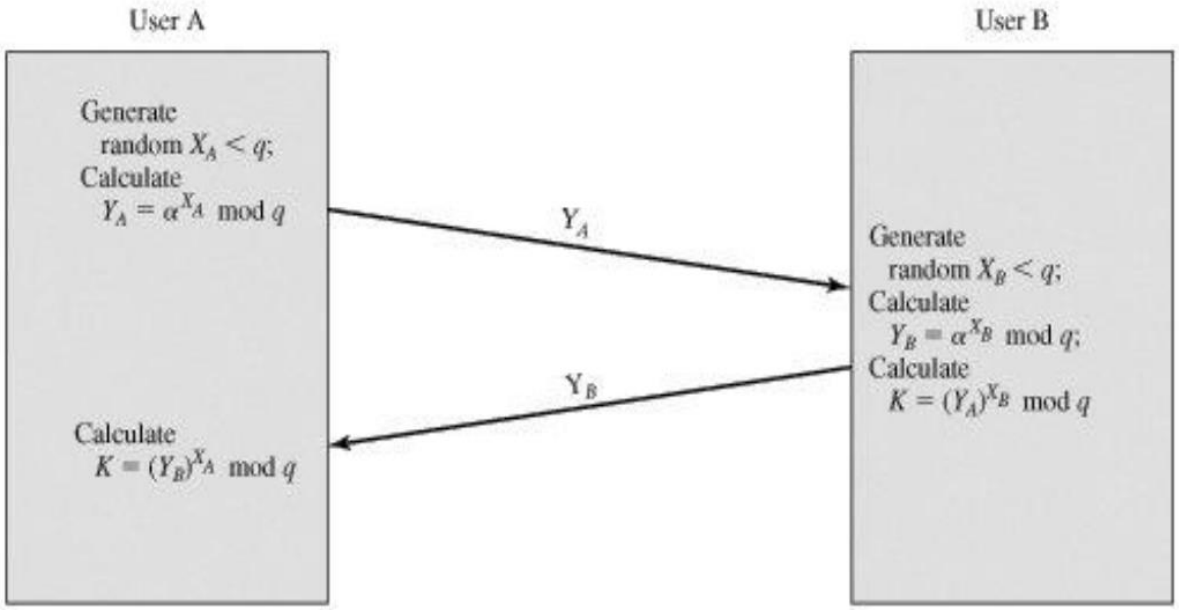
ii. Calculate Y_A (Public key of user A) $Y_A = \alpha^{X_A} \bmod q$

iii. Assume X_B (Private Key of user B) $X_B < q$

iv. Calculate Y_B (Public key of user B) $Y_B = \alpha^{X_B} \bmod q$

v. $K = (Y_B)^{X_A} \bmod q$ for User A

vi. $K = (Y_A)^{X_B} \bmod q$ for User B



Example:

i. $q = 11,\ \alpha = 2$

ii. α is a primitive root of p, that is $\alpha \bmod p$, $\alpha^2 \bmod p$, $\alpha^3 \bmod p$ ..., $\alpha^{p-1} \bmod p$ is $1, 2, 3, \ldots, p-1$.

iii. Select $X_A = 8$

$Y_A = \alpha^{X_A} \bmod q$

$Y_A = 2^8 \bmod 11$

$Y_A = 3$

iv. Select $X_B = 4$

$Y_B = \alpha^{X_B} \bmod q$

$Y_B = 2^4 \bmod 11$

$Y_B = 5$

v. For user A

$K = (Y_B)^{X_A} \bmod q$

$K = (5)^8 \bmod 11$

$K = 4$

vi. For user B

$K = (Y_A)^{X_B} \bmod q$

$K = (3)^4 \bmod 11$

$K = 4$

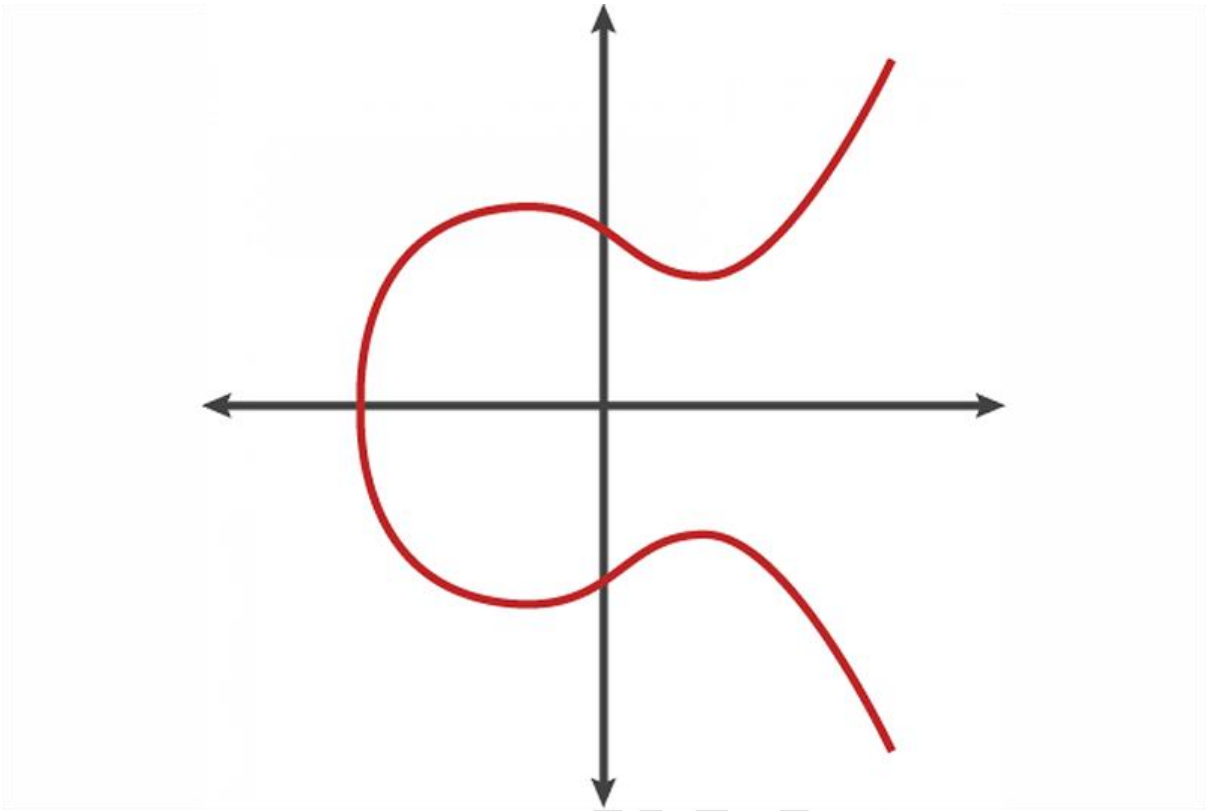
User A and B have a same Key value.

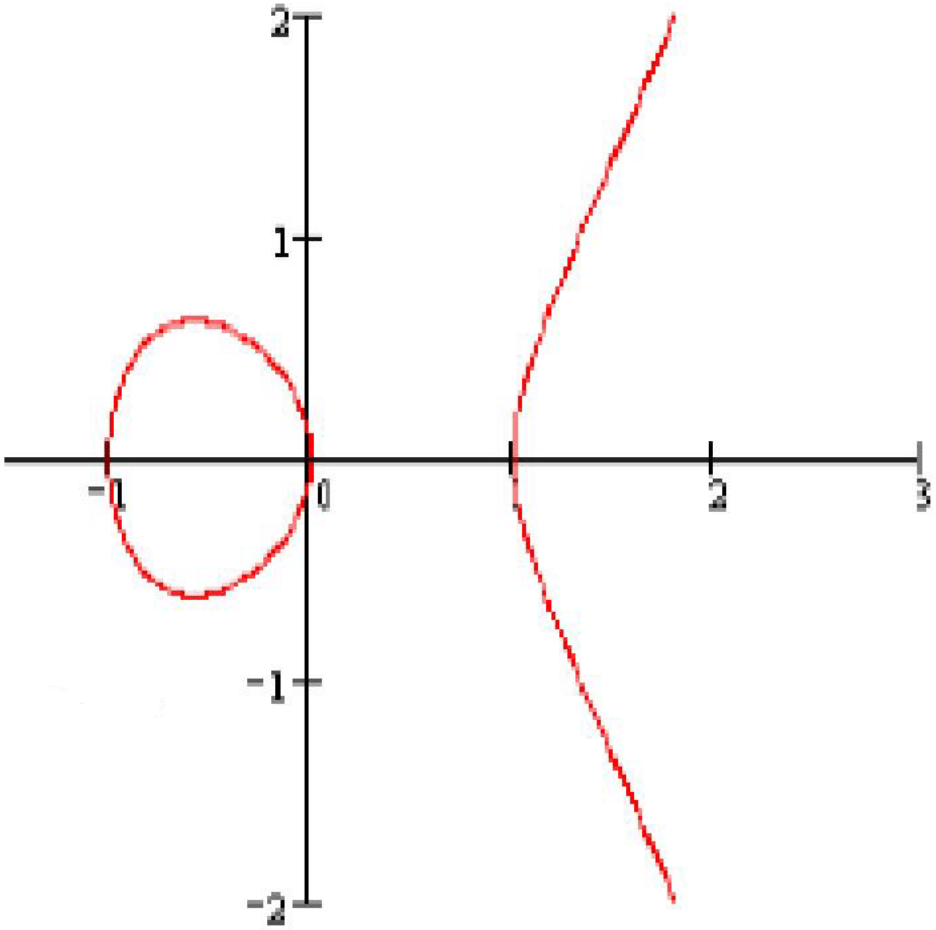
What is Elliptic Curve Cryptography(ECC)?:

Elliptic curve cryptography is used to implement public key cryptography. It was discovered by Victor Miller of IBM and Neil Koblitz of the University of Washington in the year 1985. ECC popularly used an acronym for Elliptic Curve Cryptography. It is based on the latest mathematics and delivers a relatively more secure foundation than the first generation public key cryptography systems for example RSA. ECC requires a smaller key as compared to non-ECC cryptography to provide equivalent security (a 256-bit ECC security has equivalent security attained by 3072-bit RSA cryptography).

Elliptic Curves:

In general, an elliptic curve looks like as shown below. Elliptic curves can intersect almost 3 points when a straight line is drawn intersecting the curve. As we can see, the elliptic curve is symmetric about the x-axis. This property plays a key role in the algorithm.





Elliptic Curve Cryptography (ECC) consists of four steps:

Step 1): Encode a plain text message M as a point P_M on the curve $E_p(a,b)$ where p is a prime number or an integer of the form 2^m where m is an positive integer. Next, pick a generator point $G=(x_1,y_1)$ in $E_p(a,b)$ whose order is a very large value n. The **order** of a point G on an elliptic curve is the smallest positive integer n such that $nG=0$.

Note: The order of a point on an elliptic curve is the order of that point as an element of the group defined on the curve. Definition. An element ***a*** of a group (i.e. group means it will satisfy closure, Associative, Identity and Inverse with respect to a binary operation) is said to have order ***n*** if ***n*** is a least positive integer such that ***na=0***.

Step 2): **Establish the public key and the private key:**

A key exchange between users A and B can be accomplished as follows:

1. A selects an integer n_A less than n. This is A's private key. A then generates a public key $P_A=n_AG\in E_p(a,b)$.
2. B similarly selects a private key n_B less than n and computes a public key $P_B=n_BG\in E_p(a,b)$.

3. A generates the secret key $K_A = n_A P_B$ and B generates the secret key $K_B = n_B P_A$. Always we get $K_A = K_B$.

Step 3) **Encryption:**

To encrypt and send a message P_M to B, A chooses a random positive integer $k \in \{1, 2, \ldots, (p-1)\}$ and produces the ciphertext C_M consisting of the pair of points: $C_M = (C_1, C_2) = (kG,\ P_M + kP_B)$. Note that A will use B's public key P_B to encrypt the message P_M.

(similarly, to encrypt and send a message P_M to A, B chooses a random positive integer $k \in \{1, 2, \ldots, (p-1)\}$ and produces the ciphertext C_M consisting of the pair of points: $C_M = (C_1, C_2) = (kG,\ P_M + kP_A)$. Note that B will use A's public key P_A to encrypt the message P_M.)

Step 4) **Decryption:** To decrypt the ciphertext which is shared by A, B multiplies the first point in the pair by B's private key and subtracts the result from the second point: $P_M = C_2 - n_B C_1$.

That is, $\text{R.H.S} = C_2 - n_B C_1$

$$= (P_M + kP_B) - n_B (kG)$$
$$= (P_M + kP_B) - k(n_B G)$$
$$= (P_M + kP_B) - k(P_B), \quad \text{since } P_B = n_B G$$
$$= P_M$$
$$= \text{Plain text} = \text{L.H.S}$$

(Similarly, to decrypt the ciphertext which is shared by B, A multiplies the first point in the pair by A's private key and subtracts the result from the second point: $P_M = C_2 - n_B C_1$.)

Elliptic curve cryptography (ECC) comparison with RSA algorithm

Symmetric Key Size (bits)	RSA and Diffie-Hellman Key Size (bits)	Elliptic Curve Key Size (bits)
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

Uses:

- Websites make extensive use of ECC to secure customers hypertext transfer protocol connections.
- It is used for encryption by combining the key agreement with a symmetric encryption scheme.
- It is also used in several integer factorization algorithms like Lenstra elliptic-curve factorization.
- Time stamping uses an encryption model called a blind signature scheme. It is possible using Elliptic Curve Cryptography.

Probabilistic encryption

Probabilistic encryption is the use of randomness in an encryption algorithm, so that when encrypting the same message several times it will, in general, yield different ciphertexts. The term 'probabilistic encryption' is typically used in reference to public key encryption algorithms.

Definition of Semantic security: Semantic security is a notion to describe the security of an encryption scheme. An adversary is allowed to choose between two plaintexts, m_0 and m_1, and he receives an encryption of either one of the plaintexts. An encryption scheme is semantically secure, if an adversary cannot guess with better probability than 1/2 whether the given ciphertext is an encryption of message m_0 or m_1.

Goldwasser-Micali Encryption:

The Goldwasser–Micali (GM) cryptosystem is an asymmetric key encryption algorithm developed by Shafi Goldwasser and Silvio Micali in 1982. GM has the distinction of being the first probabilistic public-key encryption scheme which is provably secure under standard cryptographic assumptions. However, it is not an efficient cryptosystem, as ciphertexts may be several hundred times larger than the initial plaintext. To prove the security properties of the cryptosystem, Goldwasser and Micali proposed the widely used definition of semantic security. Goldwasser–Micali consists of three algorithms: 1) a probabilistic key generation algorithm which produces a public and a private key, 2) a probabilistic encryption algorithm, and 3) a deterministic decryption algorithm.

1) **Key Generation**

- Choose two large primes p and q (of bit size > 512) and let $n = pq$.
- Generate random integers a, b with $\left(\frac{a}{p}\right) = \left(\frac{b}{q}\right) = -1$ (i.e. Legendre symbol of *a* with *p* and Legendre symbol of *b* with *q* is -1, clearly *a* and *b* are respectively quadratic non residues modulo *p* and *q*).

- Use Chinese Remainder Theorem (CRT) to generate $x \pmod{n}$ with $x \equiv a \pmod{p}$ and $x \equiv b \pmod{q}$.
- The Public key is (n, x) and the private key is p.

2) Encryption:

- The input is the r-bit plaintext message $m_1 m_2 \ldots m_r$.
- For each $i = 1, 2, \ldots, r$, choose $a_i \in Z \bigcap (1, n)$ where Z is an integer set and compute $C_i = x^{m_i} a_i^2 \pmod{n}$.
- The ciphertext message is the r-tuple $(C_1, C_2, \ldots, C_r) \in \left(Z_n^*\right)^r$ where $Z_n^* = \{1, 2, \ldots, (n-1)\}$.

3) Decryption:

- **For** $i = 1, 2, \ldots, r$, take

$$m_i = 0 \text{ if } \left(\frac{C_i}{p}\right) = 1, \text{ or}$$

$$m_i = 1 \text{ if } \left(\frac{C_i}{p}\right) = -1,$$

where $\left(\frac{C_i}{p}\right)$ represents a Legendre symbol of C_i with p.

Remarks:

- **Probabilistic encryption:** The ciphertext C_i depends on the choice of a_i.
- **Message expansion:** An r-bit plaintext message generates an $(r|n|)$-bit ciphertext message.
- Without the knowledge of p (the private key), we do not know how to determine whether C_i is a quadratic residue or not modulo n.

Example:

Key generation

- Take $p = 653$ and $q = 751$, so $n = pq = 490403$.
- Take $a = 159$ and $b = 432$, so $x \equiv 313599 \pmod{n}$.
- The public-key is $(490403, 313599)$ and the private key is 653.

Encryption

- Let us encrypt the 3-bit message $m_1m_2m_3 = 101$.
- Choose $a_1 = 356217$ and compute $c_1 \equiv xa_1^2 \equiv 398732 \pmod{n}$.
- Choose $a_2 = 159819$ and compute $c_2 \equiv x0\ a_2^2 \equiv 453312 \pmod{n}$.
- Choose $a_3 = 482474$ and compute $c_3 \equiv xa_3^2 \equiv 12380 \pmod{n}$.

Decryption

- $\left(\frac{398732}{p}\right) = -1$, so $m_1 = 1$.
- $\left(\frac{453312}{p}\right) = 1$, so $m_2 = 0$.
- $\left(\frac{12380}{p}\right) = -1$, so $m_3 = 1$.

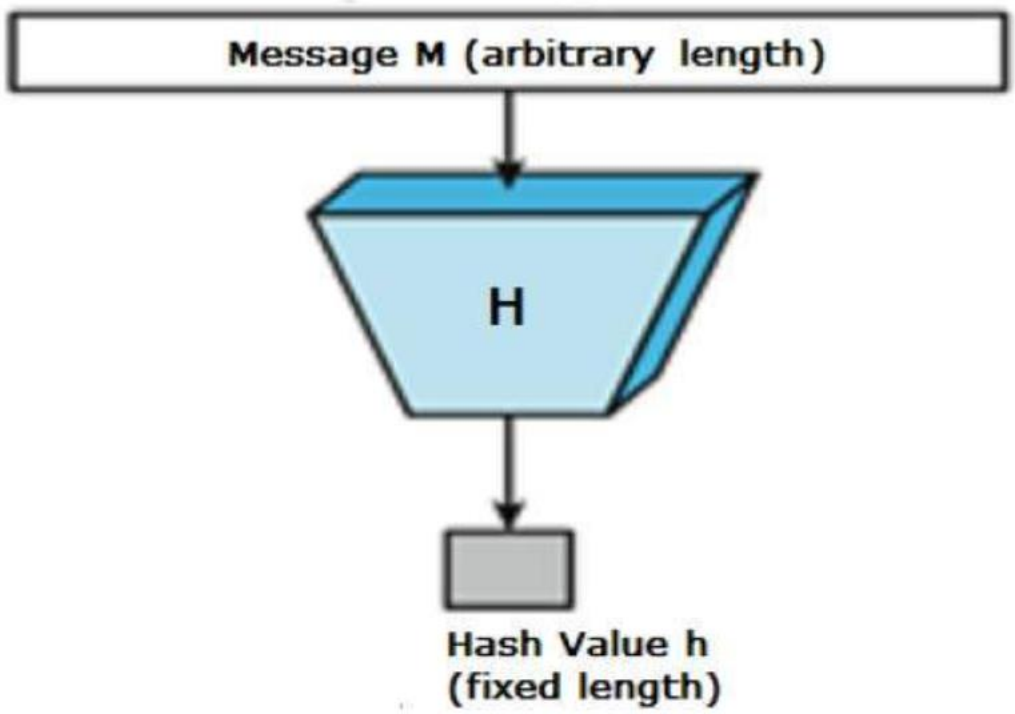
Unit-5

Hash functions

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.**

Applications of Hash functions in cryptography

The most versatile cryptographic algorithm is the cryptographic hash function. It is used in a wide variety of security applications and Internet protocols. The following are various applications where it is employed.

1. Message Authentication
2. Digital Signatures
3. One way password file
4. Intrusion detection and prevention

1. Message Authentication:

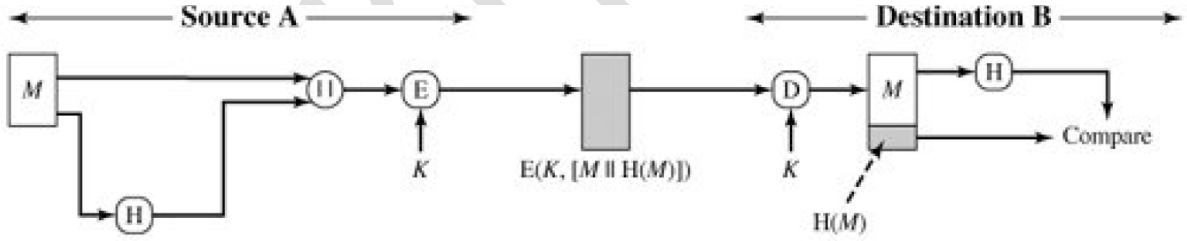
- Message authentication is a mechanism or service used to verify the integrity of a message.

- Message authentication assures that data received are exactly as sent (i.e., contain no modifications, insertion, deletion or reply).
- When a hash function is used to provide message authentication, the hash function value is often referred to as a message digest.
- The essence of the use of a hash function for message integrity is as follows.
    - The sender computes a hash value as a function of the bits in the message and transmits both the hash value and the message.
    - The receiver performs the same hash calculation on the message bits and compares this value with the incoming hash value.
    - If there is a mismatch, the receiver knows that the message (or possibly the hash value) has been altered.
    - The hash value must be transmitted in a secure fashion. That is, the hash value must be protected so that if an adversary alters or replaces the message, it is not feasible for adversary to also alter the hash value to fool the receiver.

In the following we discuss different ways of using a hash code to provide message authentication:

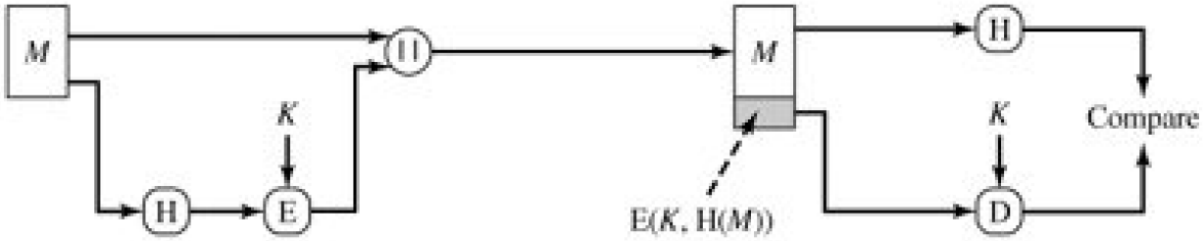
Method 1:

The message plus concatenated hash code is encrypted using symmetric encryption. This is identical in structure to the internal error control strategy shown in the following figure. The same line of reasoning applies: Because only A and B share the secret key, the message must have come from A and has not been altered. The hash code provides the structure or redundancy required to achieve authentication. Because encryption is applied to the entire message plus hash code, confidentiality is also provided.



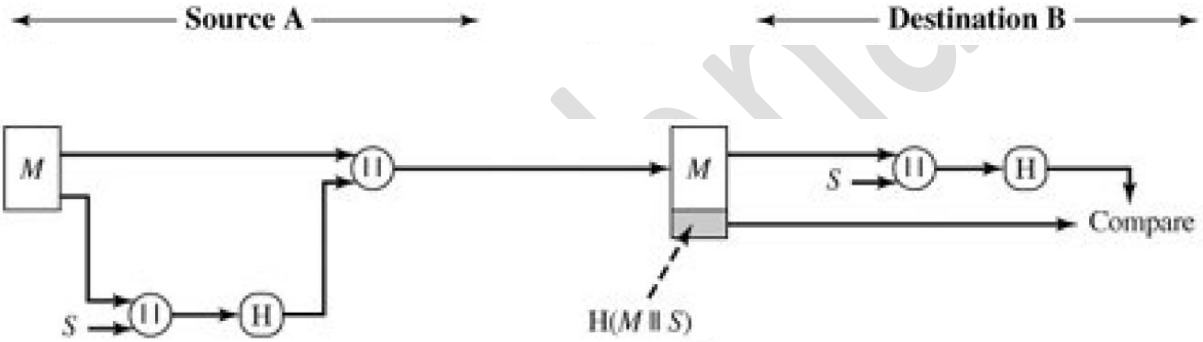
Method 2:

Only the hash code is encrypted, using symmetric encryption. This reduces the processing burden for those applications that do not require confidentiality. Note that the combination of hashing and encryption results in an overall function. That is, E(*K*, H(*M*)) is a function of a variable-length message *M* and a secret key *K*, and it produces a fixed-size output that is secure against an opponent who does not know the secret key.



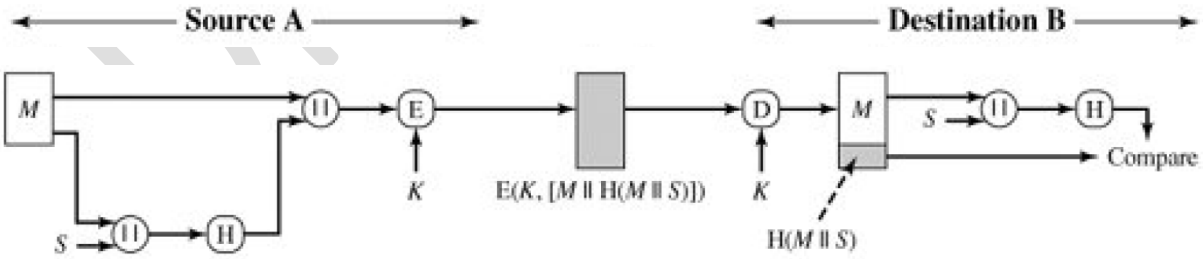
Method 3:

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



Method 4:

Confidentiality can be added to the approach of method 4 by encrypting the entire message plus the hash code.



2. **Digital Signatures:**

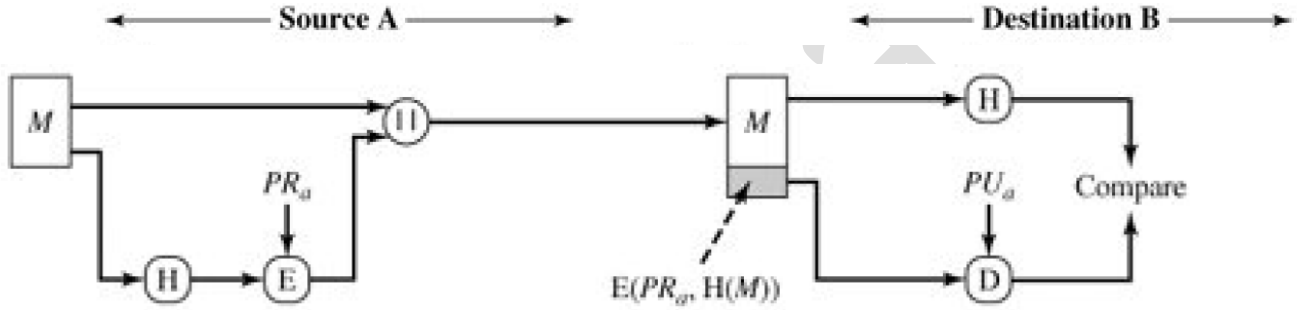
   Another important application, which is similar to the message authentication application, is the digital signature. A digital signature is an authentication mechanism that enables the creator of a message to attach a code that acts as a

signature. The signature is formed by taking the hash of the message and encrypting the message with the creator's private key. The signature guarantees the source and integrity of the message.

In the case of the digital signature, the hash value of a message is encrypted with a user's private key. Anyone who knows the user's public key can verify the integrity of the message that is associated with the digital signature. In this case, an attacker who wishes to alter the message would need to know the user's private key. Following figures illustrates, in a simplified fashion, how a hash code is used to provide a digital signature.

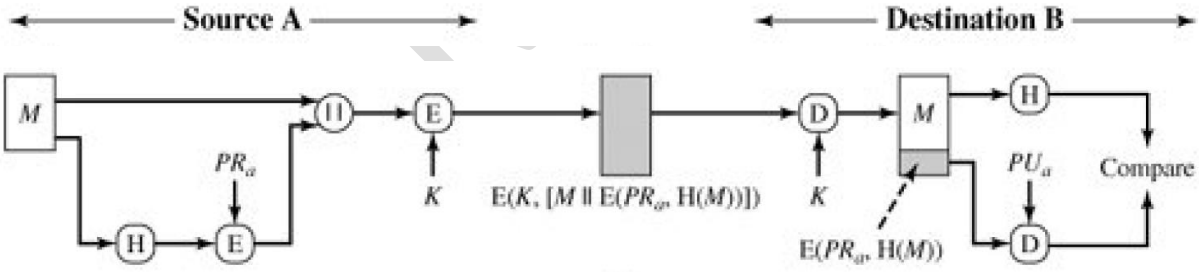
Method 1:

Only the hash code is encrypted, using public-key encryption and using the sender's private key. This provides authentication. It also provides a digital signature, because only the sender could have produced the encrypted hash code. In fact, this is the essence of the digital signature technique.



Method 2:

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



3. <u>One way password file:</u>

A scheme in which the hash of the password is stored by the system rather than the password itself is the one-way password file system.

That is, the system stores only the hash value of a function based on the user's password. When the user presents a password, the system transforms that password and compares it with the stored hash value. In practice, the system usually performs a one-way transformation (not reversible) in which the password is used to generate a key for the one-way function and in which a fixed-length output is produced.

4. **Intrusion detection and prevention:**

- Hash functions can be used for intrusion detection and virus detection.
- Store H(F) for each file on a system and secure the hash values.
- One can later determine if a file has been modified by recomputing H(F).
- An intruder would need to change F without changing H(F).

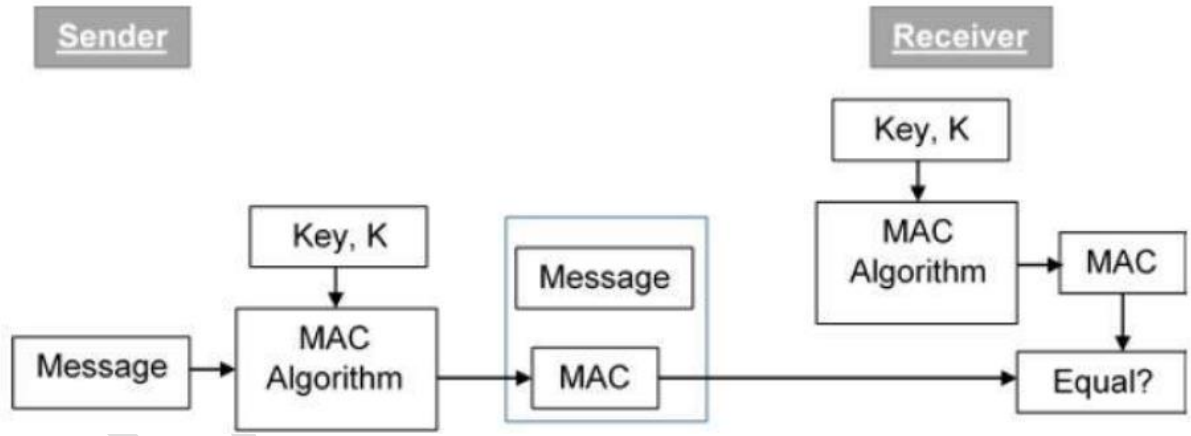
Message Authentication Code (MAC)

An authentication technique which involves the use of a secret key to generate a small fixed-size block of data, known as a **cryptographic checksum** or MAC, that is appended to the message.

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.

_____XXXXXXXXXXXX_____XXXXXXXXXX_____

Q. Explain different methods to provide authentication using MAC function (Long answer question).

Q. Draw a neat diagram where authentication is tied to plaintext w.r.t. MAC. (for short answer question, just draw the figure b).

Q. Draw a neat diagram where authentication is tied to ciphertext w.r.t. MAC. (short answer Question, just draw the figure c).

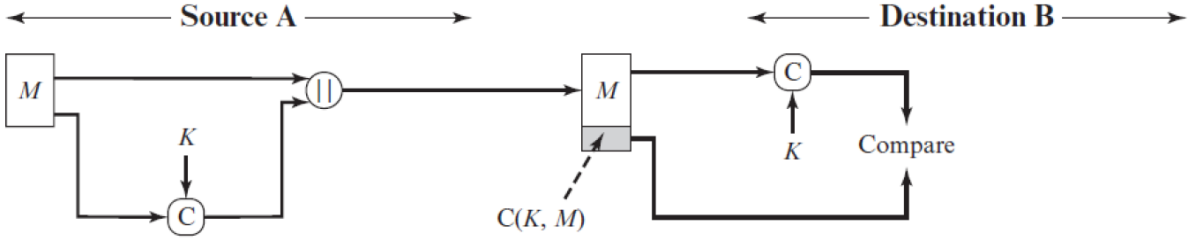
Ans: The MAC technique assumes that two communicating parties, say A and B, share a common secret key *K*.

- When A has a message to send to B, it calculates the MAC as a function of the message and the key:

  "MAC = C(K, M)"

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

- The message plus (concateness) MAC are transmitted to the intended recipient.
- The recipient performs the same calculation on the received message, using the same secret key, to generate a new MAC.
- The received MAC is compared to the calculated MAC (see Figure a).



(a) Message authentication

If we assume that only the receiver and the sender know the identity of the secret key, and if the received MAC matches the calculated MAC, then

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

- A MAC function is similar to encryption. One difference is that the MAC algorithm need not be reversible, as it must be for decryption.
- In general, the MAC function is a many-to-one function.

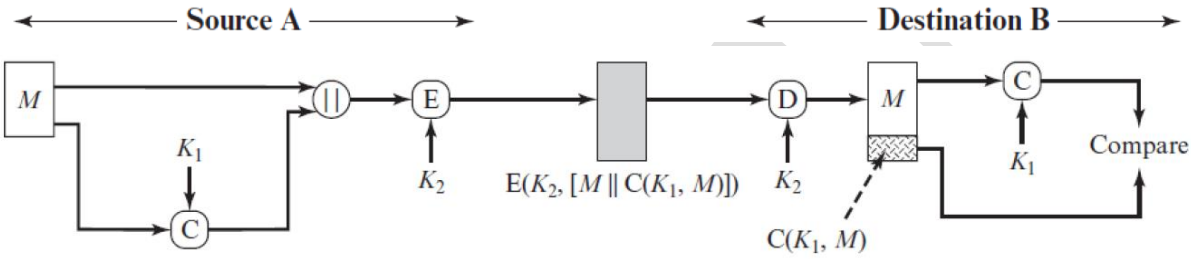
- The domain of the function consists of messages of some arbitrary length, whereas the range consists of all possible MACs and all possible keys.

- If an *n*-bit MAC is used, then there are 2*n* possible MACs, whereas there are *N* possible messages with $N >> 2n$. Furthermore, with a *k*-bit key, there are 2*k* possible keys.

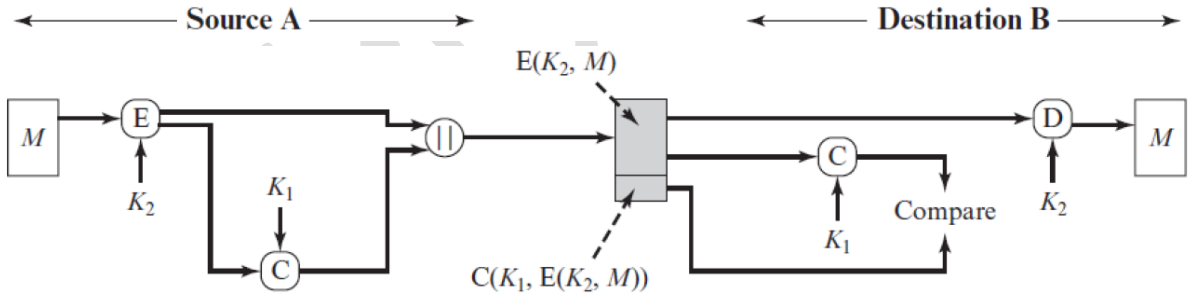
- For example, suppose that we are using 100-bit messages and a 10-bit MAC. Then, there are a total of 2100 different messages but only 210 different MACs. So, on average, each MAC value is generated by a total of 2100/210 = 290 different messages. If a 5-bit key is used, then there are 25 = 32 different mappings from the set of messages to the set of MAC values.

- The process depicted in Figure (a) provides authentication but not confidentiality, because the message as a whole is transmitted in the clear.

- Confidentiality can be provided by performing message encryption either after (Figure b) or before (Figure c) the MAC algorithm.



(b) Message authentication and confidentiality; authentication tied to plaintext



(c) Message authentication and confidentiality; authentication tied to ciphertext

- In both these cases, two separate keys are needed, each of which is shared by the sender and the receiver. In the first case, the MAC is calculated with the message as input and is then concatenated to the message.
- The entire block is then encrypted. In the second case, the message is encrypted first.
- Then the MAC is calculated using the resulting ciphertext and is concatenated to the ciphertext to form the transmitted block.
- Typically, it is preferable to tie the authentication directly to the plaintext, so the method of Figure b is used.

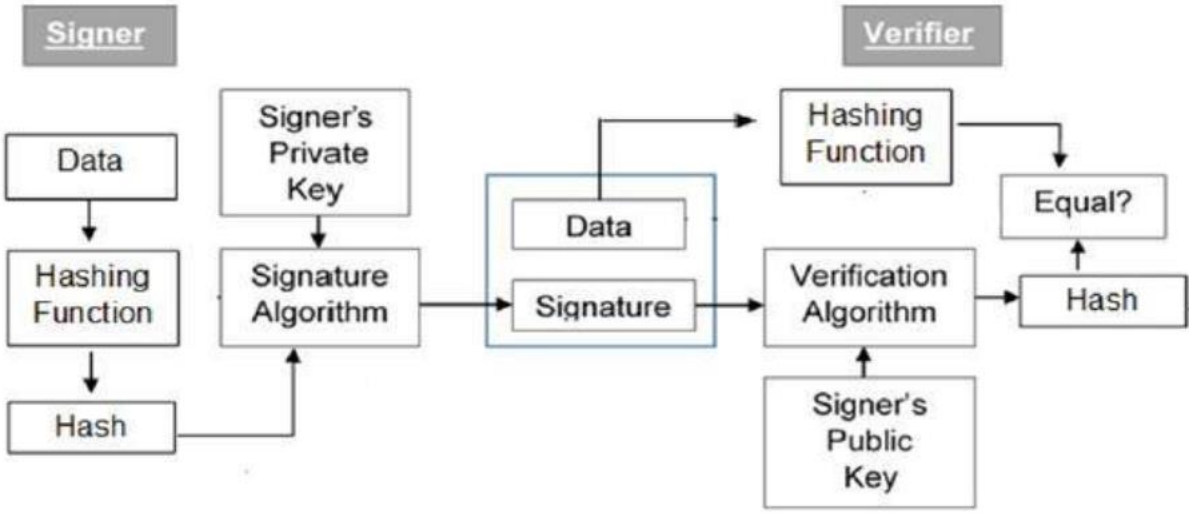
Digital Signature Standard (DSS) / Digital Signature Algorithm (DSA) :

DSA stand for **Digital Signature Algorithm**. It is used for digital signature and its

verification. It is based on mathematical concept of modular exponentiation and discrete logarithm. It was developed by **National Institute of Standards and Technology (NIST)** in1991.

Digital Signature Algorithm (DSA) is one of the Federal Information Processing Standard for making digital signatures depends on the mathematical concept to cryptograph the signature digitally in this algorithm. **Digital signature** is a cryptographic value that is calculated from the data and a secret key known only by the signer or the person whose signature is that.

In fact, in the real world, the receiver of message needs assurance that the message belongs to the sender and he should not be able to hack the origination of that message for misuse or anything. Their requirement is very crucial in business applications or any other things since the likelihood of a dispute over exchanged data is very high to secure that data.



Explanation of the block diagram:

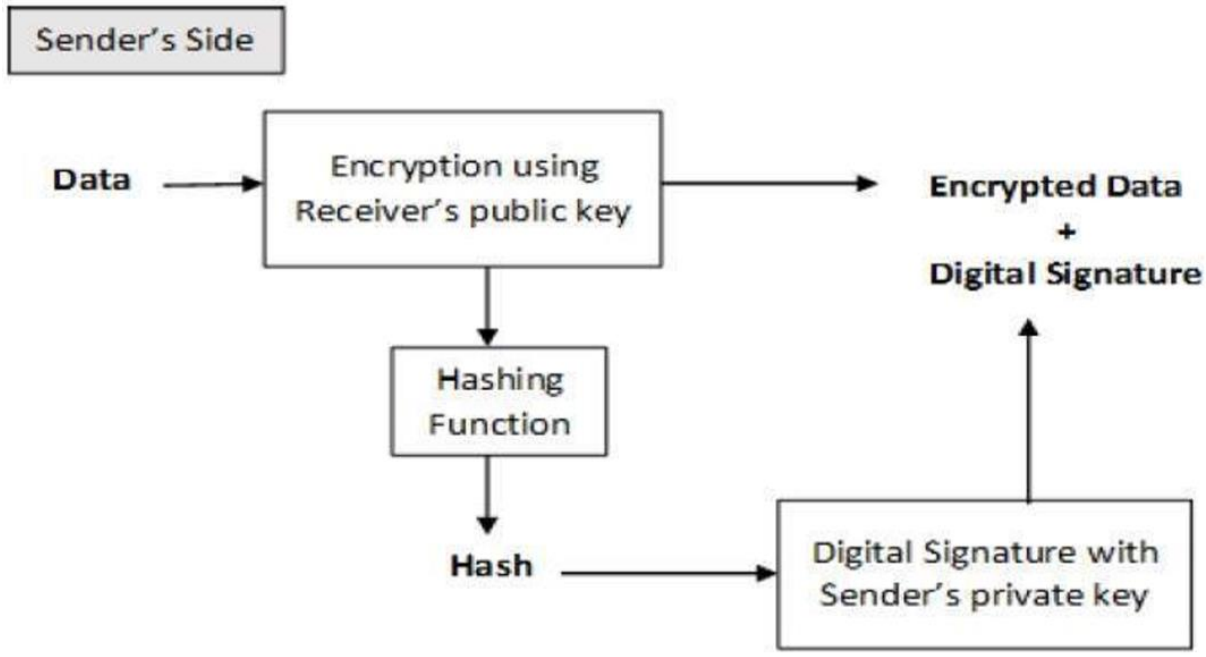
- Firstly, each person adopting this scheme has a public-private key pair in cryptography.
- The key pairs used for encryption or decryption and signing, or verifying are different for every signature. Here, the private key used for signing is referred to as the signature key and the public key as the verification key in this algorithm.
- Then, people take the signer feeds data to the hash function and generates a hash of data of that message.
- Now, the Hash value and signature key are then fed to the signature algorithm which produces the digital signature on a given hash of that message. This

signature is appended to the data and then both are sent to the verifier to secure that message.

- Then, the verifier feeds the digital signature and the verification key into the verification algorithm in this **DSA**. Thus, the verification algorithm gives some value as output as a ciphertext.
- Thus, the verifier also runs the same hash function on received data to generate hash value in this algorithm.
- Now, for verification, the signature, this hash value, and output of verification algorithm are compared with each variable. Based on the comparison result, the verifier decides whether the digital signature is valid for this or invalid.
- Therefore, the digital signature is generated by the 'private' key of the signer and no one else can have this key to secure the data, the signer cannot repudiate signing the data in the future to secure that data by the cryptography.

Encryption with Digital Signature:

It is desirable to exchange encrypted messages than plaintext to achieve confidentiality in cryptography. In fact, in the public key encryption scheme, this is a public or encryption key of the sender is available in the open domain, and hence anyone can spoof his identity and send an encrypted message to the receiver in this algorithm.



DSA algorithm involves four operations:

1. Key Generation
2. Key Distribution
3. Signing
4. Signature Verification

1. Key Generation:

- Choose a prime number p where $2^{L-1} < p < 2^L$, L=512 to 1024 bits and L is a multiple of 64, i.e, bit length of between 512 and 1024 bits increments of 64 bits.
- Choose another prime number q (where $2^{159} < q < 2^{160}$) which is a 160-bit prime factor of (p-1), i.e. (p-1) mod q =0.
- $g = h^{\frac{(p-1)}{q}} \bmod p$, where h is any number less than $(p-1)$ with $h^{\frac{(p-1)}{q}} \pmod p > 1, i.e, 1 < h < p-1$.

2. Key Distribution:
   - Choose an integer x, such that $0 < x < q$.
   - Compute $y = g^x \bmod p$.
   - Thus, Package the public key as {p,q,g,y} is this.
   - And, Package the private key as {p,q,g,x} is this.
3. Signing:
   - Firstly, generate the message digest H(M)=h, using a hash algorithm like SHA1, where M=Message.
   - Then, generate a random number k, such that 0 < k < q.
   - Then, Compute $r = \left[g^k \pmod p\right] \pmod q$. If r = 0, select a different k.
   - Then, Compute $S = \left[k^{-1}\left(H(M) + xr\right)\right] \bmod q$. If s = 0, select a different k.
   - Thus, Package the digital signature as {r, S}.
4. Signature Verification:
   - Firstly, Generate the message digest H(M)=h, using the same hash algorithm.
   - Then, Compute $W = S^{-1} \pmod q$, W is called the modular multiplicative inverse of S modulo q in this.
   - Then, Compute $U1 = \left[H(M)W\right] \bmod q$.
   - And Compute $U2 = rW \pmod q$.
   - Then, Compute $V = \left[\left(g^{U1} y^{U2}\right) \bmod p\right] \bmod q$.
   - Wherever, if $V = r$, the digital signature is valid.

Importance of Digital Signature:

Therefore, all cryptographic analysis of the digital signature using public-key cryptography is considered a very important or main and useful tool to achieve information security in cryptography in cryptoanalysis.

Thus, apart from the ability to provide non-repudiation of the message, the digital signature also provides message authentication and data integrity in cryptography.

This is achieved by the digital signature are:

- **Message authentication**: Therefore, when the verifier validates the digital signature using the public key of a sender, he is assured that signature has been created only by a sender who possesses the corresponding secret private key and no one else does by this algorithm.
- **Data Integrity**: In fact, in this case, an attacker has access to the data and modifies it, the digital signature verification at the receiver end fails in this algorithm, Thus, the hash of modified data and the output provided by the verification algorithm will not match the signature by this algorithm. Now, the receiver can safely deny the message assuming that data integrity has been breached for this algorithm.
- **Non-repudiation**: Hence, it is just a number that only the signer knows the signature key, he can only create a unique signature on a given data of that message to change in cryptography. Thus, the receiver can present data and the digital signature to a third party as evidence if any dispute arises in the future to secure the data.

Q). List the type of attacks and forgeries in Digital Signatures.

Ans: **Types of Digital Signature Attacks :**

There are three types of attacks on Digital Signatures:

1. Chosen-message Attack
2. Known-message Attack
3. Key only Attack

Let us consider an example where C is the attacker and A is the victim whose message and signature are under attack.

1. **Chosen-message Attack :**

   The chosen attack method is of two types:

   i. **Generic chosen-method –** In this method C tricks A to digitally sign the messages that A does not intend to do and without the knowledge about A's public key.

   ii. **Direct chosen-method –** In this method C has the knowledge about A's public key and obtains A's signature on the messages and replaces the original message with the message C wants A to sign with having A's signature on them unchanged.

2. **Known-message Attack :**

   In the known message attack, C has few previous messages and signatures of A. Now C tries to forge the signature of A on to the documents that A does not intend to sign by using the brute force method by analysing the previous data to recreate the signature of A. This attack is similar to known-plain text attack in encryption.

3. **Key-only Attack :**

   In key-only attack, the public key of A is available to everyone and C makes use of this fact and try to recreate the signature of A and digitally sign the documents or messages that A does not intend to do. This would cause a great threat to authentication of the message which is non repudiated as A cannot deny signing it.

Forgeries in Digital Signatures.

The term forgery usually describes a message related attack against a cryptographic digital signature scheme. That is an attack trying to fabricate a digital signature for a message without having access to the respective signer's private signing key. The security requirement of *unforgeability* of digital signatures is also called nonrepudiation. There are three types of forgery in digital signatures.

i. **Universal forgery:** C finds an efficient signing algorithm that provides an equivalnt way of constructing signatures on arbitrary messages.
ii. **Selective forgery:** C forges a signature for a particular message chosen by C.
iii. **Existential forgery:** C forges a signature for at least one message. C has no control over the message. Consequently, this forgery may only be a minor nuisance to A.

_____XXXXXXXXXXX_____XXXXXXXXXXXXX_____

Digital Signature Requirements:

Based on the properties and attacks just discussed above, 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.

Difference between RSA algorithm and DSA :

RSA	DSA
It is a cryptosystem algorithm.	It is digital signature algorithm.
It is used for secure data transmission.	It is used for digital signature and its verification.
It was developed in 1977.	While it was developed in 1991.
It was developed by **Ron Rivest**, **Adi Shamir** and **Leonard Adleman**.	It was developed by **National Institute of Standards and Technology (NIST)**.
It uses mathematical concept of factorization of product of two large primes.	It uses modular exponentiation and discrete logarithm.

It is slower in key generation.	While it is faster in key generation as compared to RSA.
It in faster than DSA in encryption.	While it is slower in encryption.
It is slower in decryption.	While it is faster in decryption.
It is best suited for verification and encryption.	It is best suited for signing in and decryption.