

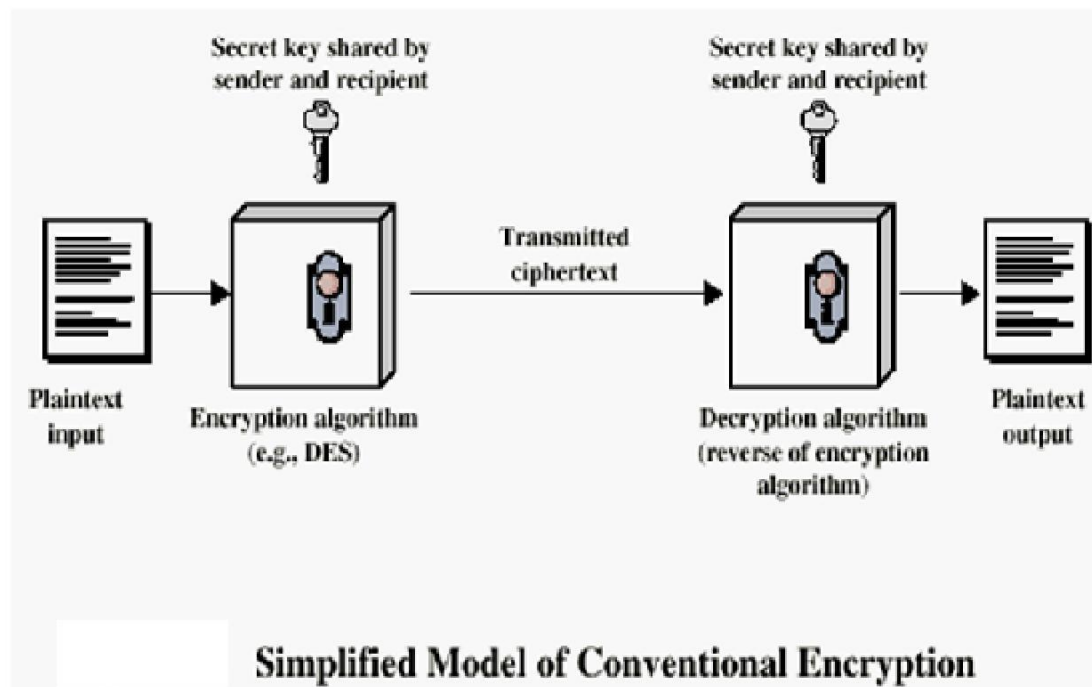
## Unit-2

CONVENTIONAL ENCRYPTION PRINCIPLES, CONVENTIONAL ENCRYPTION ALGORITHMS, CIPHER BLOCK MODES OF OPERATION, LOCATION OF ENCRYPTION DEVICES, KEY DISTRIBUTION APPROACHES OF MESSAGE AUTHENTICATION, SECURE HASH FUNCTIONS AND HMAC

# Conventional Encryption principles

A Symmetric encryption scheme has five ingredients

1. **Plain Text**: This is the original message or data which is fed into the algorithm as input.
2. **Encryption Algorithm**: This encryption algorithm performs various substitutions and transformations on the plain text.
3. **Secret Key**: The key is another input to the algorithm. The substitutions and transformations performed by algorithm depend on the key.



4. **Cipher Text**: This is the scrambled (unreadable) message which is output of the encryption algorithm. This cipher text is dependent on plaintext and secret key. For a given plaintext, two different keys produce two different cipher texts.
5. **Decryption Algorithm**: This is the reverse of encryption algorithm. It takes the cipher text and secret key as inputs and outputs the plain text.

Two main requirements are needed for secure use of conventional encryption:

- (i). A strong encryption algorithm is needed. It is desirable that the algorithm should be in such a way that, even the attacker who knows the algorithm and has access to one or more cipher texts would be unable to decipher the ciphertext or figure out the key.
- (ii). The secret key must be distributed among the sender and receiver in a very secured way. If in any way the key is discovered and with the knowledge of algorithm, all communication using this key is readable.

The important point is that the security of conventional encryption depends on the secrecy of the key, not the secrecy of the algorithm i.e. it is not necessary to keep the algorithm secret, but only the key is to be kept secret. This feature that algorithm need not be kept secret made it feasible for wide spread use and enabled manufacturers develop low cost chip implementation of data encryption algorithms. With the use of conventional algorithm, the principal security problem is maintaining the secrecy of the key.

## Cryptography

A cipher is a secret method of writing, as by code. **Cryptography**, in a very broad sense, is the study of techniques related to aspects of information security. Hence cryptography is concerned with the writing (ciphering or encoding) and deciphering (decoding) of messages in secret code. Cryptographic systems are classified along three independent dimensions:

### **1. The type of operations used for performing plaintext to ciphertext**

All the encryption algorithms make use of two general principles; substitution and transposition through which plaintext elements are rearranged. Important thing is that no information should be lost.

### **2. The number of keys used**

If single key is used by both sender and receiver, it is called symmetric, single-key, secret-key or conventional encryption. If sender and receiver each use a different key, then it is called asymmetric, two-key or public-key encryption.

### **3. The way in which plaintext is processed**

A block cipher process the input as blocks of elements and generated an output block for each input block. Stream cipher processes the input elements continuously, producing output one element at a time as it goes along.

Substitution: Method by which units of plaintext are replaced with ciphertext according to a regular system.

Transposition: Here, units of plaintext are rearranged in a different and usually quite complex order, but the units themselves are left unchanged.

## Cryptanalysis

The process of attempting to discover the plaintext or key is known as cryptanalysis. It is very difficult when only the ciphertext is available to the attacker as in some cases even the encryption algorithm is not known. The most common attack under these circumstances is brute-force approach of trying all the possible keys. This attack is made impractical when the key size is considerably large. The table below gives an idea on types of attacks on encrypted messages.

Type of Attack	Known to Cryptanalyst
Ciphertext only	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> </ul>
Known plaintext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• One or more plaintext-ciphertext pairs formed with the secret key</li> </ul>
Chosen plaintext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key</li> </ul>
Chosen ciphertext	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key</li> </ul>
Chosen text	<ul style="list-style-type: none"> <li>• Encryption algorithm</li> <li>• Ciphertext to be decoded</li> <li>• Plaintext message chosen by cryptanalyst, together with its corresponding ciphertext generated with the secret key</li> <li>• Purported ciphertext chosen by cryptanalyst, together with its corresponding decrypted plaintext generated with the secret key</li> </ul>

**Cryptology** covers both cryptography and cryptanalysis. *Cryptology* is a constantly evolving science; ciphers are invented and, given time, are almost certainly breakable. *Cryptanalysis* is the best way to understand the subject of cryptology. *Cryptographers* are constantly searching for the perfect security system, a system that is both fast and hard and a system that encrypts quickly but is hard or impossible to break. *Cryptanalysts* are always looking for ways to break the security provided by a cryptographic system, mostly through mathematical understanding of the cipher structure.

Cryptography can be defined as the conversion of data into a scrambled code that can be deciphered and sent across a public or a private network.

➤ A **Ciphertext-only attack** is an attack with an attempt to decrypt ciphertext when only the ciphertext itself is available.

➤ A **Known-plaintext attack** is an attack in which an individual has the plaintext samples and its encrypted version(ciphertext) thereby allowing him to use both to reveal further secret information like the key

➤ A **Chosen-plaintext attack** involves the cryptanalyst be able to define his own plaintext, feed it into the cipher and analyze the resulting ciphertext.

➤ A **Chosen-ciphertext attack** is one, where attacker has several pairs of plaintext-ciphertext and ciphertext chosen by the attacker.

An encryption scheme is **unconditionally secure** if the ciphertext generated by the scheme does not contain enough information to determine uniquely the corresponding plaintext, no matter how much ciphertext and time is available to the opponent. Example for this type is One-time Pad.

An encryption scheme is **computationally secure** if the ciphertext generated by the scheme meets the following criteria:

- Cost of breaking cipher exceeds the value of the encrypted information.
- Time required to break the cipher exceeds the useful lifetime of the information.

The average time required for exhaustive key search is given below:

Key Size (bits)	Number of Alternative Keys	Time required at 1 decryption/ $\mu$ s	Time required at $10^6$ decryptions/ $\mu$ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu\text{s} = 35.8 \text{ minutes}$	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142 \text{ years}$	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu\text{s} = 5.4 \times 10^{24} \text{ years}$	$5.4 \times 10^{18} \text{ years}$
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu\text{s} = 5.9 \times 10^{36} \text{ years}$	$5.9 \times 10^{30} \text{ years}$



# Substitution Encryption Techniques

These techniques involve substituting or replacing the contents of the plaintext by other letters, numbers or symbols. Different kinds of ciphers are used in substitution technique.

## Caesar Ciphers:

It is the oldest of all the substitution ciphers. A Caesar cipher replaces each letter of the plaintext with an alphabet. Two examples can be given:

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 Choose k, Shift all letters by k

- For example, if  $k = 5$
- A becomes F, B becomes G, C becomes H, and so on...

Mathematically give each letter a number,

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

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

➤ then have Caesar cipher as:

$$c = E(p) = (p + k) \bmod (26)$$

$$p = D(c) = (c - k) \bmod (26)$$

With a Caesar cipher, there are only 26 possible keys, of which only 25 are of any use, since mapping A to A etc doesn't really obscure the message!

## Monoalphabetic Ciphers :

Here, Plaintext characters are substituted by a different alphabet stream of characters shifted to the right or left by  $n$  positions. When compared to the Caesar ciphers, these monoalphabetic ciphers are more secure as each letter of the ciphertext can be any permutation of the 26 alphabetic characters leading to  $26!$  or greater than  $4 \times 10^{26}$  possible keys. But it is still vulnerable to cryptanalysis, when a cryptanalyst is aware of the nature of the plaintext, he can find the regularities of the language. To overcome these attacks, multiple substitutions for a single letter are used. For example, a letter can be substituted by different numerical cipher symbols such as 17, 54, 69..... etc. Even this method is not completely secure as each letter in the plain text affects on letter in the ciphertext.

Or, using a common key which substitutes every letter of the plain text.

The key                      ABCDEFGH IJ KLMNOPQRSTUVWXYZ

QWERTYU IOPAS DFGHJ KLZXCVC BNM

Would encrypt the message

*I think therefore I am*

into

**OZIIIOFAZIITKTYGKTOQD**

But any attacker would simply break the cipher by using frequency analysis by observing the number of times each letter occurs in the cipher text and then looking upon the English letter frequency table. So, substitution cipher is completely ruined by these attacks. Monoalphabetic ciphers are easy to break as they reflect the frequency of the original alphabet. A countermeasure is to provide substitutes, known as homophones for a single letter.

### Playfair Ciphers:

It is the best known multiple –letter encryption cipher which treats digrams in the plaintext as single units and translates these units into ciphertext digrams. The Playfair Cipher is a digram substitution cipher offering a relatively weak method of encryption. It was used for tactical purposes by British forces in the Second Boer War and in World War I and for the same purpose by the Australians and Germans during World War II. This was because Playfair is reasonably fast to use and requires no special equipment. A typical scenario for Playfair use would be to protect important but non-critical secrets during actual combat. By the time the enemy cryptanalysts could break the message, the information was useless to them.

It is based around a 5x5 matrix, a copy of which is held by both communicating parties, into which 25 of the 26 letters of the alphabet (normally either j and i are represented by the same letter or x is ignored) are placed in a random fashion.

For example, the plain text is Shi Sherry loves Heath Ledger and the agreed key is sherry. The matrix will be built according to the following rules.

- in pairs,
- without punctuation,
- All Js are replaced with Is.  
→ SH IS HE RR YL OV ES HE AT HL ED GE R
- Double letters which occur in a pair must be divided by an X or a Z.  
→
- E.g. LI TE RA LL Y LI TE RA LX LY  
→ SH IS HE RX RY LO VE SH EA TH LE DG ER

The alphabet square is prepared using, a 5\*5 matrix, no repetition letters, no Js and key is written first followed by the remaining alphabets with no i and j.

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

For the generation of cipher text, there are three rules to be followed by each pair of letters.

- letters appear on the same row : replace them with the letters to their immediate right respectively
- letters appear on the same column : replace them with the letters immediately below respectively
- not on the same row or column : replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair.

Based on the above three rules, the cipher text obtained for the given plain text is

→ **HE GH ER DR YS IQ WH HE SC OY KR AL RY**

Another example which is simpler than the above one can be given

as: Here, key word is playfair. Plaintext is Hellothere

hellothere becomes----he lx lo th er ex .

Applying the rules again, for each pair,

If they are in the same row, replace each with the letter to its right (mod 5) *he*

*KG*

If they are in the same column, replace each with the letter below it (mod 5) *lo*

*RV*

Otherwise, replace each with letter we'd get if we swapped their column indices *lx*

→  
*YV*

So the cipher text for the given plain text is **KG YV RV QM GI KU**

p	l	a	y	f
i	r	b	c	d
e	g	h	k	m
n	o	q	s	t
u	v	w	x	z

To decrypt the message, just reverse the process. Shift up and left instead of down and right. Drop extra x's and locate any missing i's that should be j's. The message will be back into the original readable form. no longer used by military forces because of the advent of digital encryption devices. Playfair is now regarded as insecure for any purpose because modern hand-held computers could easily break the cipher within seconds.

## Hill Cipher:

It is also a multiletter encryption cipher. It involves substitution of 'm' ciphertext letters for 'm' successive plaintext letters. For substitution purposes using 'm' linear equations, each of the characters are assigned a numerical values i.e. a=0, b=1, c=2, d=3,.....z=25.

For example if m=3, the system can be defined

$$c_1 = (k_{11}p_1 + k_{12}p_2 + k_{13}p_3) \bmod 26$$

$$c_2 = (k_{21}p_1 + k_{22}p_2 + k_{23}p_3) \bmod 26$$

$$c_3 = (k_{31}p_1 + k_{32}p_2 + k_{33}p_3) \bmod 26$$

If we represent in matrix form, the above statements as matrices and column vectors:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \bmod 26$$

Thus,  $C = KP \bmod 26$ , where  $C$  = Column vectors of length 3

$P$  = Column vectors of length 3

$K$  = 3x3 encryption key matrix.

For decryption process, inverse of matrix  $K$  i.e.  $K^{-1}$  is required which is defined by the equation  $KK^{-1} = K^{-1}K = I$ , where  $I$  is the identity matrix that contains only 0's and 1's as its elements. Plaintext is recovered by applying  $K^{-1}$  to the cipher text. It is expressed as

$$C = E_K(P) = KP \bmod 26$$

$$= D_K(C) = K^{-1}C \bmod 26.$$

$$= K^{-1}KP = IP = P$$

Example: The plain text is I can't do it and the size of  $m$  is 3 and key  $K$  is chosen as

**I can't do it**

**8 2 0 13 19 3 14 8 19**

following:

$$\begin{pmatrix} 9 & 18 & 10 \\ 16 & 21 & 1 \\ 5 & 12 & 23 \end{pmatrix}$$

The encryption process is carried out as follows

$$\begin{pmatrix} 4 \\ 14 \\ 12 \end{pmatrix} = \begin{pmatrix} 9 & 18 & 10 \\ 16 & 21 & 1 \\ 5 & 12 & 23 \end{pmatrix} \begin{pmatrix} 8 \\ 2 \\ 0 \end{pmatrix} \bmod 26$$

$$\begin{pmatrix} 19 \\ 12 \\ 14 \end{pmatrix} = \begin{pmatrix} 9 & 18 & 10 \\ 16 & 21 & 1 \\ 5 & 12 & 23 \end{pmatrix} \begin{pmatrix} 13 \\ 19 \\ 3 \end{pmatrix} \bmod 26$$

$$\begin{pmatrix} 18 \\ 21 \\ 9 \end{pmatrix} = \begin{pmatrix} 9 & 18 & 10 \\ 16 & 21 & 1 \\ 5 & 12 & 23 \end{pmatrix} \begin{pmatrix} 14 \\ 8 \\ 19 \end{pmatrix} \bmod 26$$

So, the encrypted text will be given as **EOM TMY SVJ**

The main advantages of hill cipher are given below:

- It perfectly hides single-letter frequencies.
  - Use of **3x3** Hill ciphers can perfectly hide both the single letter and two-letter frequency information.
  - Strong enough against the attacks made only on the cipher text.
- But, it still can be easily broken if the attack is through a known plaintext.

## Polyalphabetic Ciphers

In order to make substitution ciphers more secure, more than one alphabet can be used. Such ciphers are called **polyalphabetic**, which means that the same letter of a message can be represented by different letters when encoded. Such a one-to-many correspondence makes the use of frequency analysis much more difficult in order to crack the code. We describe one such cipher named for *Blaise de Vigenere* a 16-th century Frenchman.

The **Vigenere cipher** is a polyalphabetic cipher based on using successively shifted alphabets, a different shifted alphabet for each of the 26 English letters. The procedure is based on the tableau shown below and the use of a keyword. The letters of the keyword determine the shifted alphabets used in the encoding process.

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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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



For the message COMPUTING GIVES INSIGHT and keyword LUCKY we proceed by repeating the keyword as many times as needed above the message, as follows.

L	U	C	K	Y	L	U	C	K	Y	L	U	C	K	Y	L					
C	O	M	P	U	T	I	N	G	G	I	V	E	S	I	N	S	I	G	H	T

Encryption is simple: Given a key letter x and a plaintext letter y, the ciphertext letter is at the intersection of the row labeled x and the column labeled y; so for L, the ciphertext letter would be N. So, the ciphertext for the given plaintext would be given as:

L	U	C	K	Y	L	U	C	K	Y	L	U	C	K	Y	L					
C	O	M	P	U	T	I	N	G	G	I	V	E	S	I	N	S	I	G	H	T
N	I	O	Z	S	E	C	P	Q	E	T	P	G	C	G	Y	M	K	Q	F	E

<==MESSAGE

<==Encoded Message

Decryption is equally simple: The key letter again identifies the row and position of ciphertext letter in that row decides 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 and thereby making the letter frequency information is obscured. Still, breaking this cipher has been made possible because this reveals some mathematical principles that apply in cryptanalysis. To overcome the drawback of the periodic nature of the keyword, a new technique is proposed which is referred as an autokey system, in which a key word is concatenated with the plaintext itself to provide a running key. For ex

In the above example, the key would be *luckycomputinggivesin*

Still, this scheme is vulnerable to cryptanalysis as both the key and plaintext share the same frequency distribution of letters allowing a statistical technique to be applied. Thus, the ultimate defense against such a cryptanalysis is to choose a keyword that is as long as plaintext and has no statistical relationship to it. A new system which works on binary data rather than letters is given as

$C_i = p_i \oplus k_i$  where,

$p_i$  = ith binary digit of plaintext

$k_i$  = ith binary digit of key

$C_i$  = ith binary digit of ciphertext

$\oplus$  = exclusive-or operation.

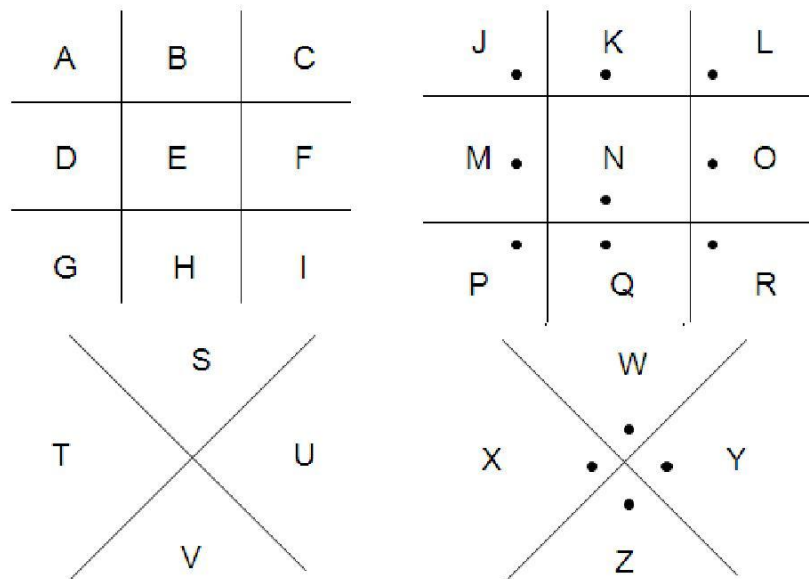
Because of the properties of XOR, decryption is done by performing the same bitwise operation.

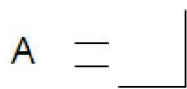
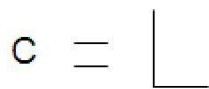
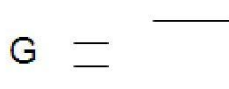
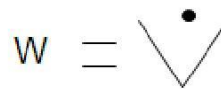
$$p_i = C_i \oplus k_i$$

A very long but, repetition key word is used making cryptanalysis difficult.

## Pigpen Cipher

Pigpen cipher is a variation on letter substitution. Alphabets are arranged as follows:



A =  C =  G =  W = 

Alphabets will be represented by the corresponding diagram. E.g., WAG would be



This is a weak cipher.

## Transposition techniques

A **transposition cipher** is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed. Transposition ciphers encrypt plaintext by moving small pieces of the message around. Anagrams are a primitive transposition cipher. This table shows "VOYAGER" being encrypted with a primitive transposition cipher where every two letters are switched with each other:

V	O	Y	A	G	E	R
O	V	A	Y	E	G	R

Another simple example for transposition cipher is the rail fence technique, in which the plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

For example, write the message “meet me after the toga party” out as:

m e m a t r h t g p r y  
e t e f e t e o a a t



giving ciphertext : **MEMATRHTGPRYETEFETEOAAT**

The following example shows how a pure permutation cipher could work: You write your plaintext message along the rows of a matrix of some size. You generate ciphertext by reading along the columns. The order in which you read the columns is determined by the encryption key:

key: 2 5 3 1 6 4

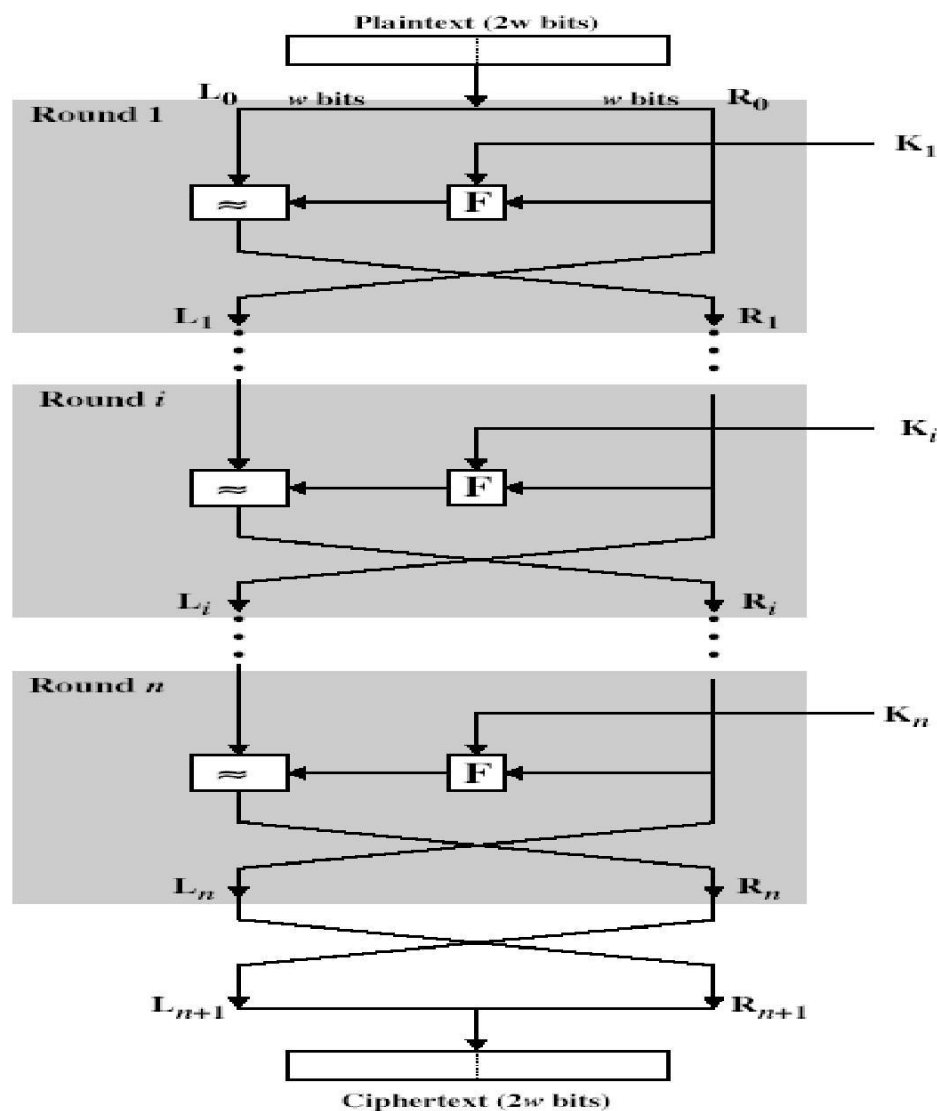
plaintext: m e e t m e  
a t m i d n  
i g h t f o  
r t h e g o  
d i e s x y

ciphertext: **TITESMAIRDEMHHREENOOYETGTI**

The cipher can be made more secure by performing multiple rounds of such permutations.

# Feistel Cipher Structure

Most symmetric block ciphers are based on a *Feistel Cipher Structure*. It was first described by Horst Feistel of IBM in 1973 and is still forms the basis for almost all conventional encryption schemes. It makes use of two properties namely *diffusion* and *confusion*; identified by Claude Shannon for frustrating statistical cryptanalysis. Confusion is basically defined as the concealment of the relation between the secret key and the cipher text. On the other hand, diffusion is regarded as the complexity of the relationship between the plain text and the cipher text.



The function of Feistel Cipher is shown in the above figure and can be explained by following steps:

- The input to the encryption algorithm is a plaintext block of length  $2w$  bits and a key  $K$ .
- The plaintext block is divided into two halves:  $L_i$  and  $R_i$ .
- The two halves pass through  $n$  rounds of processing and then combine to produce the cipher text block
- Each Round  $i$  has inputs  $L_{i-1}$  and  $R_{i-1}$ , derived from the previous round, as well as a unique subkey  $K_i$  generated by a sub-key generation algorithm.
- All rounds have the same structure which involves substitution (mapping) on left half of data, which is done by applying a round function  $F$  to right half of data and then taking XOR of the output of that function and left half of data. The round function  $F$  is common to every round but parameterized by round subkey  $K_i$ .
- Then a permutation is performed that consists of interchange of the two halves of data.

For each round  $i = 0, 1, \dots, n$ , compute

$$\begin{aligned} L_{i+1} &= R_i \\ R_{i+1} &= L_i \oplus F(R_i, K_i). \end{aligned}$$

Then the ciphertext is  $(R_{n+1}, L_{n+1})$ .

Decryption of a ciphertext  $(R_{n+1}, L_{n+1})$  is accomplished by computing for  $i = n, n-1, \dots, 0$

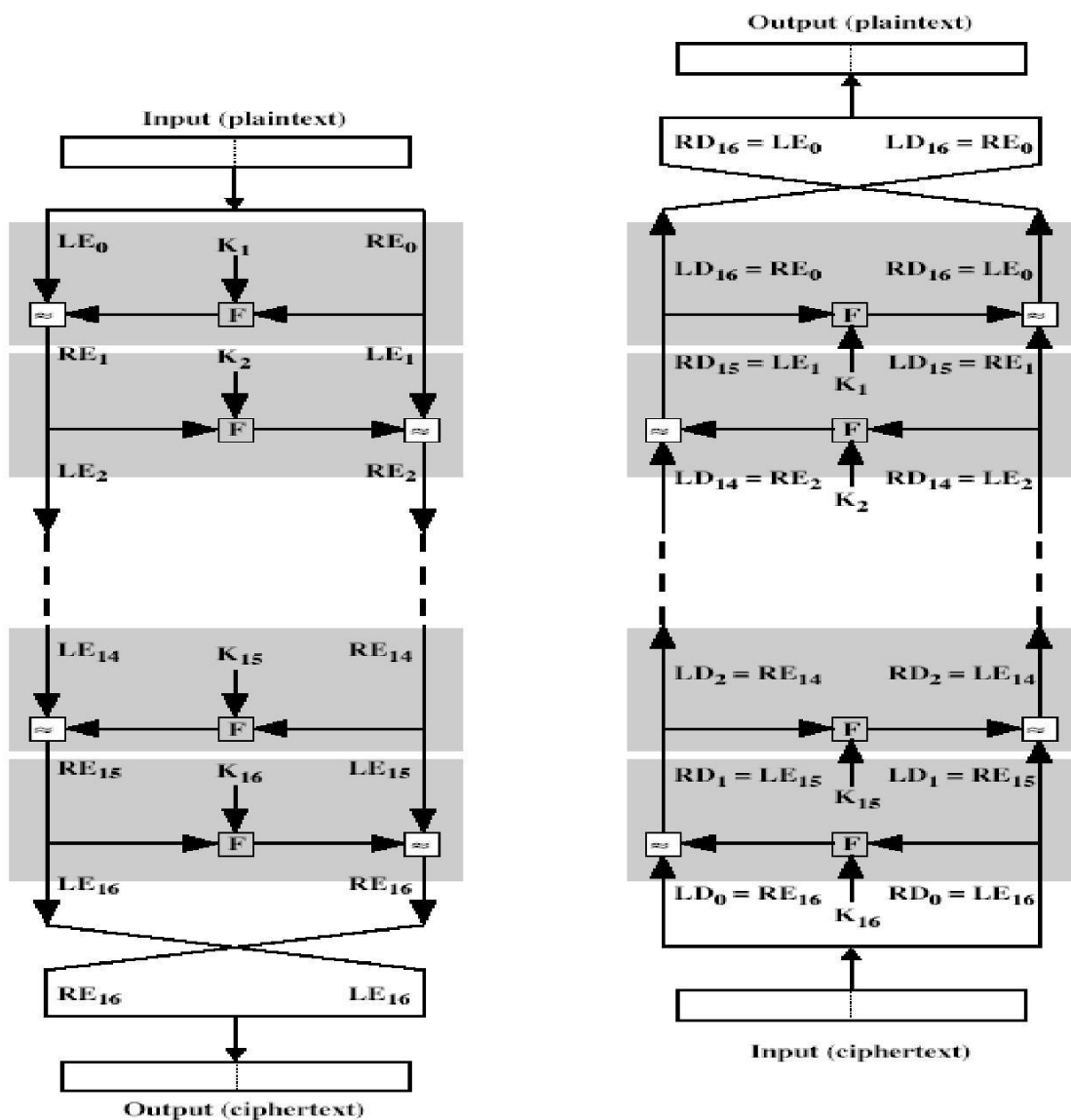
$$\begin{aligned} R_i &= L_{i+1} \\ L_i &= R_{i+1} \oplus F(L_{i+1}, K_i). \end{aligned}$$

Then  $(L_0, R_0)$  is the plaintext again.

The structure is a particular form of substitution-permutation network (SPN) proposed by Shannon. The realization or development of a Feistel encryption scheme depends on the choice of the following parameters and design features:

- **Block size:** larger block sizes mean greater security but slower processing. Block size of 64 bits has been nearly universal in block cipher design.
- **Key Size:** larger key size means greater security but slower processing. Most common key length in modern algorithms is 128 bits.
- **Number of rounds:** multiple rounds offer increasing security but slows cipher. Typical size is 16 rounds.
- **Subkey generation algorithm:** greater complexity will lead to greater difficulty of cryptanalysis.
- **Round Function:** greater complexity will make cryptanalysis harder.
- **Fast software en/decryption & ease of analysis:** are more recent concerns for practical use and testing.





### Feistel Cipher Decryption

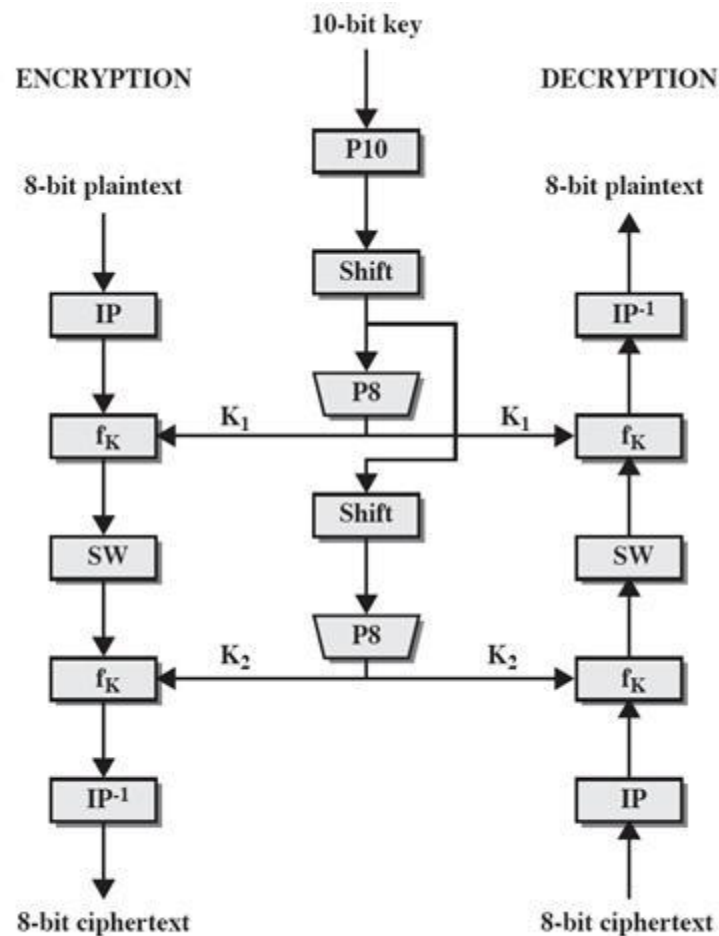
The process of decryption with a Feistel cipher is same as the encryption process. Use the ciphertext as input to the algorithm, but use the subkeys  $K_i$  in the reverse order. Use  $K_n$  in the first round and  $K_{n-1}$  in the second round and so on until  $K_1$  is used in the last round. Main advantage is we need not implement two different algorithms for encryption and decryption.

The Feistel cipher has the advantage that encryption and decryption operations are very similar, even identical in some cases requiring only a reversal in the key schedule. Therefore, the size of the code or circuitry required to implement such a cipher is nearly halved.

## Conventional Encryption Algorithms

### *Simplified DES*

S-DES is a reduced version of the DES algorithm. It has similar properties to DES but deals with a much smaller block and key size (operates on 8-bit message blocks with a 10-bit key). The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce that ciphertext as input and produces the original 8-bit block of plaintext. S-DES scheme is shown below:



Simplified DES Scheme

The encryption algorithm involves five functions: and initial permutation(IP), a complex function labeled  $f_k$ , which involves both permutations and substitution operations and depends on a key input, a single permutation function (SW) that switches the two halves of the data, the function  $f_k$  again and finally a permutation function that is inverse of the IP i.e.  $IP^{-1}$ .

As shown in figure, the function  $f_k$  takes the data from encryption function along with 8-bit key. The key is chosen to be 10-bit length from which two 8-bit subkeys are generated. The initial 10-bit key is subjected to a permutation (P10) followed by a shift operation. The output of this shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first key ( $k_1$ ) and also feeds into another shift and another instance of P8 to produce the second subkey ( $k_2$ ). The encryption algorithm can be written as:

$$\text{Ciphertext} = \text{IP}^{-1} ( f_{k_2}(\text{SW}(f_{k_1}(\text{IP}(\text{plaintext})))) )$$

Where  $K_1 = \text{P8}(\text{shift}(\text{p10}(\text{key})))$

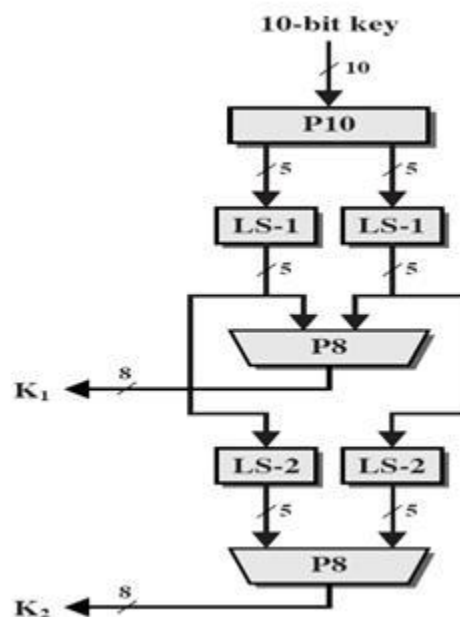
$K_2 = \text{P8}(\text{shift}(\text{shift}(\text{p10}(\text{key}))))$

Decryption is also shown in the above figure and can be given as:

$$\text{Plaintext} = \text{IP}^{-1} ( f_{k_1}(\text{SW}(f_{k_2}(\text{IP}(\text{ciphertext})))) )$$

### Key Generation:

The key generation process is shown below:



As shown above, a 10-bit key shared between sender and receiver is used and first passed through a permutation P10, Where P10 is a permutation with table:

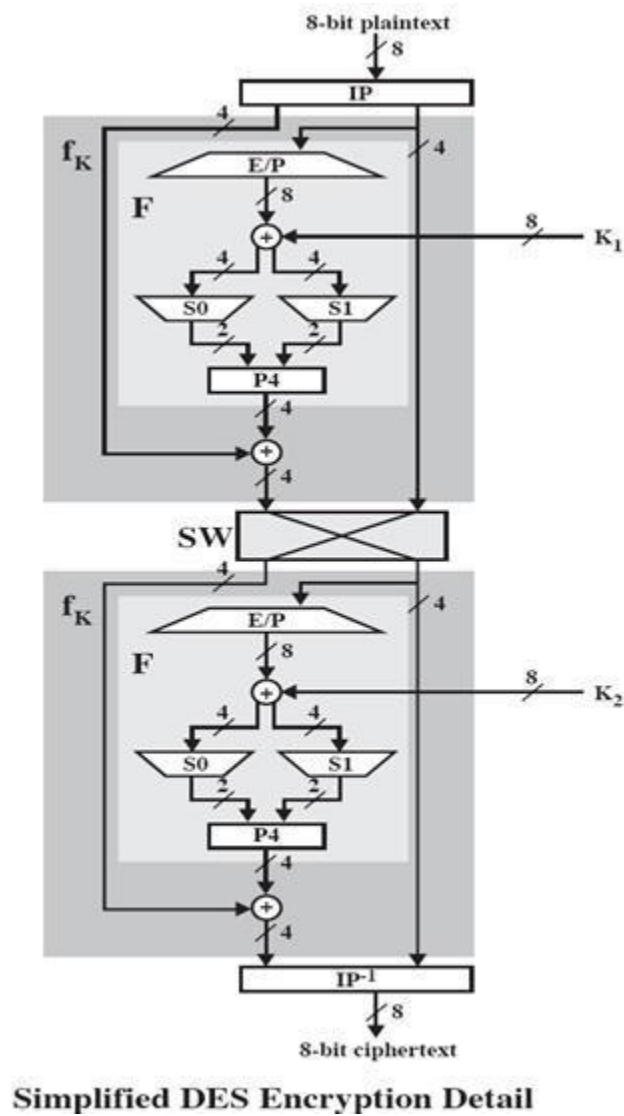
P10									
3	5	2	7	4	10	1	9	8	6

LS-1 is a circular left shift of 1 bit position, and LS-2 is a circular left shift of 2 bit positions. P8 is another permutation which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 ( $K_1$ ) and then the outputs from the two LS-1 functions are taken and a circular left shift of 2 bit positions is performed on each string and then P8 is applied again to produce  $K_2$ .

### S-DES Encryption:



As shown above, the input to algorithm is an 8-bit block of plaintext which is permuted using the IP function. The inverse to this function  $IP^{-1}$  is applied towards the end of algorithm. IP is the initial permutation and  $IP^{-1}$  is its inverse.

IP							
2	6	3	1	4	8	5	7

$IP^{-1}$							
4	1	3	5	7	2	8	6

### The function $f_k$

It is the most complex component of S-DES. Function  $f_k$  consists of a combination of permutation and substitution functions.

$$f_k(L, R) = (L \oplus F(R, SK), R)$$

where, SK is a subkey (i.e.  $K_1$  or  $K_2$ ), L and R denote the leftmost and rightmost 4 bits of the 8-bit input  $f_k$  and let F be a mapping function from 4-bit strings to 4-bit strings. The first operation is expansion/permutation operation given by:

E/P							
4	1	2	3	2	3	4	1

$S_0$  and  $S_1$  are to S-boxes operates according to the following tables:

$S_0$ :

1	0	3	2
3	2	1	0
0	2	1	3
3	1	3	2

$S_1$ :

0	1	2	3
2	0	1	3
3	0	1	0
2	1	0	3

And P4 would be another permutation.

P4			
2	3	4	1

The output of P4 would be the output of function F.

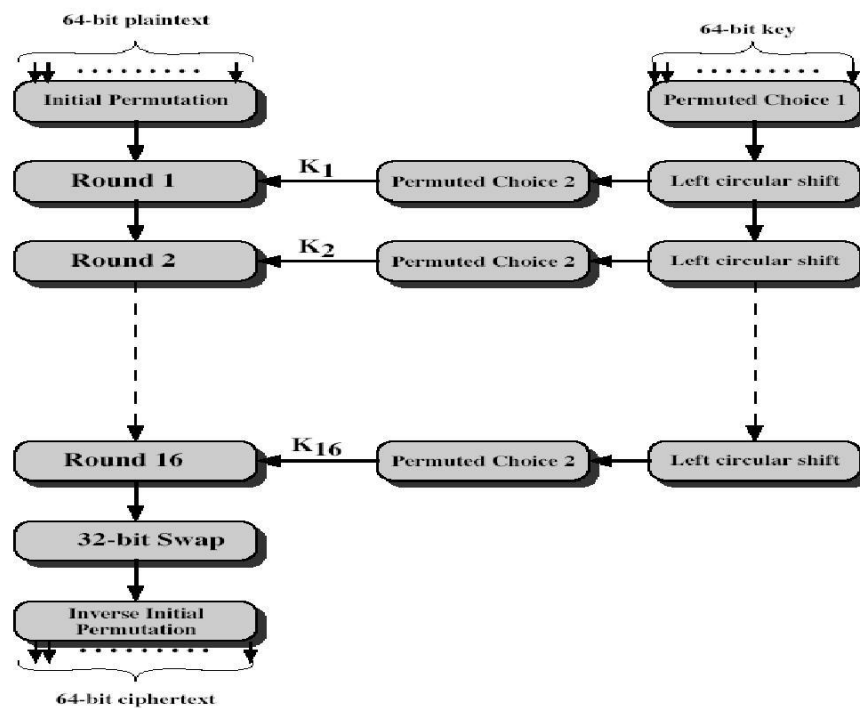


### The Switch Function:

This function interchanges the left and right 4 bits so that the second instance of  $f_k$  operates on a different 4 bits. For second instance all other parameters remain same, but the key is  $K_2$ . The S-boxes operates as follows:- The first and fourth input bits are treated as 2-bit numbers that specify a row of the S-box, and the second and third input bits specify a column of S-box. The entry in that row and column in base2 is the 2-bit output.

## Data Encryption Standard

In 1974, IBM proposed "Lucifer", an encryption algorithm using 64-bit keys. Two years later (1977), NBS (now NIST) in consultation with NSA made a modified version of that algorithm into a standard. DES uses the two basic techniques of cryptography - confusion and diffusion. At the simplest level, diffusion is achieved through numerous permutations and confusion is achieved through the XOR operation and the S-Boxes. This is also called an S-P network. The DES encryption scheme can be explained by the following figure



The plain text is 64 bits in length and the key is 56 bits in length. Longer plain text amounts are processed in 64-bit blocks. The main phases in the left hand side of the above figure i.e. processing of the plain text are,



**Initial Permutation (IP):** The plaintext block undergoes an initial permutation. 64 bits of the block are permuted.



**A Complex Transformation:** 64 bit permuted block undergoes 16 rounds of complex transformation. Subkeys are used in each of the 16 iterations.



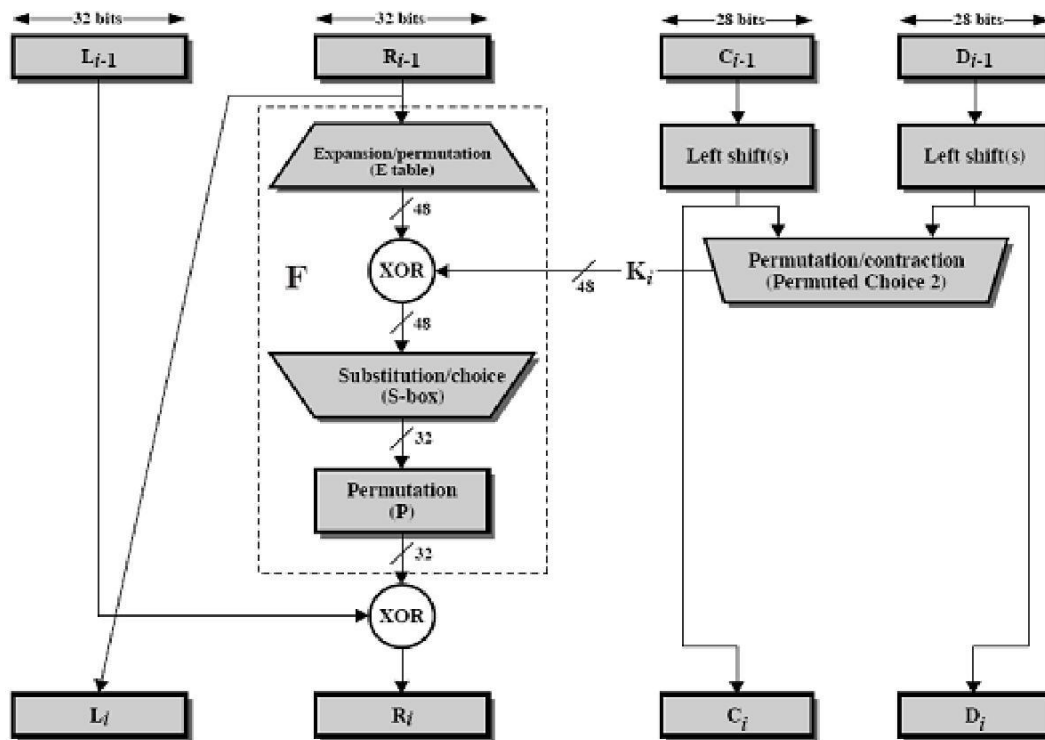
**32-bit swap:** The output of 16<sup>th</sup> round consists of 64bits that are a function of input plain text and key. 32 bit left and right halves of this output is swapped.



**Inverse Initial Permutation ( $IP^{-1}$ ):** The 64 bit output undergoes a permutation that is inverse of the initial permutation.

On the right hand side part of the figure, the usage of the 56 bit key is shown. Initially the key is passed through a permutation function. Now for each of the 16 iterations, a new subkey ( $K_i$ ) is produced by combination of a left circular shift and a permutation function which is same for each iteration. A different subkey is produced because of repeated shifting of the key bits.

The following figure shows a closer view of algorithms for a single iteration. The 64bit permuted input passes through 16 iterations, producing an intermediate 64-bit value at the conclusion of each iteration.



Single Round of DES Algorithm

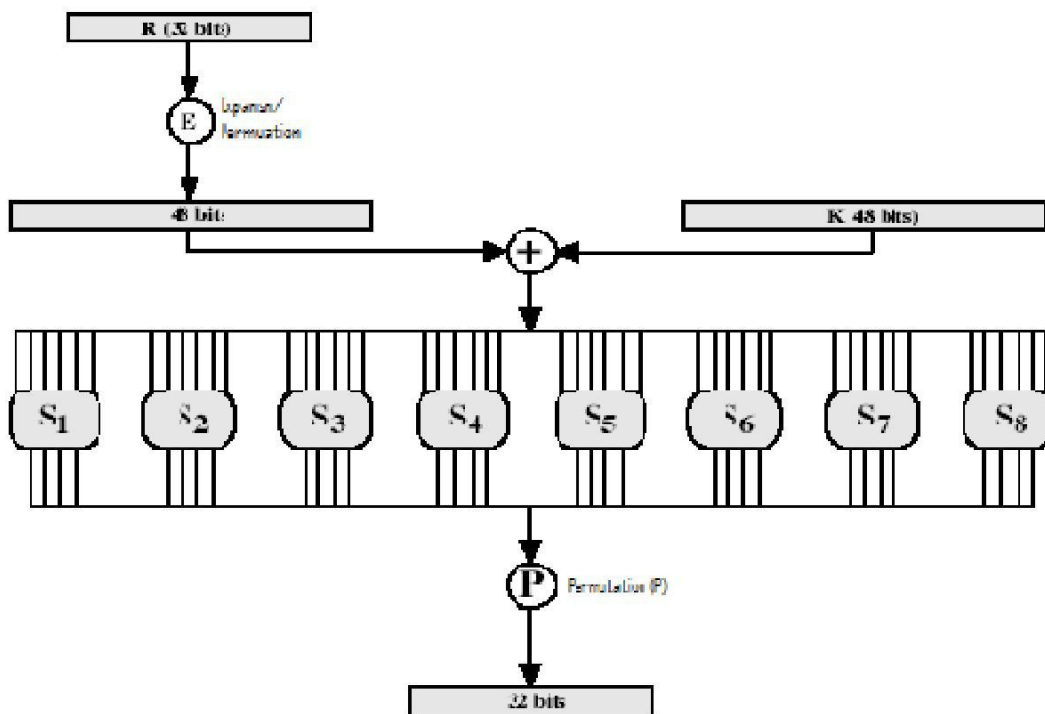
The left and right halves of each 64 bit intermediate value are treated as separated 32-bit quantities labeled L (left) and R (Right). The overall processing at each iteration is given by following steps, which form one round in an S-P network.

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$$

Where Function F can be described as  $P(S(E(R_{i-1}) \oplus K(i)))$

The left hand output of an iteration ( $L_i$ ) is equal to the right hand input to that iteration  $R_{i-1}$ . The right hand output  $R_i$  is exclusive OR of  $L_{i-1}$  and a complex function F of  $R_{i-1}$  and  $K_i$ . The function F can be depicted by the following figure.  $S_1, S_2, \dots, S_8$  represent the "S-boxes", which maps each combination of 48 input bits into a particular 32 bit pattern. For the generation of subkey of length 48 bits, a 56bit key is used which is first passed through a permutation function and then halved to get two 28 bit quantities labeled  $C_0$  and  $D_0$ . At each iteration, these two C and D are subjected to a circular left shift or rotation of 1 or 2 bits. These shifted values serve as input to the next iteration and also to another permutation function which produces a 48-bit output. This output is fed as input to function  $F(R_{i-1}, K_i)$ .



The first and last bits of the input to the box  $S_i$  form a 2-bit binary number to select one of four substitutions defined by the four rows in the table for  $S_i$ . The middle 4-bits select a particular column. The decimal value in the cell selected by the row and column is converted to its 4-bit representation to produce the output.

The substitution consists of a set of eight S-boxes, each of which accepts 6 bits as input and produces 4 bits as output. The process of decryption with DES is essentially the same as the encryption process: no different algorithm is used. The ciphertext is used as input to the DES algorithm and the keys are used in the reverse order i.e. K<sub>16</sub> in the first iteration, K<sub>15</sub> on the second iteration and so on until k<sub>1</sub> is used on the sixteenth and last iteration.

### **Strength of DES:**

*Avalanche Effect:* An effect in DES and other secret key ciphers where each small change in plaintext implies that somewhere around half the ciphertext changes. The avalanche effect makes it harder to successfully cryptanalyze the ciphertext. DES exhibits a strong Avalanche effect.

Concern about the strength of DES falls into two categories i.e. strength of algorithm itself and use of 56-bit key. Though many attempts were made over the years to find and exploit weaknesses in the algorithm, none of them were successful in discovering any fatal weakness in DES. A serious concern is with the key size as the time passed the security in DES became getting compromised by the advent of supercomputers which succeeded in breaking the DES quickly using a brute-force attack. If the only form of attack that could be made on an encryption algorithm is brute force, the way of countering it is obviously using long keys. If a key of size 128 bits is used, it takes approximately  $10^{18}$  years to break the code making the algorithm unbreakable by brute-force approach.

The two analytical attacks on DES are Differential cryptanalysis and Linear cryptanalysis. Both make use of Known plaintext-ciphertext pairs and try to attack the round structure and the S-Boxes. Recent advancements showed that using Differential cryptanalysis, DES can be broken using  $2^{47}$  plaintext-ciphertext pairs and for linear cryptanalysis, the number is even reduced to  $2^{41}$ .

### **Triple DES**

The first answer to problems of DES is an algorithm called Double DES which includes double encryption with two keys. It increases the key size to 112 bits, which seems to be secure. But, there are some problems associated with this approach.

issue of reduction to single stage:

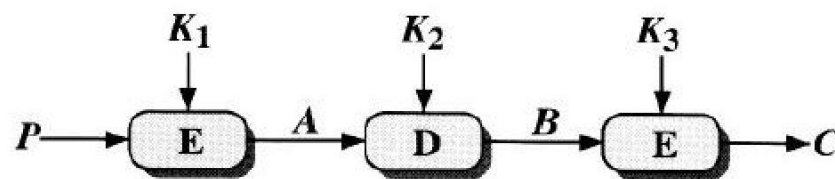
In other words, could there be a key K<sub>3</sub> such that  $E_{K_2}(E_{K_1}(P)) = E_{K_3}(P)$ ?

“meet-in-the-middle” attack:

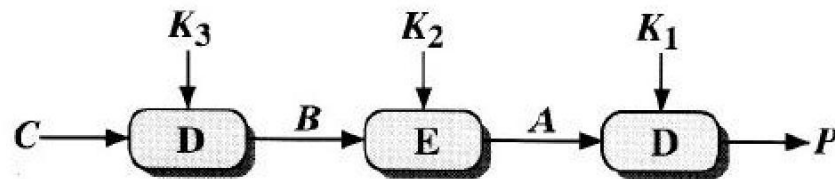
- Works when given a known (P,C) pair
- since  $X = E_{K_1}(P) = D_{K_2}(C)$
- attack by encrypting P with all  $2^{56}$  keys K<sub>1</sub> and store
- then decrypt C with all possible  $2^{56}$  keys K<sub>2</sub> and match X value

- Test the two keys for the second pair of plaintext-ciphertext and if they match, correct keys are found

Triple DES was the answer to many of the shortcomings of DES. Since it is based on the DES algorithm, it is very easy to modify existing software to use Triple DES. 3DES was developed in 1999 by IBM – by a team led by Walter Tuchman. 3DES prevents a meet-in-the-middle attack. 3DES has a 168-bit key and enciphers blocks of 64 bits. It also has the advantage of proven reliability and a longer key length that eliminates many of the shortcut attacks that can be used to reduce the amount of time it takes to break DES. 3DES uses three keys and three executions of the DES algorithm. The function follows an encrypt-decrypt-encrypt (EDE) sequence.



(a) Encryption



(b) Decryption

$$C = E_{K3}[D_{K2}[E_{K1}[P]]]$$

Where C= ciphertext, P= plaintext and

$E_K[X]$  = encryption of X using key K

$D_K[Y]$  = decryption of Y using key K

Decryption is simply the same operation with the keys reversed

$$P = D_{K1}[E_{K2}[D_{K3}[C]]]$$

Triple DES runs three times slower than standard DES, but is much more secure if used properly. With three distinct keys, TDEA has an effective key length of 168 bits making it a formidable algorithm. As the underlying algorithm is DEA, it offers the same resistance to cryptanalysis as is DEA.

Triple DES can be done using 2 keys or 3 keys.



## International Data Encryption Standard

The International Data Encryption Standard Algorithm (IDEA) is a symmetric block cipher that was proposed to replace DES. It is a minor revision of an earlier cipher, PES (Proposed Encryption Standard). IDEA was originally called IPES (Improved PES) and was also included in PGP. IDEA is a block cipher which uses a 128-bit length key to encrypt successive 64-bit blocks of plaintext.

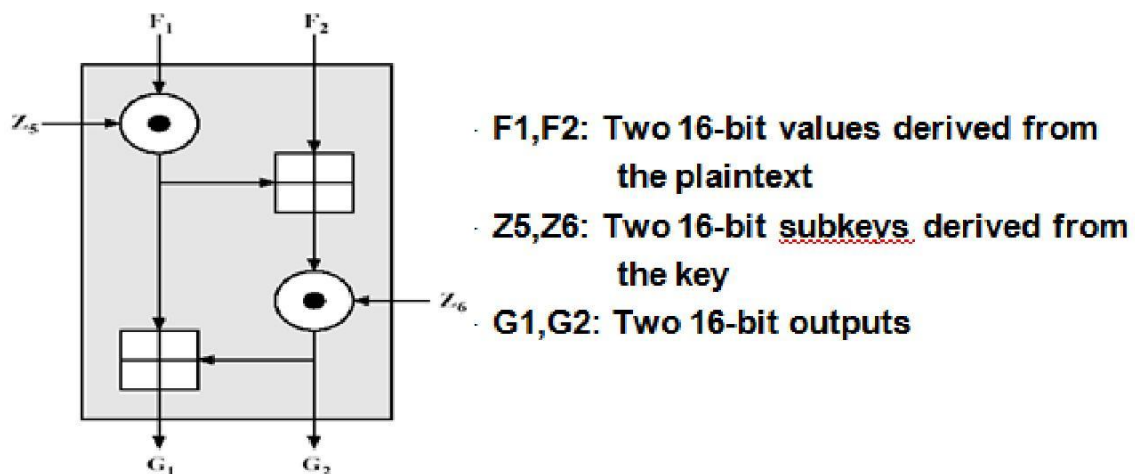
The main design goals of IDEA are,

- **Block Length:** Block size of 64 bits is considered strong enough to deter statistical analysis. Also usage of Cipher Feedback Mode of operation provides better strength.
- **Key Length:** Its key size of 128 bits is very secure to deter exhaustive search.

IDEA's overall scheme is based on three different operations: Bit by Bit XOR denoted as  $\oplus$ , addition mod  $2^{16}$  denoted as  $\boxplus$  and multiplication mod  $(2^{16} + 1)$  as  $\odot$ . All operations are on 16-bit sub-blocks, with no permutations used. Completely avoid substitution boxes and table lookups used in the block ciphers. The algorithm structure has been chosen such that when different key sub-blocks are used, the encryption process is identical to the decryption process.

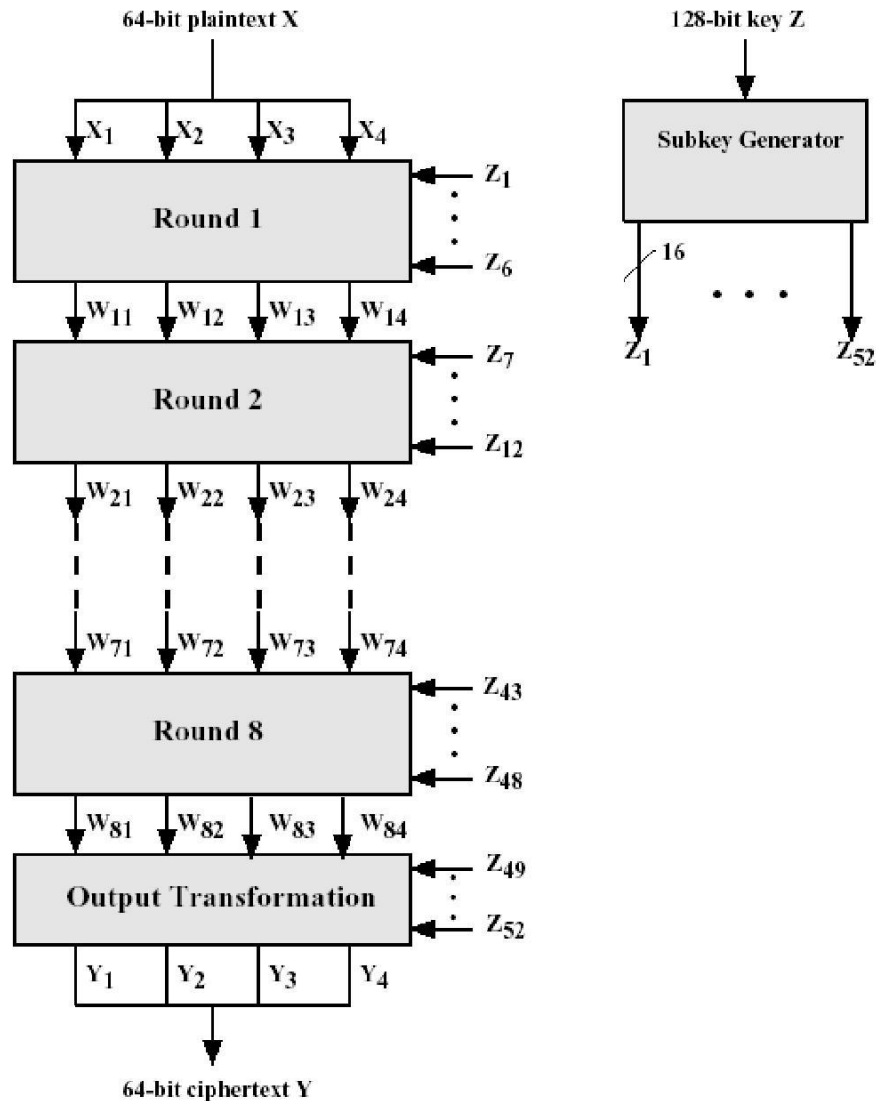
In IDEA, **Confusion** is achieved by using these three separate operations in combination providing a complex transformation of the input, making cryptanalysis much more difficult (than with a DES which uses just a single XOR).

The main basic building block is the Multiplication/Addition (MA) structure shown below:



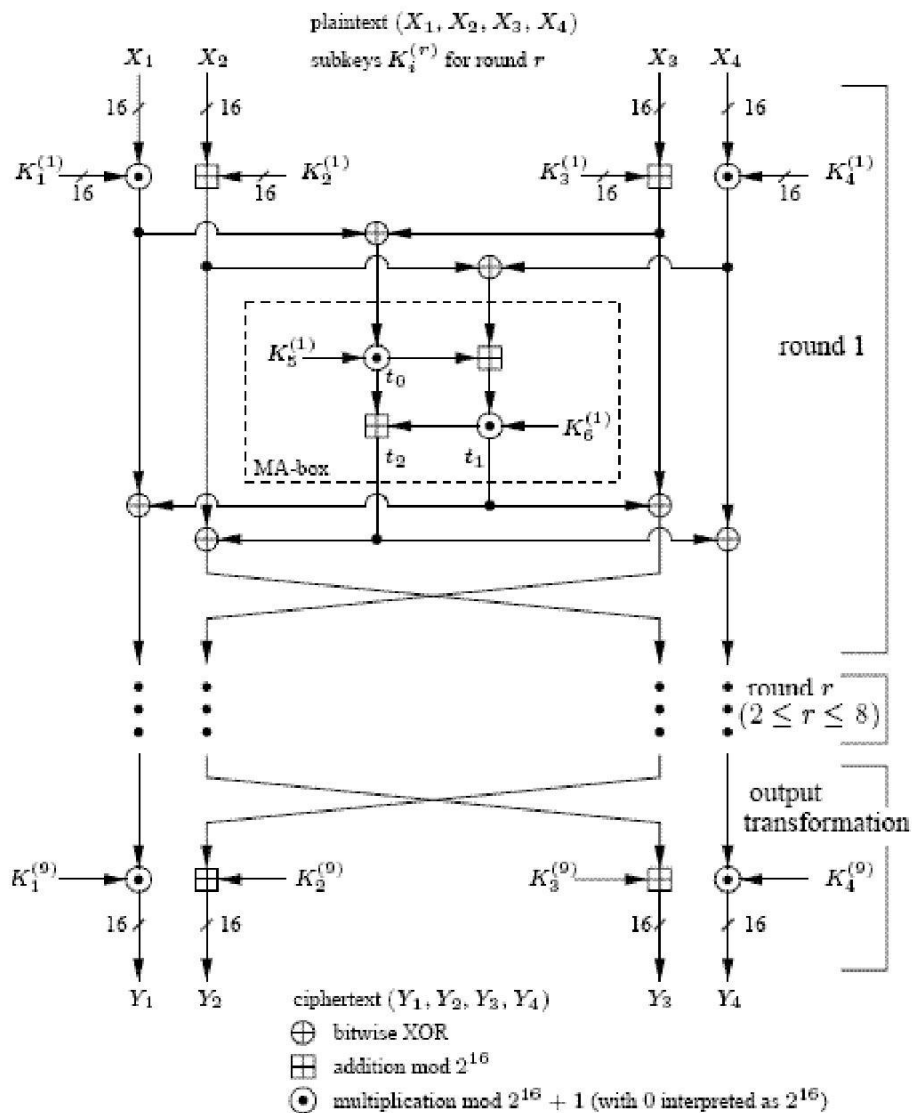
**Diffusion** is provided by this MA structure where, each output bit depends on every bit of inputs (plaintext-derived inputs and subkey inputs). This MA structure is repeated eight times, providing very effective diffusion

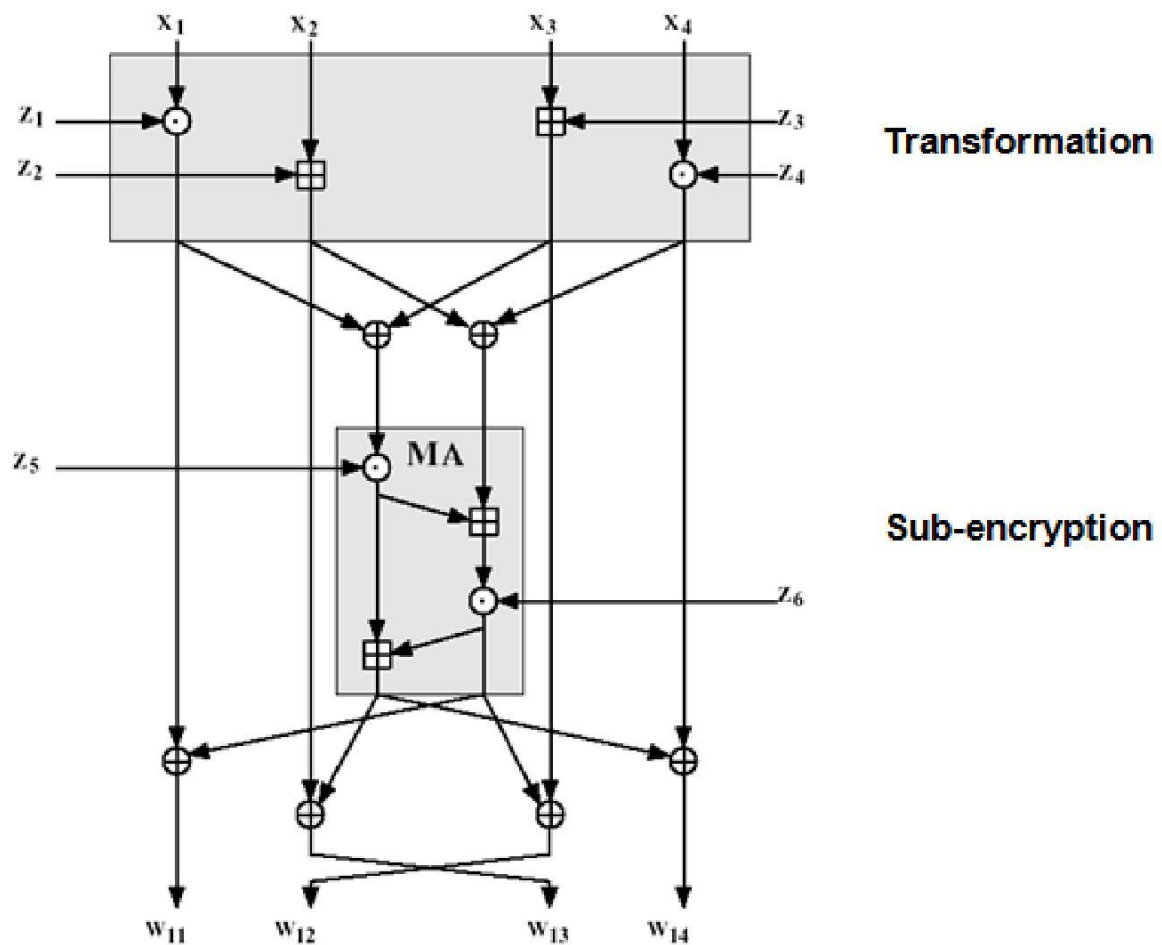
The overall scheme for IDEA is shown below:



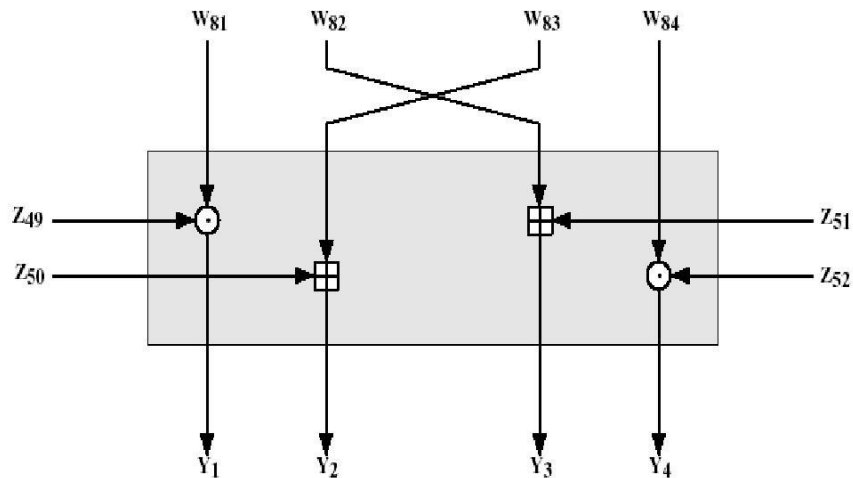
The encryption function takes two inputs; one being the plaintext to be encrypted and the key. The plaintext is 64 bits in length and key is 128 bits in length. The IDEA algorithm consists of eight rounds followed by a final transformation function. The algorithm divides the input into four 16-bit sub-blocks. Each of the rounds takes four 16-bit sub-blocks as input and produces four 16-bit output blocks. The final transformation also produces four 16-bit blocks, which are concatenated to form the 64-bit ciphertext. Each of the rounds also makes use of six 16-bit subkeys, whereas the final transformation uses four subkeys, for a total of 52 subkeys.

First, the 128-bit key is partitioned into eight 16-bit sub-blocks which are then directly used as the first eight key sub-blocks {i.e.  $Z_1, Z_2 \dots Z_8$  are taken directly from the 128-bit key where  $Z_1$  equals the first 16 bits,  $Z_2$  corresponding to next 16 bits and so on}. The 128-bit key is then cyclically shifted to the left by 25 positions, after which the resulting 128-bit block is again partitioned into eight 16-bit sub-blocks to be directly used as the next eight key sub-blocks. The cyclic shift procedure described above is repeated until all of the required 52 16-bit key sub-blocks have been generated. The following figure shows the single round in the encryption algorithm.





IDEA deviates from the Feistel Structure that the round starts with a transformation that combines four input subblocks with four subkeys, using the addition and multiplication operations. These four output blocks are then combined using the XOR operation to form two 16-bit blocks that are input to the MA structure, which also takes two subkeys as input and combines these inputs to produce 16-bit outputs. Finally, the four output blocks from the upper transformation are combined with the two output blocks of the MA (Multiplication/Addition) structure using XOR to produce the four output blocks for this round. Also, the two outputs that are partially generated by the second and third inputs( $X_2$  and  $X_3$ ) are interchanged to produce the second and third outputs ( $W_{12}$  and  $W_{13}$ ). This increases the mixing of bits being processed and makes the algorithm more resistant to differential cryptanalysis.

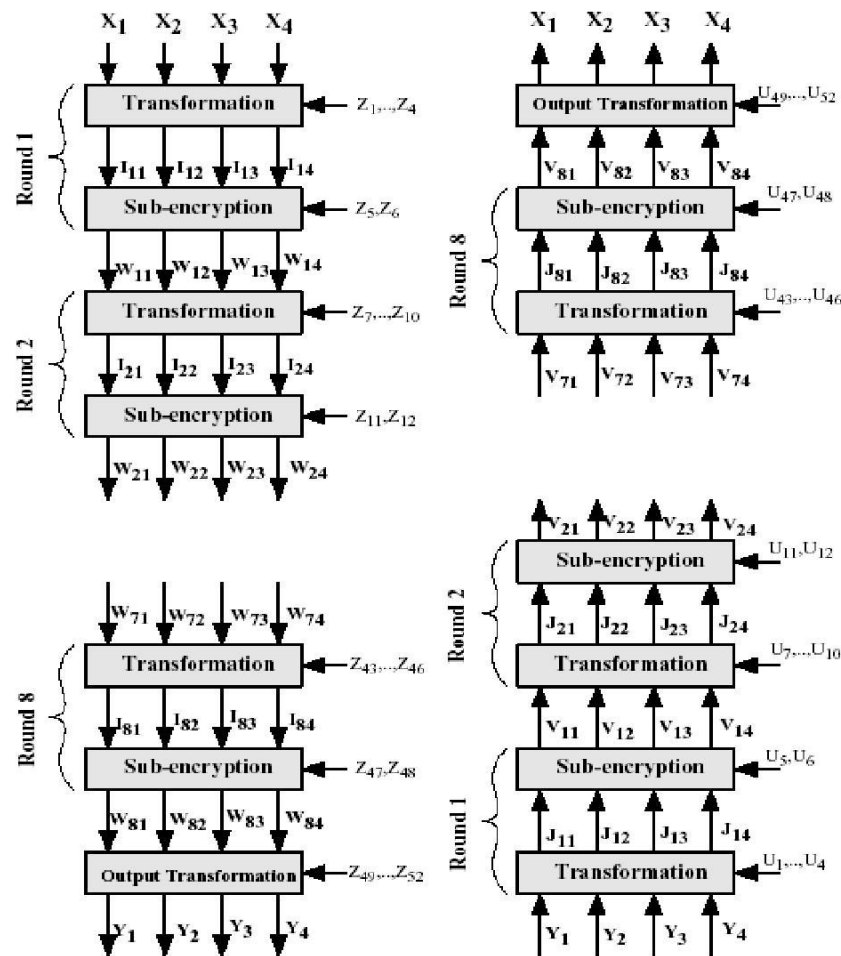


The ninth stage of the algorithm, labelled the output transformation stage has the same structure as the upper rounds, but the only difference is that the second and third inputs are interchanged before being applied to the operational units. The effect of this is undoing the interchange at the end of eighth round. The reason for this extra interchange is so that decryption has the same structure as encryption. The ninth stage requires only four subkey inputs, compared to six subkey inputs for each of the first eight stages.

The computational process used for decryption of the ciphertext is essentially the same as that used for encryption. The only difference is that each of the 52 16-bit key sub-blocks used for decryption is the inverse of the key sub-block used during encryption. In addition, the key sub-blocks must be used in the reverse order during decryption in order to reverse the encryption process.

### **Decryption Steps:**

- ② The first four subkeys of decryption round  $i$  are derived from the first four subkeys of encryption round  $(10-i)$ , where the transformation stage is counted as round 9. The first and fourth decryption subkeys are equal to the multiplicative inverse modulo  $(2^{16} + 1)$  of the corresponding first and fourth encryption subkeys. For rounds 2 through 8, the second and third decryption subkeys are equal to the additive inverse modulo  $(2^{16})$  of the corresponding third and second encryption subkeys. For rounds 1 and 9, the second and third decryption subkeys are equal to the additive inverse modulo  $(2^{16})$  of the corresponding second and third encryption subkeys.
- ② For the first eight rounds, the last two subkeys of decryption round  $i$  are equal to the last two subkeys of encryption round  $(9-i)$ .



	Encryption		Decryption	
Stage	Designation	Equivalent to	Designation	Equivalent to
Round 1	Z <sub>1</sub> Z <sub>2</sub> Z <sub>3</sub> Z <sub>4</sub> Z <sub>5</sub> Z <sub>6</sub>	Z[1..96]	U <sub>1</sub> U <sub>2</sub> U <sub>3</sub> U <sub>4</sub> U <sub>5</sub> U <sub>6</sub>	Z <sub>49</sub> <sup>-1</sup> -Z <sub>50</sub> -Z <sub>51</sub> Z <sub>52</sub> <sup>-1</sup> Z <sub>47</sub> Z <sub>48</sub>
Round 2	Z <sub>7</sub> Z <sub>8</sub> Z <sub>9</sub> Z <sub>10</sub> Z <sub>11</sub> Z <sub>12</sub>	Z[97..128; 26..89]	U <sub>7</sub> U <sub>8</sub> U <sub>9</sub> U <sub>10</sub> U <sub>11</sub> U <sub>12</sub>	Z <sub>43</sub> <sup>-1</sup> -Z <sub>45</sub> -Z <sub>44</sub> Z <sub>46</sub> <sup>-1</sup> Z <sub>41</sub> Z <sub>42</sub>
Round 3	Z <sub>13</sub> Z <sub>14</sub> Z <sub>15</sub> Z <sub>16</sub> Z <sub>17</sub> Z <sub>18</sub>	Z[90..128; 1..25; 51..82]	U <sub>13</sub> U <sub>14</sub> U <sub>15</sub> U <sub>16</sub> U <sub>17</sub> U <sub>18</sub>	Z <sub>37</sub> <sup>-1</sup> -Z <sub>39</sub> -Z <sub>38</sub> Z <sub>40</sub> <sup>-1</sup> Z <sub>35</sub> Z <sub>36</sub>
Round 4	Z <sub>19</sub> Z <sub>20</sub> Z <sub>21</sub> Z <sub>22</sub> Z <sub>23</sub> Z <sub>24</sub>	Z[83..128; 1..50]	U <sub>19</sub> U <sub>20</sub> U <sub>21</sub> U <sub>22</sub> U <sub>23</sub> U <sub>24</sub>	Z <sub>31</sub> <sup>-1</sup> -Z <sub>33</sub> -Z <sub>32</sub> Z <sub>34</sub> <sup>-1</sup> Z <sub>29</sub> Z <sub>30</sub>
Round 5	Z <sub>25</sub> Z <sub>26</sub> Z <sub>27</sub> Z <sub>28</sub> Z <sub>29</sub> Z <sub>30</sub>	Z[76..128; 1..43]	U <sub>25</sub> U <sub>26</sub> U <sub>27</sub> U <sub>28</sub> U <sub>29</sub> U <sub>30</sub>	Z <sub>25</sub> <sup>-1</sup> -Z <sub>27</sub> -Z <sub>26</sub> Z <sub>28</sub> <sup>-1</sup> Z <sub>23</sub> Z <sub>24</sub>
Round 6	Z <sub>31</sub> Z <sub>32</sub> Z <sub>33</sub> Z <sub>34</sub> Z <sub>35</sub> Z <sub>36</sub>	Z[44..75; 101..128; 1..36]	U <sub>31</sub> U <sub>32</sub> U <sub>33</sub> U <sub>34</sub> U <sub>35</sub> U <sub>36</sub>	Z <sub>19</sub> <sup>-1</sup> -Z <sub>21</sub> -Z <sub>20</sub> Z <sub>22</sub> <sup>-1</sup> Z <sub>17</sub> Z <sub>18</sub>
Round 7	Z <sub>37</sub> Z <sub>38</sub> Z <sub>39</sub> Z <sub>40</sub> Z <sub>41</sub> Z <sub>42</sub>	Z[37..100; 126..128; 1..29]	U <sub>37</sub> U <sub>38</sub> U <sub>39</sub> U <sub>40</sub> U <sub>41</sub> U <sub>42</sub>	Z <sub>13</sub> <sup>-1</sup> -Z <sub>15</sub> -Z <sub>14</sub> Z <sub>16</sub> <sup>-1</sup> Z <sub>11</sub> Z <sub>12</sub>
Round 8	Z <sub>43</sub> Z <sub>44</sub> Z <sub>45</sub> Z <sub>46</sub> Z <sub>47</sub> Z <sub>48</sub>	Z[30..125]	U <sub>43</sub> U <sub>44</sub> U <sub>45</sub> U <sub>46</sub> U <sub>47</sub> U <sub>48</sub>	Z <sub>7</sub> <sup>-1</sup> -Z <sub>9</sub> -Z <sub>8</sub> Z <sub>10</sub> <sup>-1</sup> Z <sub>5</sub> Z <sub>6</sub>
transformation	Z <sub>49</sub> Z <sub>50</sub> Z <sub>51</sub> Z <sub>52</sub>	Z[23..86]	U <sub>49</sub> U <sub>50</sub> U <sub>51</sub> U <sub>52</sub>	Z <sub>1</sub> <sup>-1</sup> -Z <sub>2</sub> -Z <sub>3</sub> Z <sub>4</sub> <sup>-1</sup>

IDEA Encryption and Decryption Subkeys

- $Z_j^{-1}$ : multiplicative inverse;  $Z_j \odot Z_j^{-1} = 1$
- $-Z_j$ : additive inverse;  $-Z_j \boxplus Z_j = 0$

Today, there are hundreds of IDEA-based security solutions available in many market areas, ranging from Financial Services, and Broadcasting to Government. The

IDEA algorithm can easily be embedded in any encryption software. Data encryption can be used to protect data transmission and storage. Typical fields are:

- Audio and video data for cable TV, pay TV, video conferencing, distance learning
- Sensitive financial and commercial data
- Email via public networks
- Smart cards

## BLOWFISH Algorithm

Blowfish is a symmetric block cipher that can be effectively used for encryption and safeguarding of data. It takes a variable-length key, from 32 bits to 448 bits. Blowfish was designed in 1993 by **Bruce Schneier** as a fast, free alternative to existing encryption algorithms. Blowfish is unpatented and license-free, and is available free for all uses. Blowfish *Algorithm* is a **Feistel Network**, iterating a simple encryption function 16 times. The block size is 64 bits, and the key can be any length up to 448 bits. Although, there is a complex initialization phase required before any encryption can take place, the actual encryption of data is very efficient on large microprocessors.

Blowfish is designed to have the following characteristics:

- **Fast:** Blowfish encrypts data on 32-bit microprocessors at a rate of 18 clock cycles per byte.
- **Compact:** Blowfish can run in less than 5k of memory.
- **Simple:** Blowfish's simple structure is easy to implement and eases the task of determining the strength of algorithm.
- **Variably Secure:** The key length is variable and can be as long as 448 bits. This allows a tradeoff between higher speed and higher security.

Blowfish encrypts 64-bit blocks of plaintext into 64-bit blocks of ciphertext. Blowfish uses a key that ranges from 32-bits to 448 bits. That key is used to generate 18 32-bit subkeys and four 8\*32 S-boxes containing a total of 1024 32-bit entries. The total is 1042 32-bit values, or 4168 bytes. The keys are stored in a K-array.

$$K_1, K_2, \dots, K_j \quad 1 \leq j \leq 14$$

The 18 32-bit subkeys are stored in the P-array:

$$P_1, P_2, \dots, P_{18}$$

There are 4 S-boxes, each with 8x32(=256) 32-bit entries

$$\begin{array}{l} S_{1,0}, S_{1,1}, \dots, S_{1,255} \\ S_{2,0}, S_{2,1}, \dots, S_{2,255} \\ S_{3,0}, S_{3,1}, \dots, S_{3,255} \\ S_{4,0}, S_{4,1}, \dots, S_{4,255} \end{array}$$

Steps in generation of P-array and S-boxes are as follows:



- P-array and then 4 S-boxes are initialized with fractional part of  $\pi$ :

$$\begin{aligned} P_1 &= 243F6A88_{16} \\ P_2 &= 85A308D3_{16} \\ &\dots \\ S_{4,254} &= 578FDFE3_{16} \\ S_{4,255} &= 3AC372E6_{16} \end{aligned}$$

- P-array is XORed with K-array (reusing K-array if necessary):

$$\begin{aligned} P_1 &= P_1 \oplus K_1, P_2 = P_2 \oplus K_2, \dots, P_j = P_j \oplus K_j, P_{j+1} = P_{j+1} \oplus K_1, \\ P_{j+2} &= P_{j+2} \oplus K_2, \dots \end{aligned}$$

- Encrypt the 64-bit block of all zeros using the current P- and S- arrays; replace  $P_1$  and  $P_2$  with the output of the encryption.
- Encrypt the output of step 3 using the current P- and S- arrays and replace  $P_3$  and  $P_4$  with the resulting ciphertext.
- Continue this process to update all elements of P and then ,in order, all elements of S, using at each step the output of the continuously changing Blowfish algorithm.
- Then update process of P-array and S-boxes is summarized as follows:

$$\begin{aligned} P_1, P_2 &= E_{P,S}[0] \\ P_3, P_4 &= E_{P,S}[P_1 \parallel P_2] \\ &\dots \\ P_{17}, P_{18} &= E_{P,S}[P_{15} \parallel P_{16}] \\ S_{1,0}, S_{1,1} &= E_{P,S}[P_{17} \parallel P_{18}] \\ &\dots \\ S_{4,254}, S_{4,255} &= E_{P,S}[P_{4,252} \parallel P_{4,253}] \end{aligned}$$

Where  $E_{P,S}[Y]$  is the ciphertext produced by encrypting Y using Blowfish with the P and S arrays.

A total of 521 executions in total are required to produce the final P and S arrays. Accordingly blowfish is not suitable for applications in which the secret key changes frequently. Furthermore, for rapid execution, the P- and S- arrays can be stored rather than rederived from the key each time the algorithm is used which requires upto 4kb of memory, making blowfish unsuitable for applications with limited memory like smartcards.

### **Blowfish Encryption/Decryption:**

Blowfish uses two primitive operations, which do not commute making cryptanalysis difficult:

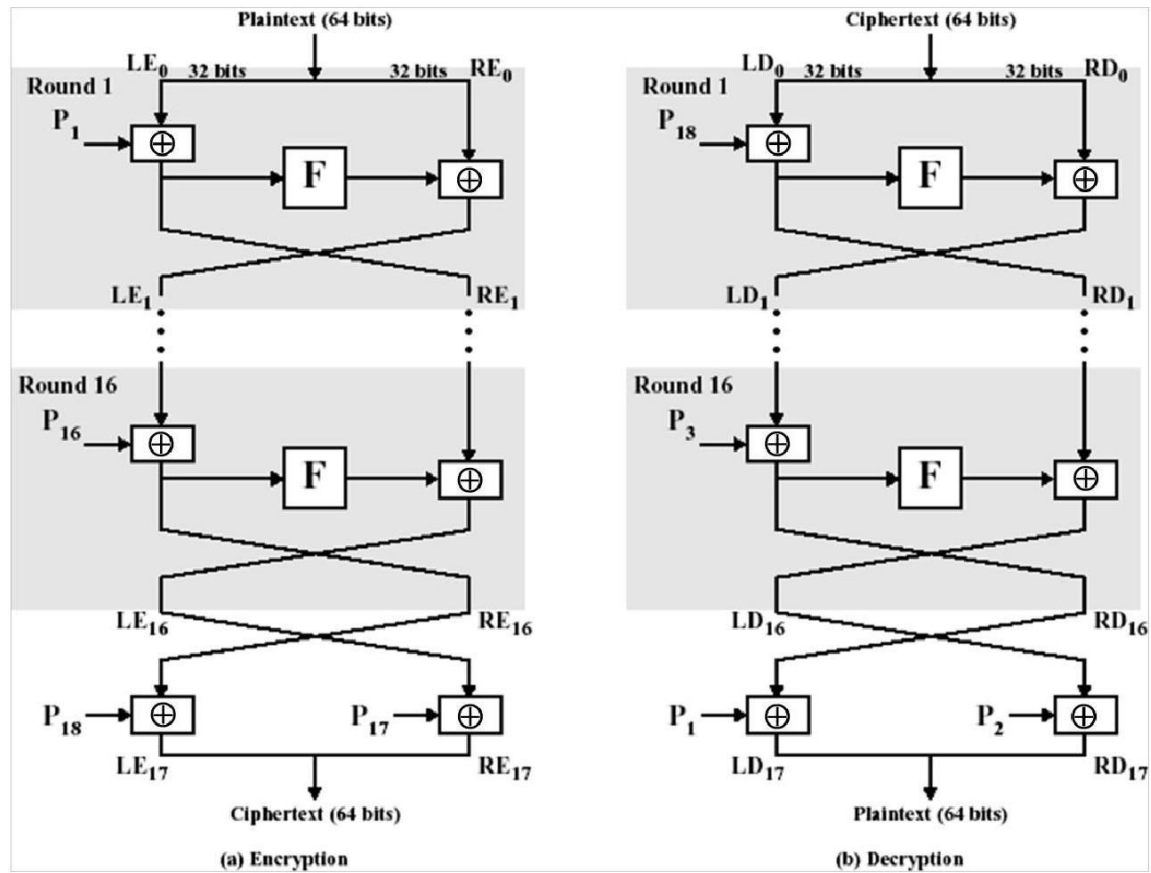
- Addition:- Addition of words, denoted by +, is performed modulo  $2^{32}$
- Bitwise exclusive-OR: This operation is denoted by  $\oplus$ .

The structure is a slight variant of classic Feistel network

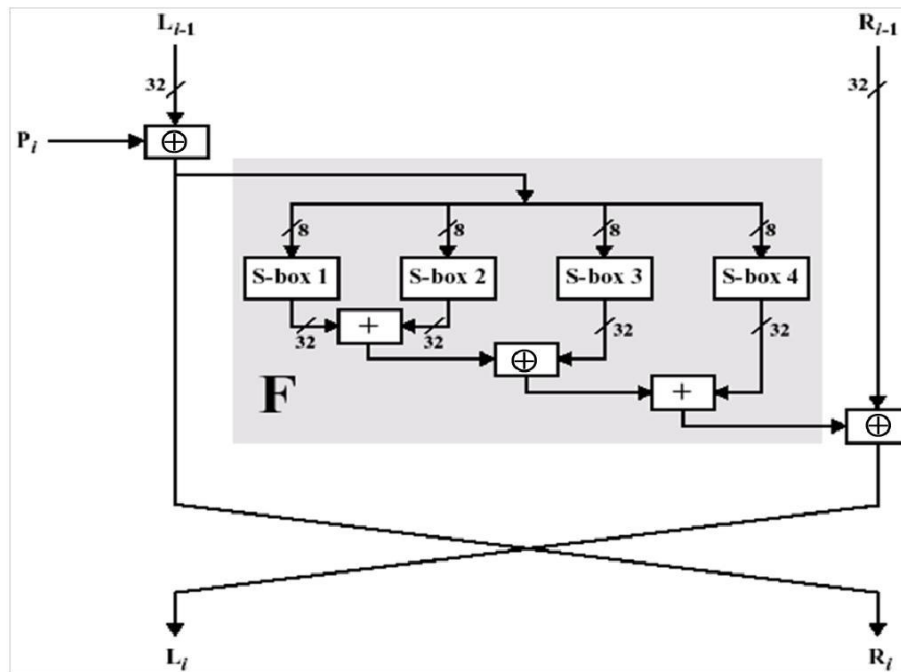
- L and R are both processed in each round

- 16 rounds
- Two extra XORs at the end

The plain text is divided into two 32-bit halves  $LE_0$  and  $RE_0$ . The resulting ciphertext is contained in the two variables  $LE_{17}$  and  $RE_{17}$ .



The function  $F$  is shown below:



The 32-bit input to F is divided into 4 bytes. If they are labelled a,b,c,d then the function can be defined as

$$F(a, b, c, d) = ((S_{1,a} + S_{2,b}) \oplus S_{3,c}) + S_{4,d}$$

Thus, each round includes the complex used of addition modulo  $2^{32}$  and XOR, plus substitution using S-boxes. Decryption of Blowfish is easily derived from the encryption algorithm. It involves using the subkeys in reverse order. Unlike most block ciphers, blowfish decryption occurs in the same algorithmic direction as encryption rather than the reverse.

Some main characteristics of Blowfish are:

- ☐ Key-dependent S-Boxes
- ☐ Operations are performed on both halves of data
- ☐ Time-consuming subkey generation process: Makes it bad for rapid key switching, but makes brute force expensive
- ☐ Perfect avalanche effect because of the function F
- ☐ Fast

# Advanced Encryption Standard

AES is a symmetric block cipher that is intended to replace DES as the approved standard for a wide range of applications. The drawbacks of 3DES being it is very slow and also it uses 64-bit block size same as DES. For reasons of both efficiency and security, a larger key size is desirable. So, NIST (National Institute of Standards and Technology) has called for proposals for a new AES, which should have security strength equal to or better than 3DES and significantly, improved efficiency. NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.

Out of all the algorithms that were submitted, five were shortlisted and upon final evaluation, NIST selected **Rijndael** as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: **Dr. Joan Daemen and Dr. Vincent Rijmen**.

## AES Evaluation:

- There are three main categories of criteria used by NIST to evaluate potential candidates.
- Security: Resistance to cryptanalysis, soundness of math, randomness of output, etc
- Cost: Computational efficiency (speed), Memory requirements
- Algorithm/Implementation Characteristics: Flexibility, hardware and software suitability, algorithm simplicity

## Simplified AES

The encryption algorithm takes a 16-bit block of plaintext as input and a 16-bit key and produces a 16-bit block of ciphertext as output. The S-AES decryption algorithm takes a 16-bit block of ciphertext and the same 16-bit key used to produce that ciphertext as input and produces the original 16-bit block of plaintext as output. The encryption algorithm involves the use of four different functions, or transformations: add key ( $A_K$ ) nibble substitution (NS), shift row (SR), and mix column (MC).

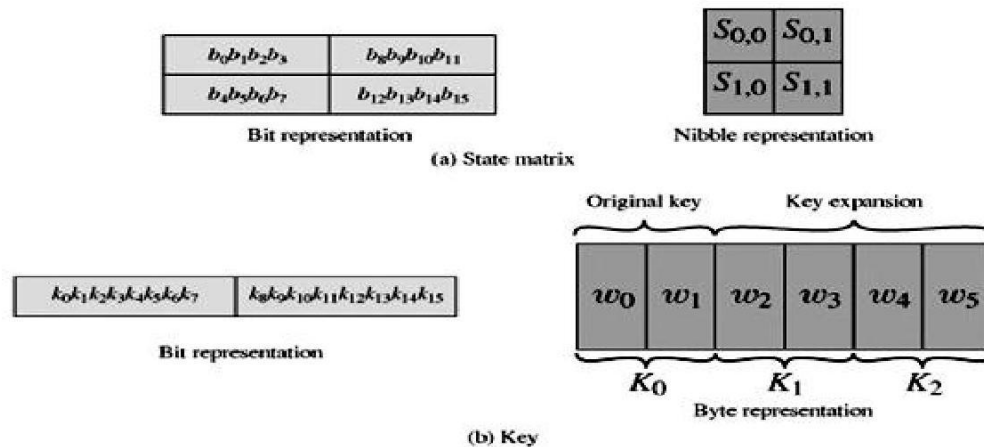
The encryption algorithm can be expressed as:

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}, \text{ so that } A_{K_0} \text{ is applied first.}$$

The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of four functions; and round 2 contains only 3 functions. Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key. S- AES encryption and decryption scheme is shown below.

Each function operates on a 16-bit state, treated as a 2 x 2 matrix of nibbles, where one nibble equals 4 bits. The initial value of the state matrix is the 16-bit plaintext; the state matrix is modified by each subsequent function in the encryption process, producing after the last function the 16-bit ciphertext. The following figure shows the ordering of nibbles within the matrix is by column. So, for example, the first eight bits of a 16-bit plaintext input to the encryption cipher occupy the first column of the matrix, and the second eight bits occupy the second column. The 16-bit key is similarly organized, but it is somewhat more convenient to view the key as two bytes rather than four nibbles. The expanded key of 48 bits is treated as three round keys, whose bits are labelled as follows:  $K_0 = k_0 \dots k_{15}$ ;  $K_1 = k_{16} \dots k_{31}$ ;  $K_2 = k_{32} \dots k_{47}$ .

### S-AES Data Structures

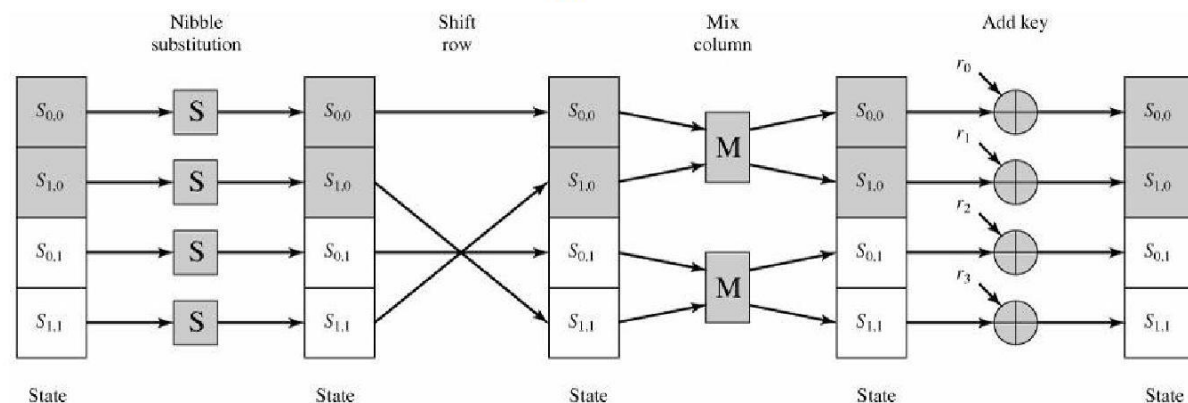


The following figure shows the essential elements of a full round of S-AES. The decryption as shown above can be given as:

$$A_{K_0} \circ \text{INS} \circ \text{ISR} \circ \text{IMC} \circ A_{K_1} \circ \text{INS} \circ \text{ISR} \circ A_{K_2}$$

in which three of the functions have a corresponding inverse function: inverse nibble substitution (INS), inverse shift row (ISR), and inverse mix column (IMC).

### S-AES Encryption Round



## S-AES Encryption and Decryption

The individual functions that are part of the encryption algorithm are given below.

### Add Key

$$\begin{array}{|c|c|} \hline A & 4 \\ \hline 7 & 9 \\ \hline \end{array} \oplus \begin{array}{|c|c|} \hline 2 & 5 \\ \hline D & 5 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 8 & 1 \\ \hline A & C \\ \hline \end{array}$$

state matrix                      key

The add key function consists of the bitwise XOR of the 16-bit state matrix and the 16-bit round key. As shown in the above example, it can also be viewed as a nibble-wise or bitwise operation. The inverse of the add key function is identical to the add key function, because the XOR operation is its own inverse.

### Nibble Substitution

The nibble substitution function is a simple table lookup. AES defines a 4 x 4 matrix of nibble values, called an S-box that contains a permutation of all possible 4-bit values. Each individual nibble of the state matrix is mapped into a new nibble in the following way: The leftmost 2 bits of the nibble are used as a row value and the rightmost 2 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 4-bit output value. For example, the hexadecimal value A references row 2, column 2 of the S-box, which contains the value 0. Accordingly, the value A is mapped into the value 0.

### S-AES S-Boxes

		<i>j</i>			
		00	01	10	11
<i>i</i>	00	9	4	A	B
	01	D	1	8	5
	10	6	2	0	3
	11	C	E	F	7

(a) S-Box

		<i>j</i>			
		00	01	10	11
<i>i</i>	00	A	5	9	B
	01	1	7	8	F
	10	6	0	2	3
	11	C	4	D	E

(b) Inverse S-Box

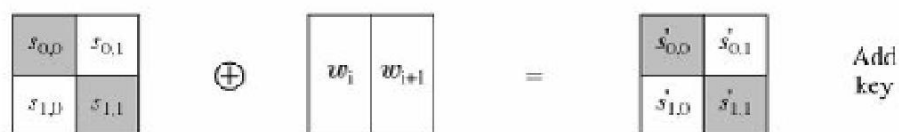
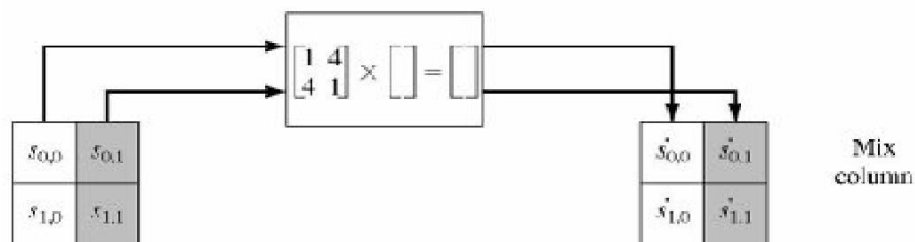
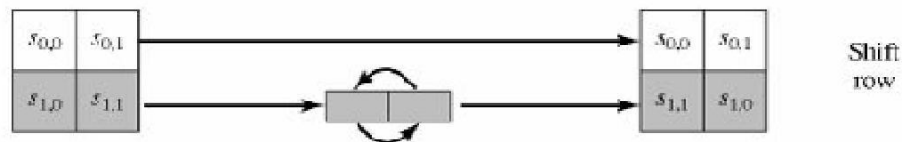
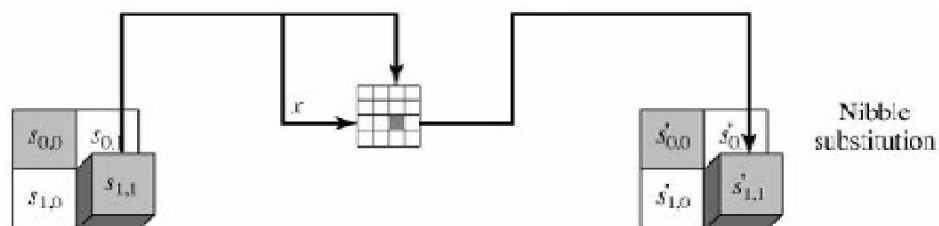
For the example, after nibble substitution, the output is

8	1
A	C

 $\rightarrow$ 

6	4
0	C

### S-AES Transformations





### Shift Row

The shift row function performs a one-nibble circular shift of the second row of the state matrix; the first row is not altered. Our example is shown below:

6	4
0	C

→

6	4
C	0

The inverse shift row function is identical to the shift row function, because it shifts the second row back to its original position.

### Mix Column

The mix column function operates on each column individually. Each nibble of a column is mapped into a new value that is a function of both nibbles in that column. The transformation can be defined by the following matrix multiplication on the state matrix.

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

Where arithmetic is performed in  $GF(2^4)$ , and the symbol  $\cdot$  refers to multiplication in  $GF(2^4)$ . The example is shown below:

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as follows:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

### Key Expansion

For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. The following figure shows the expansion into 6 words, by the calculation of 4 new words from the initial 2 words. The algorithm is as follows:

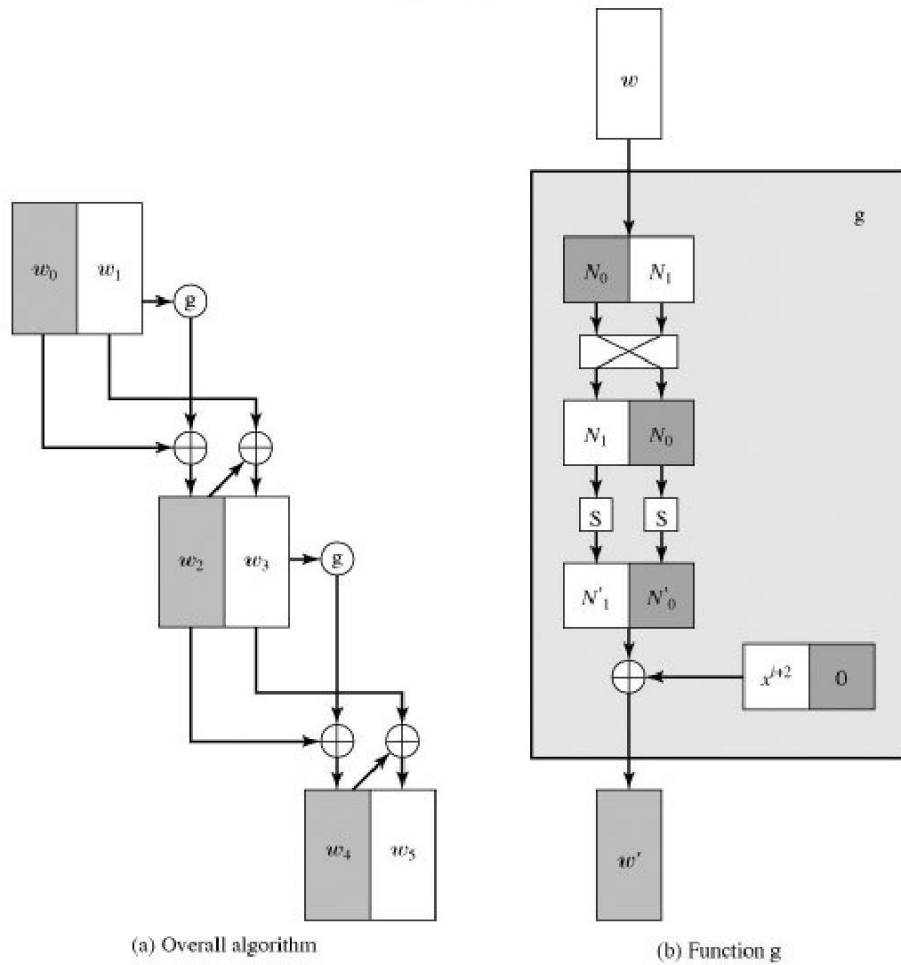
$$w_2 = w_0 \oplus g(w_1) = w_0 \oplus \text{RCON}(1) \oplus \text{SubNib}(\text{RotNib}(w_1))$$

$$w_3 = w_2 \oplus w_1$$

$$w_4 = w_2 \oplus g(w_3) = w_2 \oplus \text{RCON}(2) \oplus \text{SubNib}(\text{RotNib}(w_3))$$

$$w_5 = w_4 \oplus w_3$$

### S-AES Key Expansion



RCON is a round constant, defined as follows:  $\text{RC}[i] = x^{i+2}$ , so that  $\text{RC}[1] = x^3 = 1000$  and  $\text{RC}[2] = x^4 \bmod (x^4 + x + 1) = x + 1 = 0011$ .  $\text{RC}[i]$  forms the leftmost nibble of a byte, with the rightmost nibble being all zeros. Thus,  $\text{RCON}(1) = 10000000$  and  $\text{RCON}(2) = 00110000$ .

For example, suppose the key is  $2D55 = 0010\ 1101\ 0101\ 0101 = w_0w_1$ . Then,

$$\begin{aligned}
 w_2 &= 00101101 \oplus 10000000 \oplus \text{SubNib}(01010101) \\
 &= 00101101 \oplus 10000000 \oplus 00010001 = 10111100 \\
 w_3 &= 10111100 \oplus 01010101 = 11101001 \\
 w_4 &= 10111100 \oplus 00110000 \oplus \text{SubNib}(10011110) \\
 &= 10111100 \oplus 00110000 \oplus 00101111 = 10100011 \\
 w_5 &= 10100011 \oplus 11101001 = 01001010
 \end{aligned}$$

### *The S-Box*

The S-box is constructed as follows:

7. Initialize the S-box with the nibble values in ascending sequence row by row. The first row contains the hexadecimal values 0, 1, 2, 3; the second row contains 4, 5, 6, 7; and so on. Thus, the value of the nibble at row  $i$ , column  $j$  is  $4i + j$ .
8. Treat each nibble as an element of the finite field  $GF(2^4)$  modulo  $x^4 + x + 1$ . Each nibble  $a_0a_1a_2a_3$  represents a polynomial of degree 3.
9. Map each byte in the S-box to its multiplicative inverse in the finite field  $GF(2^4)$  modulo  $x^4 + x + 1$ ; the value 0 is mapped to itself.
10. Consider that each byte in the S-box consists of 4 bits labeled  $(b_0, b_1, b_2, b_3)$ . Apply the following transformation to each bit of each byte in the S-box: The AES standard depicts this transformation in matrix form as follows:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The prime (') indicates that the variable is to be updated by the value on the right. Remember that addition and multiplication are being calculated modulo 2.

# The AES Cipher

---

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. The number of rounds is dependent on the key size i.e. for key sizes of **128/192/256 bits**, the number of rounds are **10/12/14**. AES is an iterated cipher (rather than Feistel cipher) as it processes data as block of 4 columns of 4 bytes and operates on entire data block in every round.

Rijndael was designed to have the following characteristics:

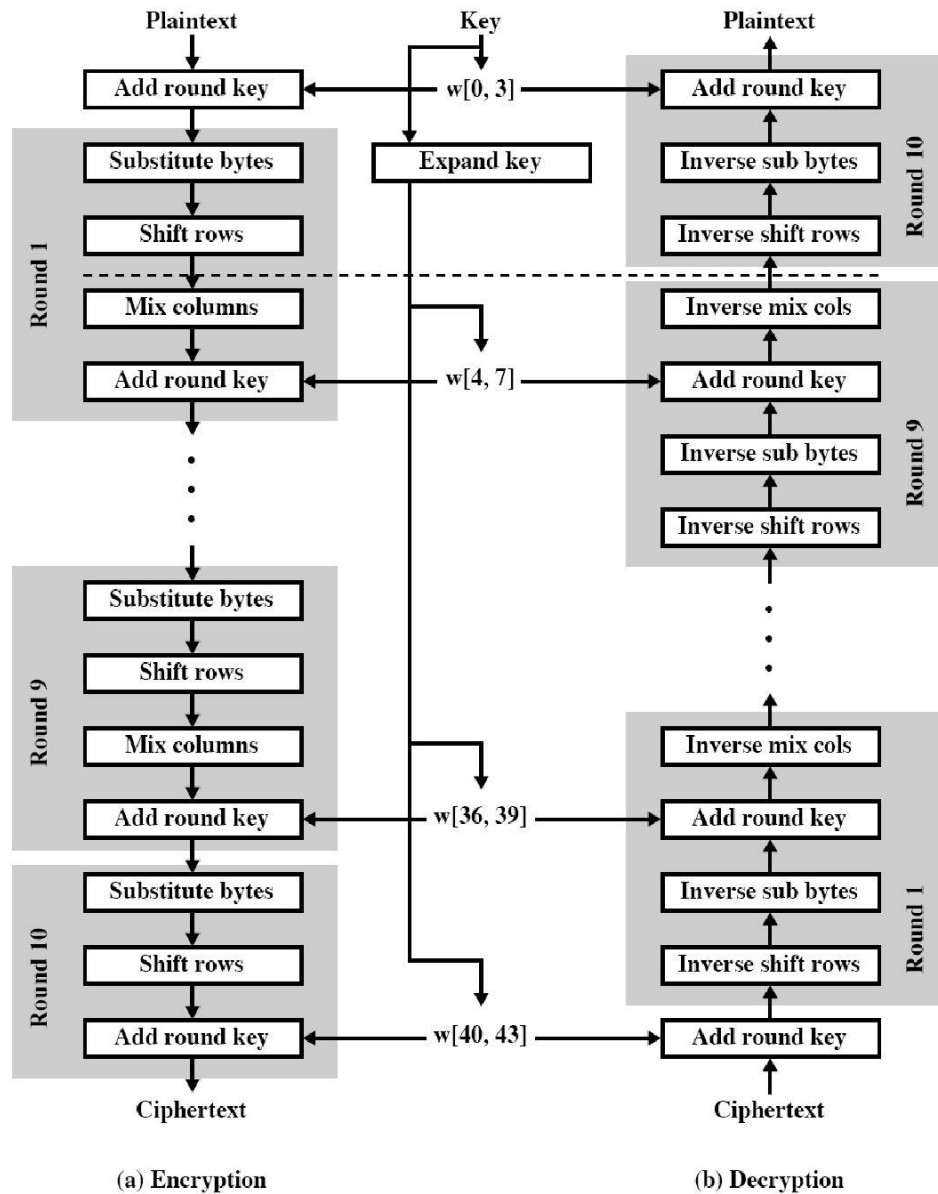
- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. In the same way, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key.

- ❑ The key that is provided as input is expanded into an array of forty-four 32-bit words,  $w[i]$ . Four distinct words (128 bits) serve as a round key for each round; these are indicated in above figure.
- ❑ Four different stages are used, one of permutation and three of substitution:
  - I. Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
  - II. ShiftRows: A simple permutation
  - III. MixColumns: A substitution that makes use of arithmetic over  $GF(2^8)$
  - IV. AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key
- 3. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. The following figure depicts the structure of a full encryption round.
- 4. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
- 5. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view

the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.

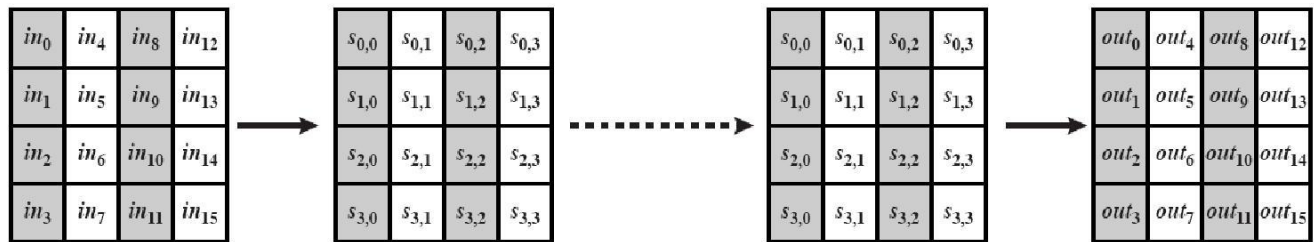
### AES Encryption and Decryption



6. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that
7. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

8. Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. AES structure figure lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.
9. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

### AES Data Structures



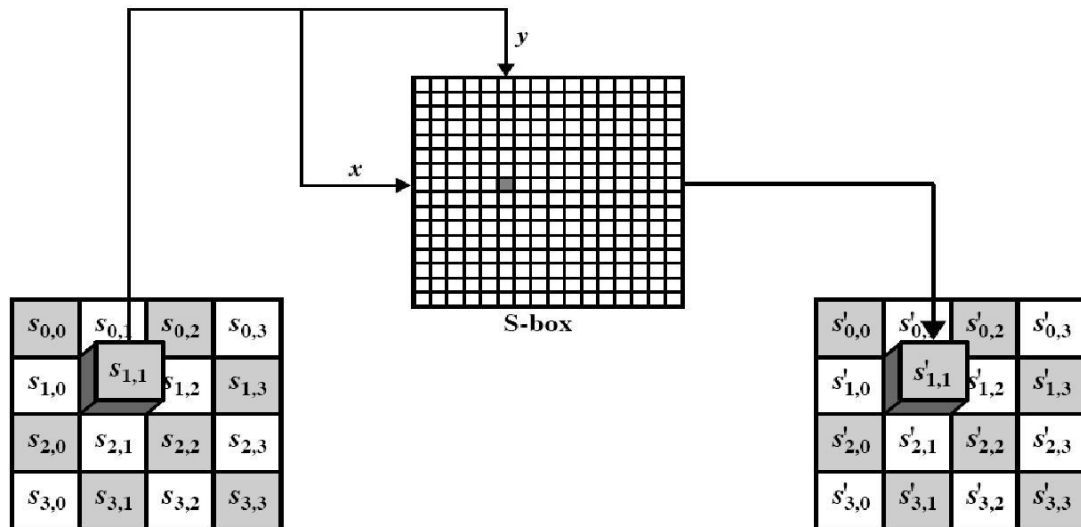
(a) Input, state array, and output



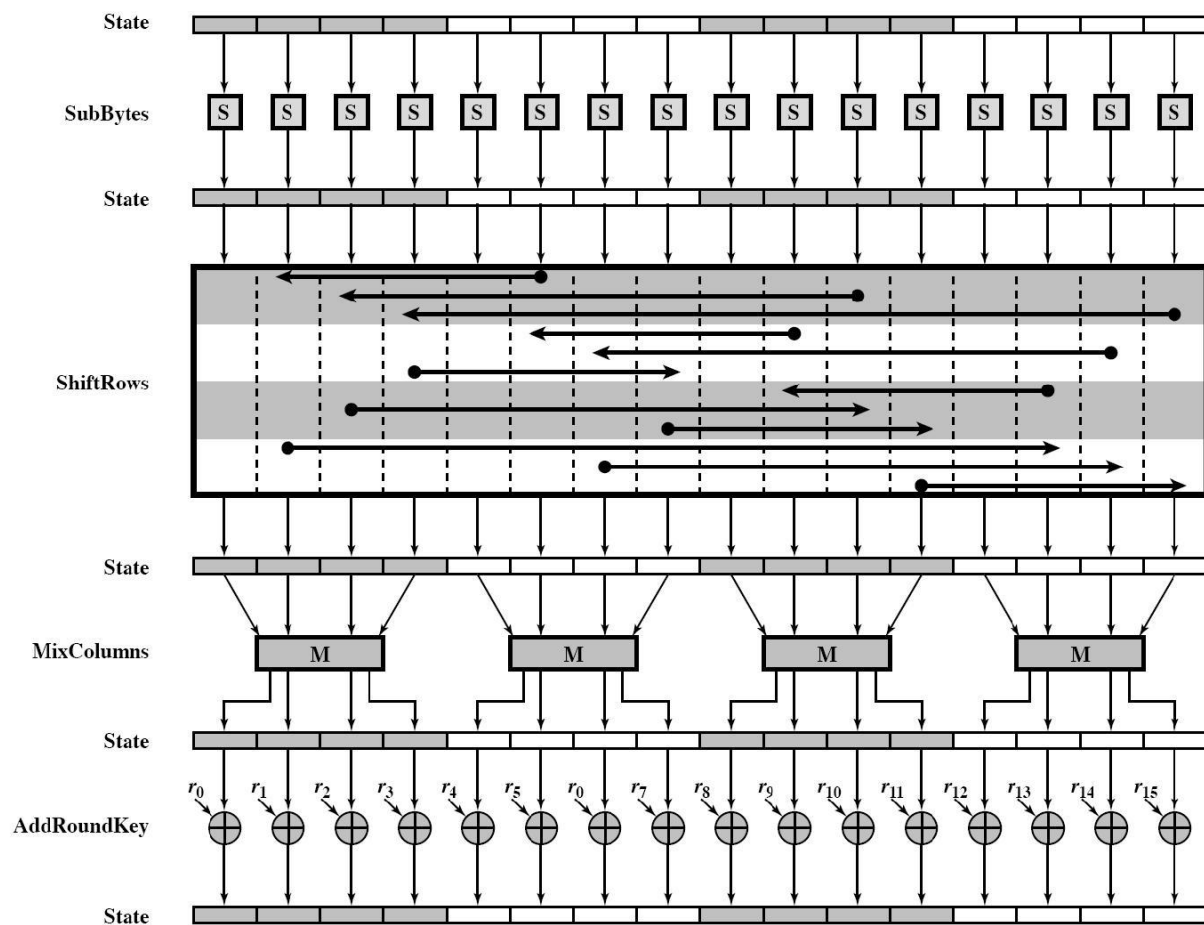
(b) Key and expanded key

### Substitute Bytes Transformation:

- The forward substitute byte transformation, called **subBytes** is a simple table look up.
- Simple substitution on each byte of state independently
- Uses an S-box of 16x16 bytes containing a permutation of all 256 8-bit values
- The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value.
- S-box constructed using defined transformation of values in  $GF(2^8)$  and is designed to be resistant to all known attacks
- The inverse substitute byte transformation called **invSubBytes** makes use of inverse S-box



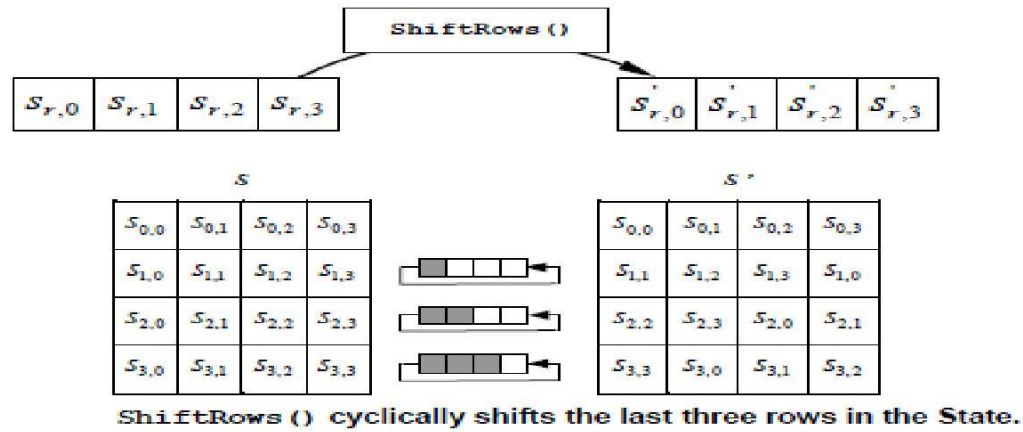
### AES Encryption Round





### ShiftRows Transformation:

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



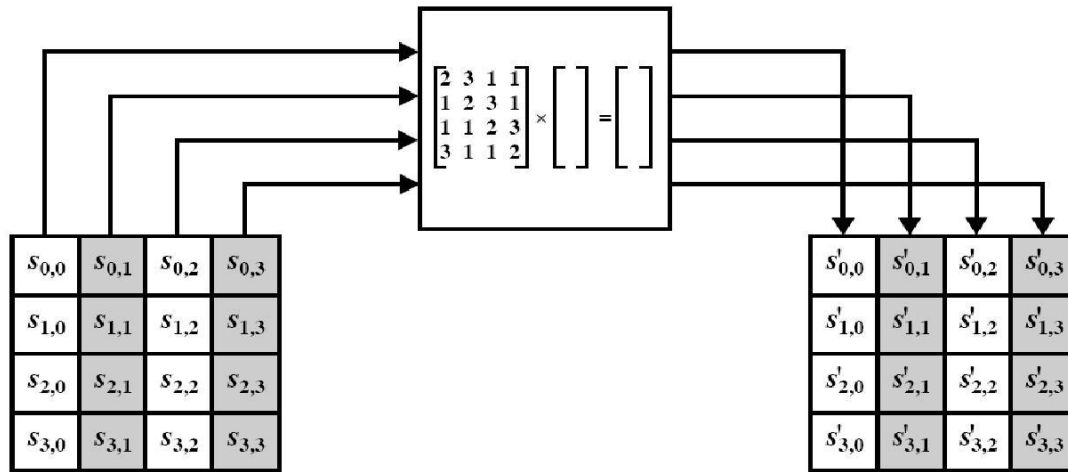
The inverse shift row transformation, called **InvShiftRows**, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

### MixColumns Transformation

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

$$\begin{aligned}
 S'_{0,c} &= (\{02\} \bullet S_{0,c}) \oplus (\{03\} \bullet S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\
 S'_{1,c} &= S_{0,c} \oplus (\{02\} \bullet S_{1,c}) \oplus (\{03\} \bullet S_{2,c}) \oplus S_{3,c} \\
 S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus (\{02\} \bullet S_{2,c}) \oplus (\{03\} \bullet S_{3,c}) \\
 S'_{3,c} &= (\{03\} \bullet S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \bullet S_{3,c})
 \end{aligned}$$

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$



Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in  $GF(2^8)$  using irreducible polynomial  $m(x) = x^8 + x^4 + x^3 + x + 1$ . The inverse mix column transformation, called **InvMixColumns**, is defined by the following matrix multiplication:

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

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### **AddRoundKey Transformation**

In the **forward add round key transformation**, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As shown below, the operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation.

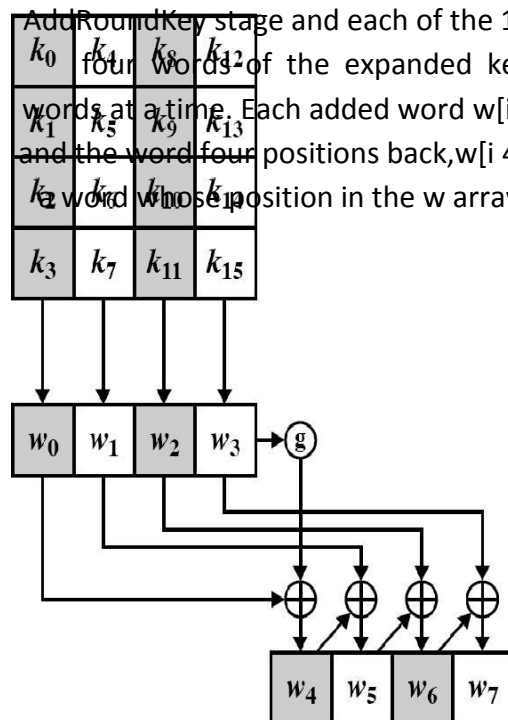
$$\begin{array}{|c|c|c|c|} \hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \oplus \begin{array}{|c|c|c|c|} \hline w_i & w_{i+1} & w_{i+2} & w_{i+3} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ \hline s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ \hline s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ \hline s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \\ \hline \end{array}$$

The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

### AES Key Expansion

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word  $w[i]$  depends on the immediately preceding word,  $w[i-1]$ , and the word four positions back,  $w[i-4]$ . In three out of four cases, a simple XOR is used. For a word whose position in the  $w$  array is a multiple of 4, a more complex function  $g$  is used.

The function  $g$  consists

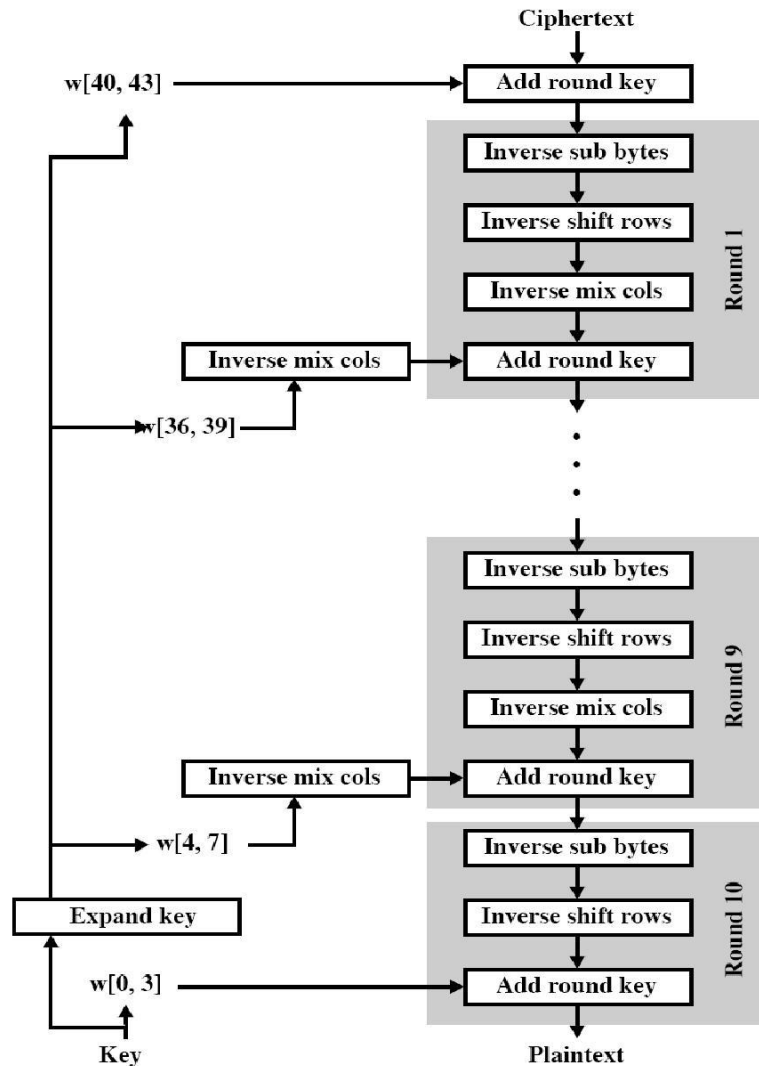


of the following subfunctions

RotWord performs a one-byte circular left shift on a word. This means that an input word  $[b_0, b_1, b_2, b_3]$  is transformed into  $[b_1, b_2, b_3, b_0]$ .

SubWord performs a byte substitution on each byte of its input word, using the S-box

- The result of steps 1 and 2 is XORed with a round constant,  $Rcon[j]$ .
- The round constant is a word in which the three rightmost bytes are always 0.
- The effect of an XOR of a word with  $Rcon$  is to only perform an XOR on the leftmost byte of the word.
- The round constant is different for each round and is defined as  $Rcon[j] = (RC[j], 0, 0, 0)$ , with  $RC[1] = 1$ ,  $RC[j] = 2 \cdot RC[j-1]$  and with multiplication defined over the field  $GF(2^8)$ .



### AES Decryption

1. AES decryption is not identical to encryption since steps done in reverse
2. But can define an equivalent inverse cipher with steps as for encryption
3. But using inverses of each step
4. With a different key schedule
5. Works since result is unchanged when
6. Swap byte substitution & shift rows
7. Swap mix columns & add (tweaked) round key

### Implementation Aspects

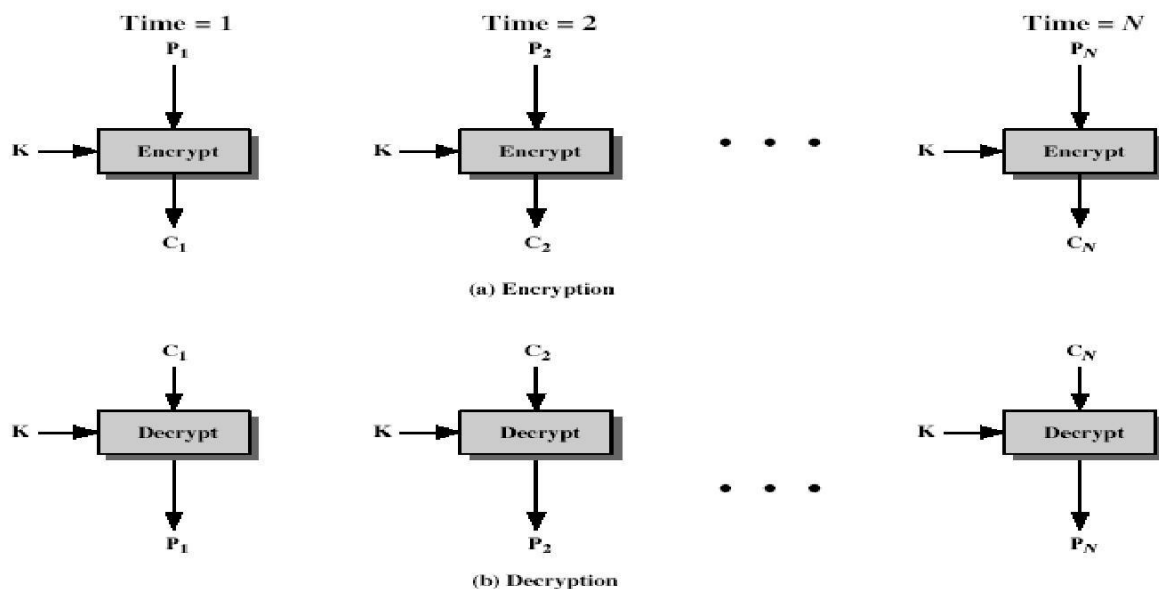
- Can efficiently implement on 8-bit CPU
- byte substitution works on bytes using a table of 256 entries
- shift rows is simple byte shift
- add round key works on byte XOR's
- mix columns requires matrix multiply in  $GF(2^8)$  which works on byte values, can be simplified to use table lookups & byte XOR's
- Can efficiently implement on 32-bit CPU
- redefine steps to use 32-bit words
- can precompute 4 tables of 256-words
- then each column in each round can be computed using 4 table lookups + 4 XORs
- at a cost of 4Kb to store tables
- Designers believe this very efficient implementation was a key factor in its selection as the AES cipher

## Cipher Block modes of Operation

To apply a block cipher in a variety of applications, four “modes of operation” have been defined by NIST (FIPS 81). The four modes are intended to cover virtually all the possible applications of encryption for which a block cipher could be used. As new applications and requirements have appeared, NIST has expanded the list of recommended modes to five in Special Publication 800-38A. These modes are intended for use with any symmetric block cipher, including triple DES and AES.

### Electronic Codebook Book (ECB)

The simplest mode is the electronic codebook (ECB) mode, in which plaintext is handled one block at a time and each block of plaintext is encrypted using the same key. *ECB is the simplest of the modes, and is used when only a single block of info needs to be sent.*



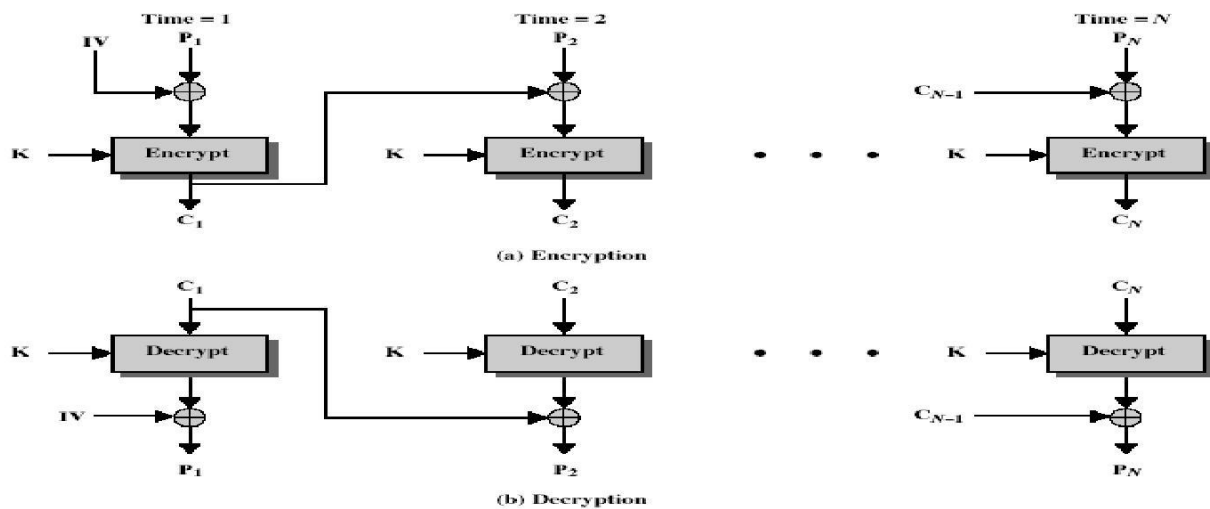
Break the plaintext into 64-bit blocks and encrypt each of them with the same key. The last block should be padded to 64-bit if it is shorter. Same block and same key always yields same cipher block. Each block is a value which is substituted, like a codebook, hence the name Electronic Code Book. Each block is encoded independently of the other blocks.

$$C_i = \text{DES}_{K1}(P_i)$$

ECB is not appropriate for any quantity of data, since repetitions can be seen, esp. with graphics, and because the blocks can be shuffled/inserted without affecting the en/decryption of each block. Its main use is to send one or a very few blocks, eg a session encryption key.

## Cipher Block Chaining Mode (CBC)

To overcome the problems of repetitions and order independence in ECB, want some way of making the ciphertext dependent on **all** blocks before it. This is what CBC gives us, by combining the previous ciphertext block with the current message block before encrypting. To start the process, use an Initial Value (IV), which is usually well known (often all 0's), or otherwise is sent, ECB encrypted, just before starting CBC use.



All cipher blocks will be chained so that if one is modified, the ciphertext cannot be decrypted correctly. Each plaintext block is XORed with the previous cipher block before encryption, hence the name CBC. The first plaintext block is XORed with an initialization vector IV, which is to be protected securely, (e.g., send it encrypted in ECB mode).

$$C_i = \text{DES}_{K1}(P_i \text{ XOR } C_{i-1})$$

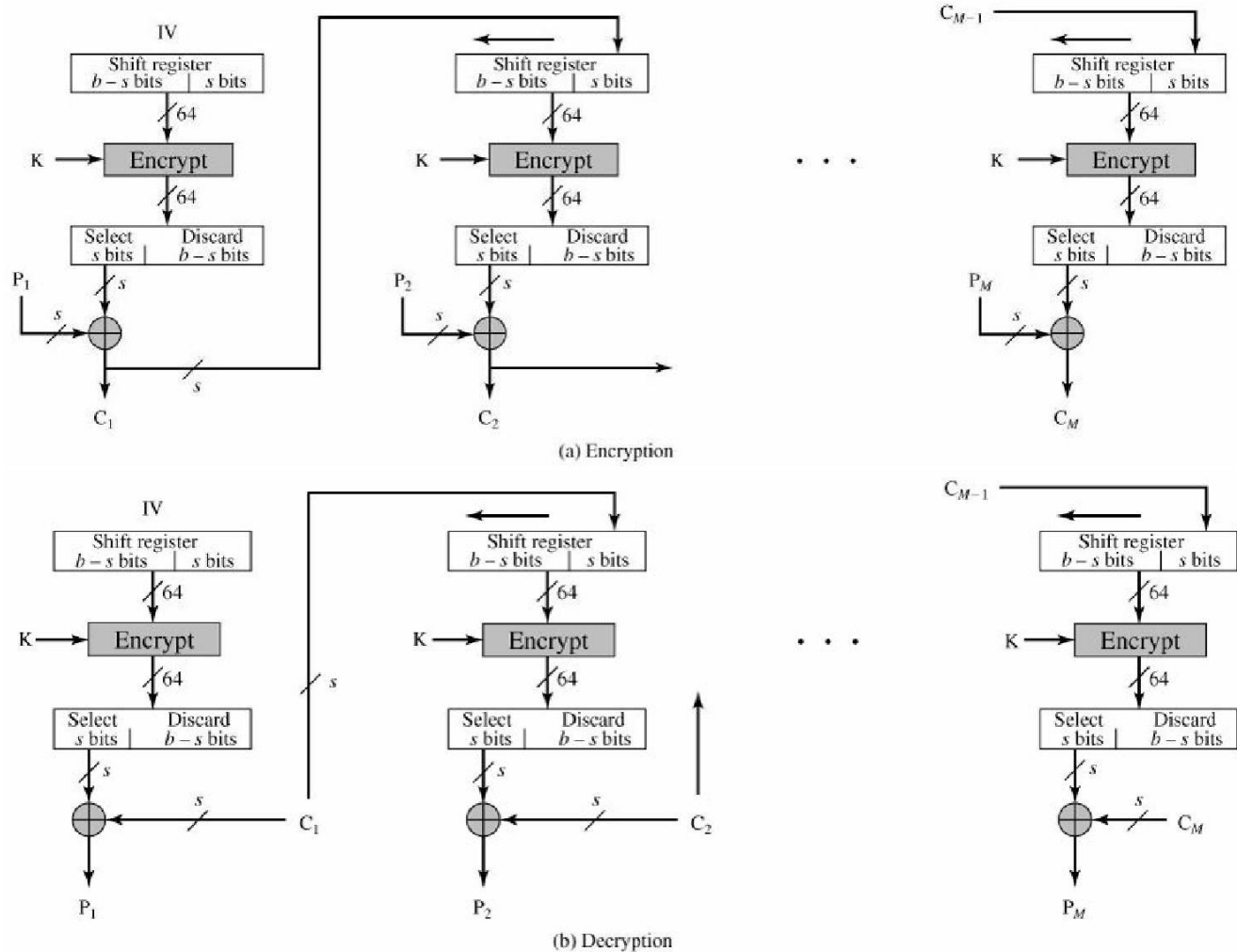
CBC is the block mode generally used. The chaining provides an avalanche effect, which means the encrypted message cannot be changed or rearranged without totally destroying the subsequent data. However there is the issue of ensuring that the IV is either fixed or sent encrypted in ECB mode to stop attacks on 1st block.

## Cipher Feed Back Mode (CFB)

If the data is only available a bit/byte at a time (eg. terminal session, sensor value etc), then must use some other approach to encrypting it, so as not to delay the info. it is possible to convert DES into a stream cipher, using either the cipher feedback (CFB) or the output feedback mode. A stream cipher eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Thus, if a character stream is being transmitted, each character can be encrypted and transmitted immediately using a character-oriented stream cipher.

One desirable property of a stream cipher is that the ciphertext be of the same length as the plaintext. Thus, if 8-bit characters are being transmitted, each character should be

encrypted to produce a cipher text output of 8 bits. If more than 8 bits are produced, transmission capacity is wasted.



The input to the encryption function is a  $b$ -bit shift register that is initially set to some initialization vector (IV). The leftmost (most significant)  $s$  bits of the output of the encryption function are XORed with the first segment of plaintext  $P_1$  to produce the first unit of ciphertext  $C_1$ , which is then transmitted. In addition, the contents of the shift register are shifted left by  $s$  bits and  $C_1$  is placed in the rightmost (least significant)  $s$  bits of the shift register. This process continues until all plaintext units have been encrypted. For decryption, the same scheme is used, except that the received ciphertext unit is XORed with the output of the encryption function to produce the plaintext unit. Note that it is the *encryption* function that is used, not the decryption function.

$$C_i = P_i \text{ XOR } \text{DES}_{K1}(C_{i-1})$$

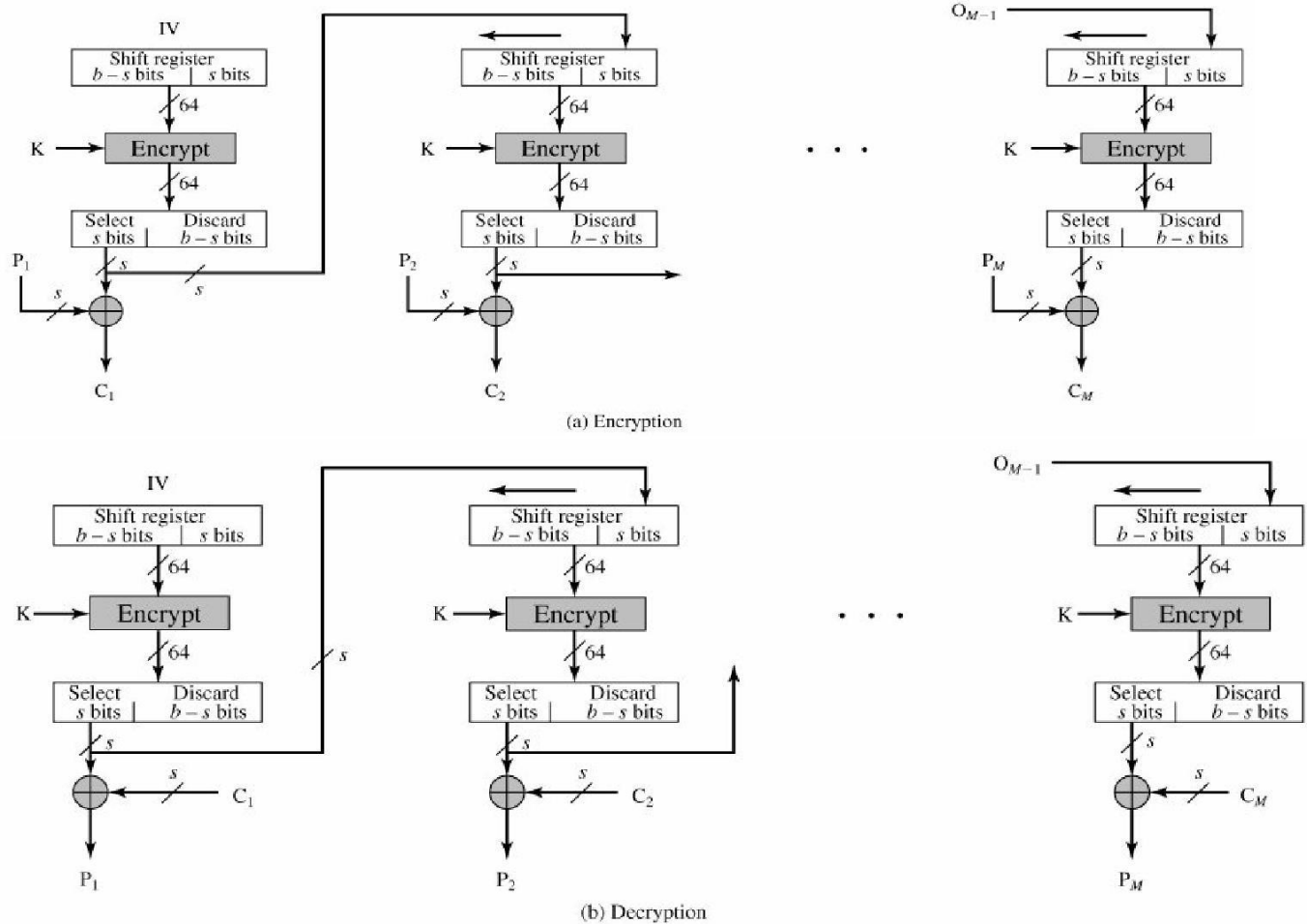
CFB is the usual stream mode. As long as can keep up with the input, doing encryptions every 8 bytes. A possible problem is that if its used over a "noisy" link, then any corrupted bit will destroy



values in the current and next blocks (since the current block feeds as input to create the random bits for the next). So either must use over a reliable network transport layer (pretty usual) or use OFB.

## Output Feedback Mode (OFB)

The output feedback (OFB) mode is similar in structure to that of CFB. It is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the ciphertext unit is fed back to the shift register.



Keystream is independent of the data and can be computed in advance.

$$C_i = P_i \text{ XOR } O_i$$

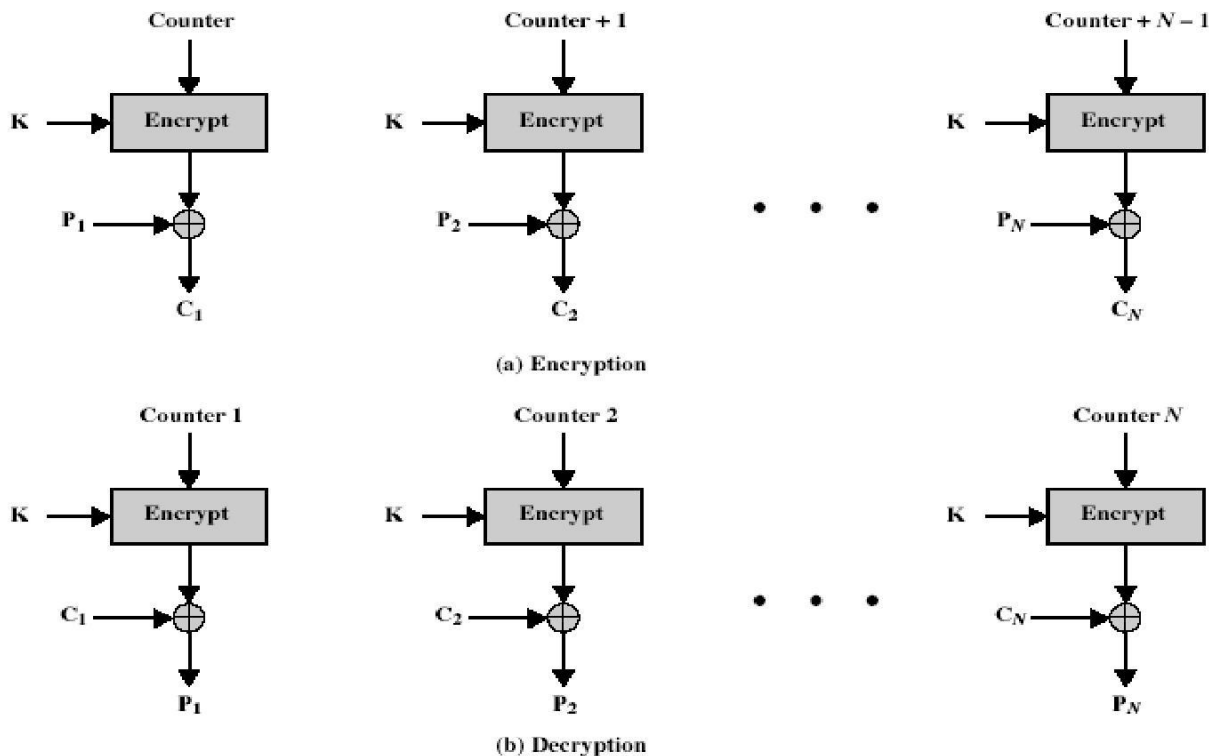
$$= \text{DES}_{K1}(O_{i-1})$$

Here the generation of the "random" bits is independent of the message being encrypted. The advantage is that firstly, they can be computed in advance, good for bursty traffic, and secondly, any bit error only affects a single bit. Thus this is good for noisy links (eg satellite TV transmissions etc). The disadvantage of OFB is that it is more vulnerable to a message stream modification attack than is CFB.

### Counter Mode (CTR)

The Counter (CTR) mode is a variant of OFB, but which encrypts a counter value (hence name). Although it was proposed many years before, it has only recently been standardized for use with AES along with the other existing 4 modes. It is being used with applications in ATM (asynchronous transfer mode) network security and IPSec (IP security).

All modes of operations except ECB make random access to the file impossible: to access data at the end of the file one has to decrypt everything. Plaintext is not encrypted directly. IV plus a constant is encrypted and the resulting ciphertext is XORed with the plaintext – add 1 to IV in each step.



If the same IV is used twice with the same key, then cryptanalyst may XOR the ciphers to get the XOR of the plaintexts –this could be used in an attack. A counter, equal to the plaintext block size is used. The only requirement stated in SP 800-38A is that the counter value must be different for each plaintext block that is encrypted. Typically the counter is initialized to some value and then incremented by 1 for each subsequent block.

CTR mode has a number of advantages in parallel h/w & s/w efficiency, can preprocess the output values in advance of needing to encrypt, can get random access to encrypted data blocks, and is simple. But like OFB have issue of not reusing the same key + counter value.

# Message Authentication

---

Message authentication is a procedure to verify that received messages come from the alleged source and have not been altered. Message authentication may also verify sequencing and timeliness. It is intended against the attacks like content modification, sequence modification, timing modification and repudiation. For repudiation, concept of digital signatures is used to counter it. There are three classes by which different types of functions that may be used to produce an authenticator. They are:

- Message encryption—the ciphertext serves as authenticator
- Message authentication code (MAC)—a public function of the message and a secret key producing a fixed-length value to serve as authenticator. This does not provide a digital signature because A and B share the same key.
- Hash function—a public function mapping an arbitrary length message into a fixed-length hash value to serve as authenticator. This does not provide a digital signature because there is no key.

## Message Encryption:

Message encryption by itself can provide a measure of authentication. The analysis differs for conventional and public-key encryption schemes. The message must have come from the sender itself, because the ciphertext can be decrypted using his (secret or public) key. Also, none of the bits in the message have been altered because an opponent does not know how to manipulate the bits of the ciphertext to induce meaningful changes to the plaintext.

- Often one needs alternative authentication schemes than just encrypting the message.
- Sometimes one needs to avoid encryption of full messages due to legal requirements.
- Encryption and authentication may be separated in the system architecture.

The different ways in which message encryption can provide authentication, confidentiality in both symmetric and asymmetric encryption techniques is explained with the table below:

### Confidentiality and Authentication Implications of Message Encryption

<p><math>A \rightarrow B: E_K[M]</math></p> <ul style="list-style-type: none"> <li>•Provides confidentiality                             <ul style="list-style-type: none"> <li>— Only A and B share <math>K</math></li> </ul> </li> <li>•Provides a degree of authentication                             <ul style="list-style-type: none"> <li>— Could come only from A</li> <li>— Has not been altered in transit</li> <li>— Requires some formatting/redundancy</li> </ul> </li> <li>•Does not provide signature                             <ul style="list-style-type: none"> <li>— Receiver could forge message</li> <li>— Sender could deny message</li> </ul> </li> </ul> <p>(a) Symmetric encryption</p>
<p><math>A \rightarrow B: E_{KU_b}[M]</math></p> <ul style="list-style-type: none"> <li>•Provides confidentiality                             <ul style="list-style-type: none"> <li>— Only B has <math>KR_b</math> to decrypt</li> </ul> </li> <li>•Provides no authentication                             <ul style="list-style-type: none"> <li>— Any party could use <math>KU_b</math> to encrypt message and claim to be A</li> </ul> </li> </ul> <p>(b) Public-key encryption: confidentiality</p>
<p><math>A \rightarrow B: E_{KR_a}[M]</math></p> <ul style="list-style-type: none"> <li>•Provides authentication and signature                             <ul style="list-style-type: none"> <li>— Only A has <math>KR_a</math> to encrypt</li> <li>— Has not been altered in transit</li> <li>— Requires some formatting/redundancy</li> <li>— Any party can use <math>KU_a</math> to verify signature</li> </ul> </li> </ul> <p>(c) Public-key encryption: authentication and signature</p>
<p><math>A \rightarrow B: E_{KU_b}[E_{KR_a}(M)]</math></p> <ul style="list-style-type: none"> <li>•Provides confidentiality because of <math>KU_b</math></li> <li>•Provides authentication and signature because of <math>KR_a</math></li> </ul> <p>(d) Public-key encryption: confidentiality, authentication, and signature</p>

## Message Authentication Code

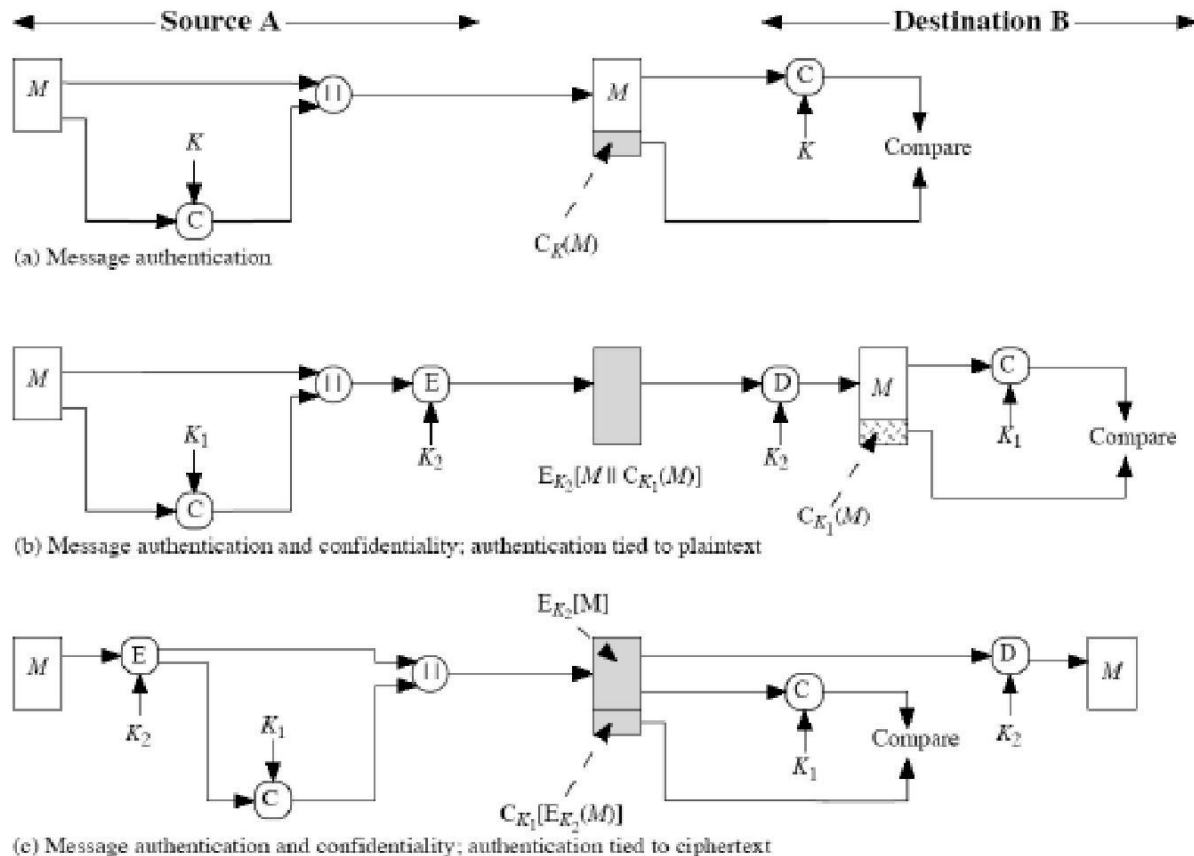
An alternative authentication technique involves the use of a secret key to generate a small fixed-size block of data, known as cryptographic checksum or MAC, which is appended to the message. This technique assumes that both the communicating parties say A and B share a common secret key  $K$ . When A has a message to send to B, it calculates MAC as a function  $C$  of key and message given as:  $MAC=C_k(M)$

The message and the MAC are transmitted to the intended recipient, who upon receiving 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 and only if they match, then:

11. The receiver is assured that the message has not been altered: Any alternations been done the MAC's do not match.

- ② The receiver is assured that the message is from the alleged sender: No one except the sender has the secret key and could prepare a message with a proper MAC.
- ② If the message includes a sequence number, then receiver is assured of proper sequence as an attacker cannot successfully alter the sequence number.

Basic uses of Message Authentication Code (MAC) are shown in the figure:



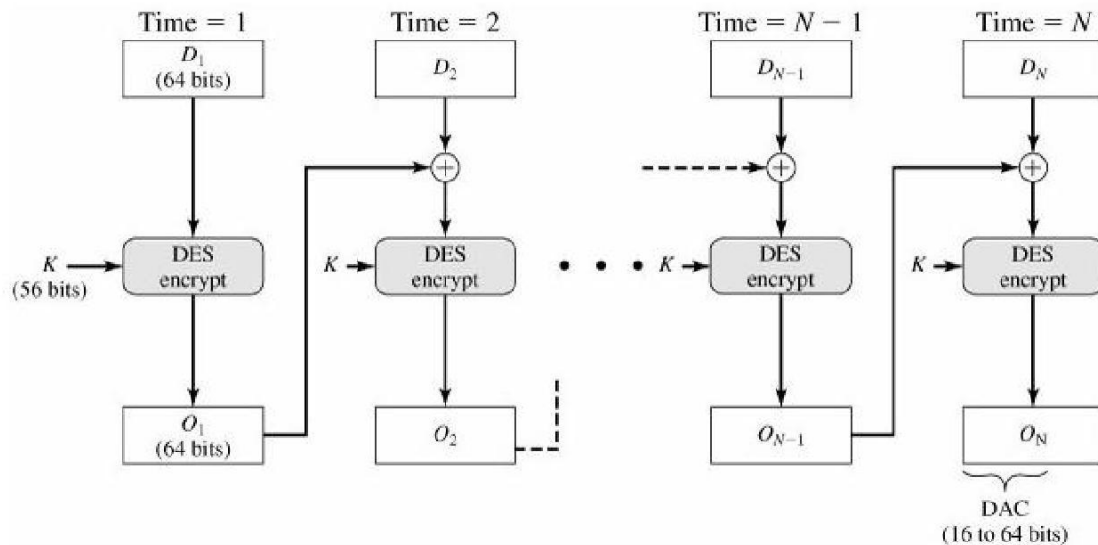
There are three different situations where use of a MAC is desirable:

- If a message is broadcast to several destinations in a network (such as a military control center), then it is cheaper and more reliable to have just one node responsible to evaluate the authenticity –message will be sent in plain with an attached authenticator.
- If one side has a heavy load, it cannot afford to decrypt all messages –it will just check the authenticity of some randomly selected messages.
- Authentication of computer programs in plaintext is very attractive service as they need not be decrypted every time wasting of processor resources. Integrity of the program can always be checked by MAC.

### Message Authentication Code Based on DES

The Data Authentication Algorithm, based on DES, has been one of the most widely used MACs for a number of years. The algorithm is both a FIPS publication (FIPS PUB 113) and an ANSI standard (X9.17). But, security weaknesses in this algorithm have been discovered and it is being replaced by newer and stronger algorithms.

The algorithm can be defined as using the cipher block chaining (CBC) mode of operation of DES shown below with an initialization vector of zero.



The data (e.g., message, record, file, or program) to be authenticated are grouped into contiguous 64-bit blocks:  $D_1, D_2, \dots, D_N$ . If necessary, the final block is padded on the right with zeroes to form a full 64-bit block. Using the DES encryption algorithm,  $E$ , and a secret key,  $K$ , a data authentication code (DAC) is calculated as follows:

$$O_1 = E(K, D_1)$$

$$O_2 = E(K, [D_2 \oplus O_1])$$

$$O_3 = E(K, [D_3 \oplus O_2])$$

•

•

•

$$O_N = E(K, [D_N \oplus O_{N-1}])$$

The DAC consists of either the entire block  $O_N$  or the leftmost  $M$  bits of the block, with  $16 \leq M \leq 64$

Use of MAC needs a shared secret key between the communicating parties and also MAC does not provide digital signature. The following table summarizes the confidentiality and authentication implications of the approaches shown above.

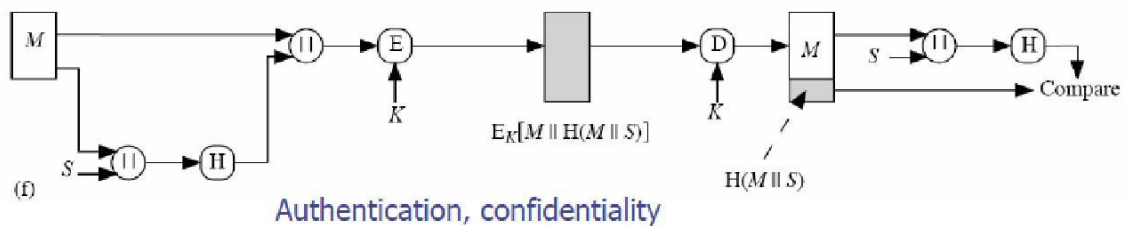
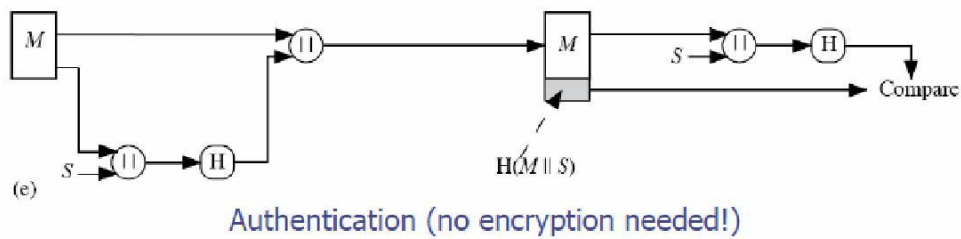
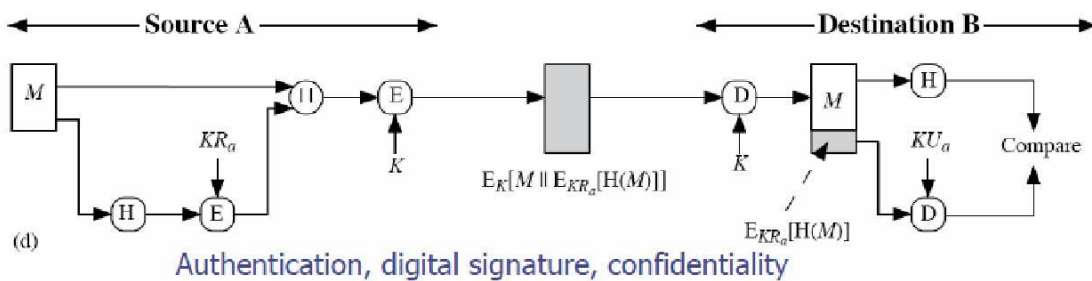
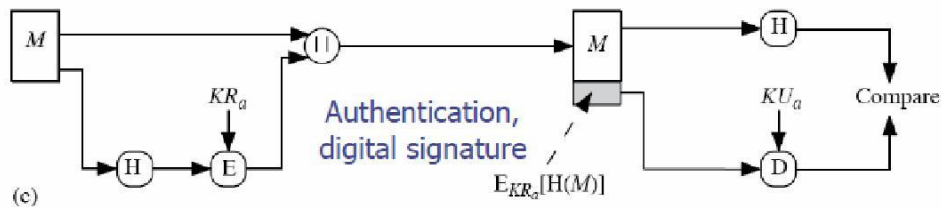
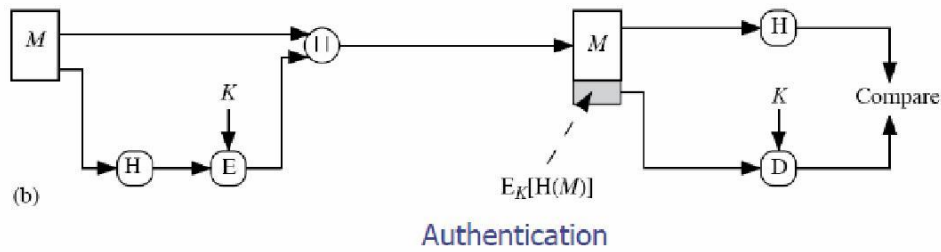
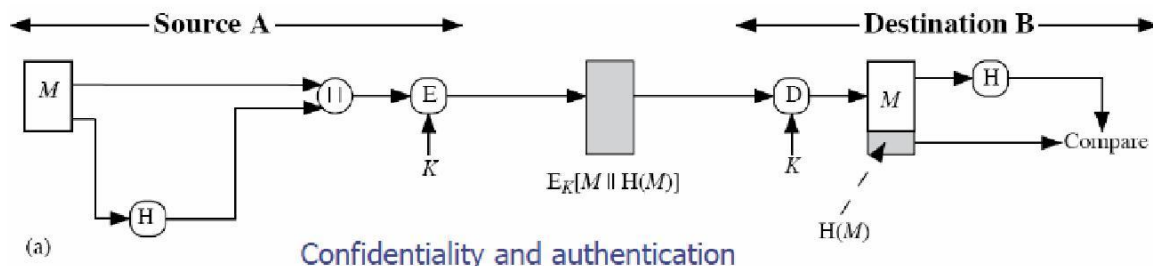
$A \rightarrow B: M \parallel C_K(M)$ <ul style="list-style-type: none"> <li>•Provides authentication — Only A and B share <math>K</math></li> </ul> <p>(a) Message authentication</p>
$A \rightarrow B: E_{K_2}[M \parallel C_{K_1}(M)]$ <ul style="list-style-type: none"> <li>•Provides authentication — Only A and B share <math>K_1</math></li> <li>•Provides confidentiality — Only A and B share <math>K_2</math></li> </ul> <p>(b) Message authentication and confidentiality: authentication tied to plaintext</p>
$A \rightarrow B: E_{K_2}[M] \parallel C_{K_1}(E_{K_2}[M])$ <ul style="list-style-type: none"> <li>•Provides authentication — Using <math>K_1</math></li> <li>•Provides confidentiality — Using <math>K_2</math></li> </ul> <p>(c) Message authentication and confidentiality: authentication tied to ciphertext</p>

## Hash Function

A variation on the message authentication code is the one-way hash function. As with the message authentication code, the hash function accepts a variable-size message  $M$  as input and produces a fixed-size hash code  $H(M)$ , sometimes called a message digest, as output. The hash code is a function of all bits of the message and provides an error-detection capability: A change to any bit or bits in the message results in a change to the hash code. A variety of ways in which a hash code can be used to provide message authentication is shown below and explained stepwise in the table.

$A \rightarrow B: E_K[M \parallel H(M)]$ <ul style="list-style-type: none"> <li>•Provides confidentiality — Only A and B share <math>K</math></li> <li>•Provides authentication — <math>H(M)</math> is cryptographically protected</li> </ul> <p>(a) Encrypt message plus hash code</p>	$A \rightarrow B: E_K[M \parallel E_{K_R}[H(M)]]$ <ul style="list-style-type: none"> <li>•Provides authentication and digital signature</li> <li>•Provides confidentiality — Only A and B share <math>K</math></li> </ul> <p>(d) Encrypt result of (c) - shared secret key</p>
$A \rightarrow B: M \parallel E_K[H(M)]$ <ul style="list-style-type: none"> <li>•Provides authentication — <math>H(M)</math> is cryptographically protected</li> </ul> <p>(b) Encrypt hash code - shared secret key</p>	$A \rightarrow B: M \parallel H(M \parallel S)$ <ul style="list-style-type: none"> <li>•Provides authentication — Only A and B share <math>S</math></li> </ul> <p>(e) Compute hash code of message plus secret value</p>
$A \rightarrow B: M \parallel E_{K_R}[H(M)]$ <ul style="list-style-type: none"> <li>•Provides authentication and digital signature — <math>H(M)</math> is cryptographically protected — Only A could create <math>E_{K_R}[H(M)]</math></li> </ul> <p>(c) Encrypt hash code - sender's private key</p>	$A \rightarrow B: E_K[M \parallel H(M) \parallel S]$ <ul style="list-style-type: none"> <li>•Provides authentication — Only A and B share <math>S</math></li> <li>•Provides confidentiality — Only A and B share <math>K</math></li> </ul> <p>(f) Encrypt result of (e)</p>







In cases where confidentiality is not required, methods b and c have an advantage over those that encrypt the entire message in that less computation is required. Growing interest for techniques that avoid encryption is due to reasons like, Encryption software is quite slow and may be covered by patents. Also encryption hardware costs are not negligible and the algorithms are subject to U.S export control.

A fixed-length hash value  $h$  is generated by a function  $H$  that takes as input a message of arbitrary length:  **$h=H(M)$** .

- ⌚ A sends  $M$  and  $H(M)$
- ⌚ B authenticates the message by computing  $H(M)$  and checking the match

Requirements for a hash function: The purpose of a hash function is to produce a “fingerprint” of a file, message, or other block of data. To be used for message authentication, the hash function  $H$  must have the following properties

- ⌚ H can be applied to a message of any size
- ⌚ H produces fixed-length output
- ⌚ Computationally easy to compute  $H(M)$  for any given  $M$
- ⌚ Computationally infeasible to find  $M$  such that  $H(M)=h$ , for a given  $h$ , referred to as the *one-way property*
- ⌚ Computationally infeasible to find  $M'$  such that  $H(M')=H(M)$ , for a given  $M$ , referred to as *weak collision resistance*.
- ⌚ Computationally infeasible to find  $M, M'$  with  $H(M)=H(M')$  (to resist to birthday attacks), referred to as *strong collision resistance*.

Examples of simple hash functions are:

- Bit-by-bit XOR of plaintext blocks:  $h = D1 \oplus D2 \oplus \dots \oplus Dn$
- rotated XOR –before each addition the hash value is rotated to the left with 1 bit
- Cipher block chaining technique without a secret key.

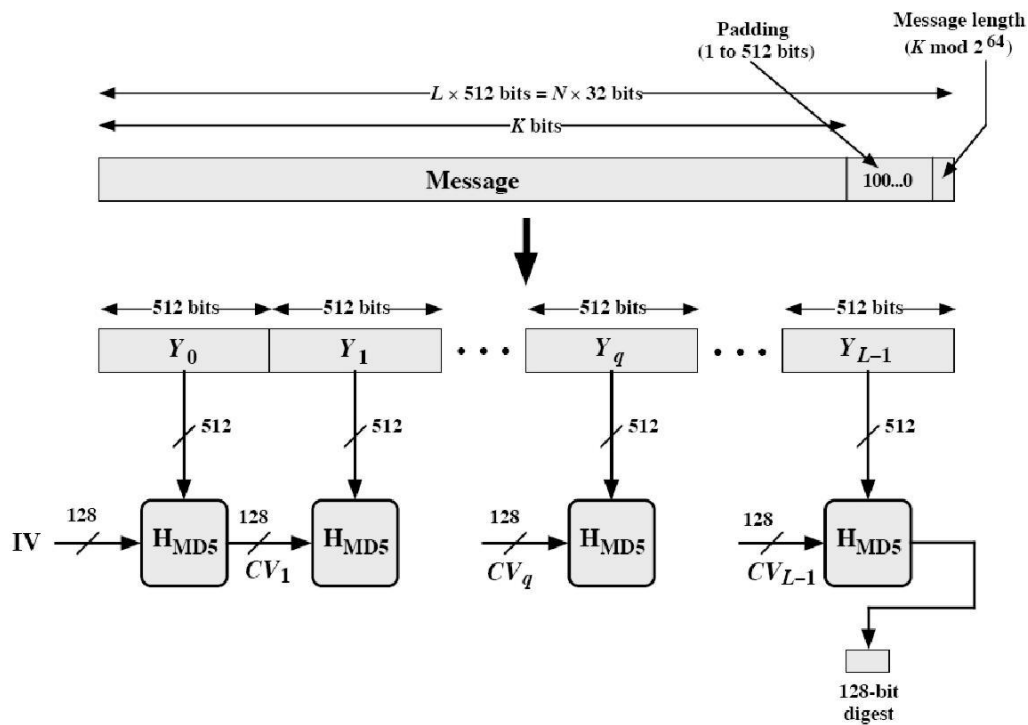
## MD5 Message Digest Algorithm

The MD5 message-digest algorithm was developed by Ron Rivest at MIT and it remained as the most popular hash algorithm until recently. The algorithm takes as input, a message of arbitrary length and produces as output, a 128-bit message digest. The input is processed in 512-bit blocks. The processing consists of the following steps:

- 1.) Append Padding bits: The message is padded so that its length in bits is congruent to 448 modulo 512 i.e. the length of the padded message is 64 bits less than an integer multiple of 512 bits.

Padding is always added, even if the message is already of the desired length. Padding consists of a single 1-bit followed by the necessary number of 0-bits.

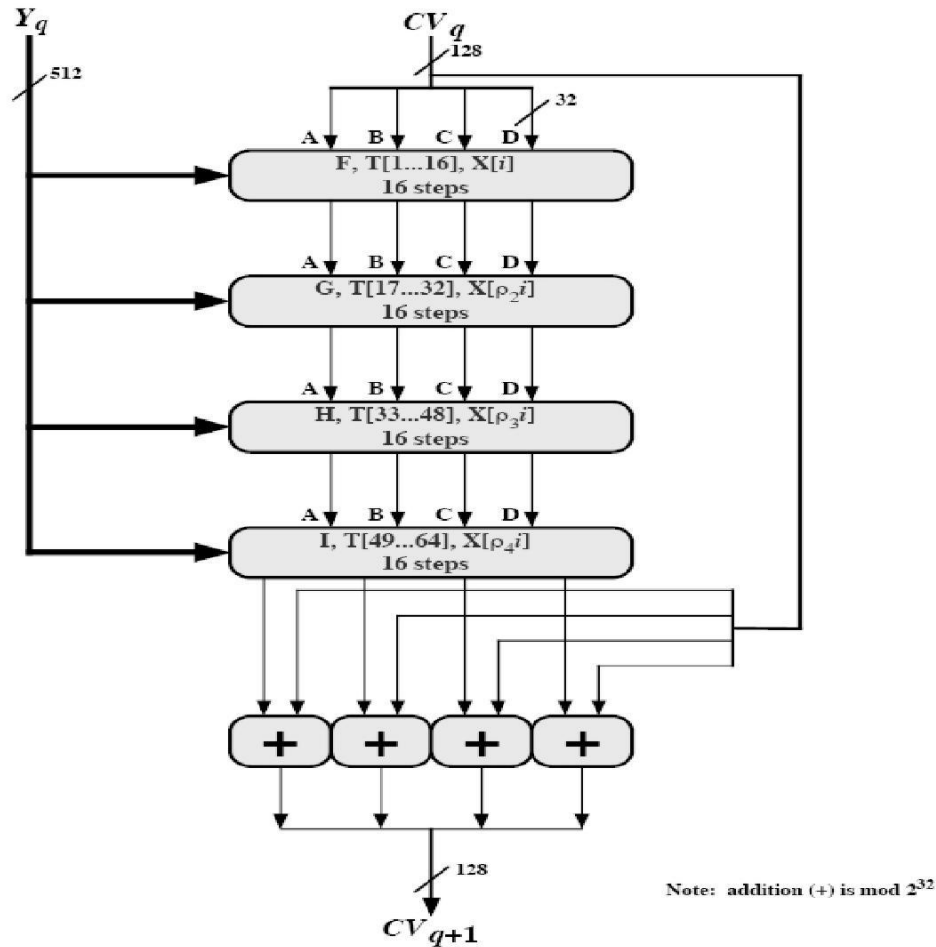
- 2.) **Append length:** A 64-bit representation of the length in bits of the original message (before the padding) is appended to the result of step-1. If the length is larger than 264, the 64 least representative bits are taken.
- 3.) **Initialize MD buffer:** A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (A, B, C, D) and are initialized with  $A=0x01234567$ ,  $B=0x89ABCDEF$ ,  $C=0xFEDCBA98$ ,  $D=0x76543210$  i.e. 32-bit integers (hexadecimal values).



### Message Digest Generation Using MD5

- 4.) **Process Message in 512-bit (16-word) blocks:** The heart of algorithm is the compression function that consists of four rounds of processing and this module is labeled HMD5 in the above figure and logic is illustrated in the following figure. The four rounds have a similar structure, but each uses a different primitive logical function, referred to as F, G, H and I in the specification. Each block takes as input the current 512-bit block being processed  $Y_q$  and the 128-bit buffer value ABCD and updates the contents of the buffer. Each round also makes use of one-fourth of a 64-element table  $T^*1....64+$ , constructed from the sine function. The  $i$ th element of  $T$ , denoted  $T[i]$ , has the value equal to the integer part of  $2^{32} * \text{abs}(\sin(i))$ , where  $i$  is in radians. As the value of  $\text{abs}(\sin(i))$  is a value between 0 and 1, each element of  $T$  is an integer that can be represented in

32-bits and would eliminate any regularities in the input data. The output of fourth round is added to the input to the first round ( $CV_q$ ) to produce  $CV_{q+1}$ . The addition is done independently for each of the four words in the buffer with each of the corresponding words in  $CV_q$ , using addition modulo  $2^{32}$ . This operation is shown in the figure below:



5.) **Output:** After all  $L$  512-bit blocks have been processed, the output from the  $L$ th stage is the 128-bit message digest. MD5 can be summarized as follows:

$$CV_0 = IV$$

$$CV_{q+1} = \text{SUM}_{32}(CV_q, RF_I Y_q, RF_H[Y_q, RF_G[Y_q, RF_F[Y_q, CV_q]]])$$

$$MD = CV_L$$

Where,

$IV$  = initial value of ABCD buffer, defined in step 3.

$Y_q$  = the  $q^{\text{th}}$  512-bit block of the message

$L$  = the number of blocks in the message

$CV_q$  = chaining variable processed with the  $q^{\text{th}}$  block of the message.

RF<sub>x</sub> = round function using primitive logical function x.

MD = final message digest value

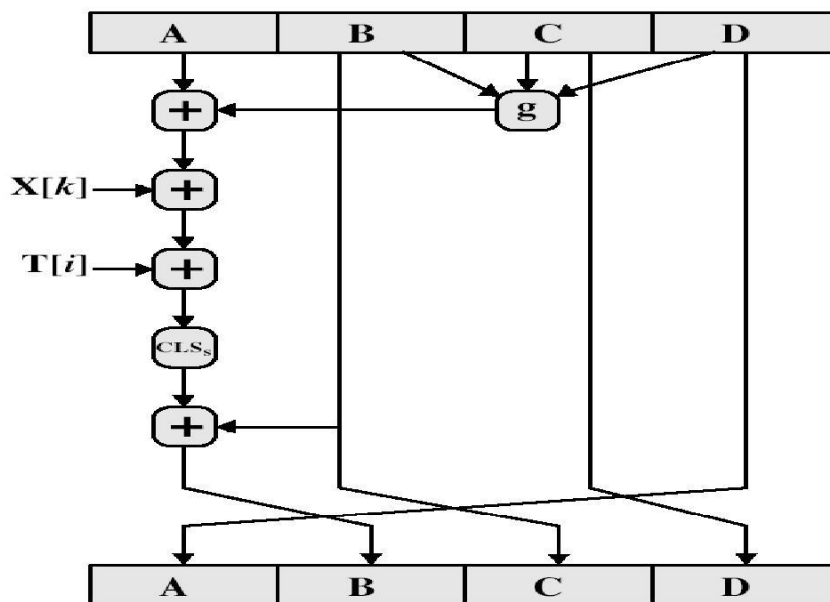
SUM<sub>32</sub> = Addition modulo  $2^{32}$  performed separately.

### MD5 Compression Function:

Each round consists of a sequence of 16 steps operating on the buffer ABCD. Each step is of the form,

$$a = b + ((a + g(b, c, d) + X[k] + T[i]) \lll s)$$

where a, b, c, d refer to the four words of the buffer but used in varying permutations. After 16 steps, each word is updated 4 times. g(b, c, d) is a different nonlinear function in each round (F, G, H, I). Elementary MD5 operation of a single step is shown below.



The primitive function g of the F, G, H, I is given as:

Round	Primitive function g	$g(b, c, d)$
1	F(b, c, d)	$(b \wedge c) \vee (b' \wedge d)$
2	G(b, c, d)	$(b \wedge d) \vee (c \wedge d')$
3	H(b, c, d)	$b \oplus c \oplus d$
4	I(b, c, d)	$c \oplus (b \vee d')$

Truth table

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

Where the logical operators (AND, OR, NOT, XOR) are represented by the symbols ( $\wedge$ ,  $\vee$ ,  $\sim$ ,  $\oplus$ ).

Each round mixes the buffer input with the next "word" of the message in a complex, non-linear manner. A different non-linear function is used in each of the 4 rounds (but the same function for all 16 steps in a round). The 4 buffer words (a,b,c,d) are rotated from step to step so all are used and updated. g is one of the primitive functions F,G,H,I for the 4 rounds respectively.  $X[k]$  is the kth 32-bit word in the current message block.  $T[i]$  is the ith entry in the matrix of constants T. The addition of varying constants T and the use of different shifts helps ensure it is extremely difficult to compute collisions.

The array of 32-bit words  $X[0..15]$  holds the value of current 512-bit input block being processed. Within a round, each of the 16 words of  $X[i]$  is used exactly once, during one step. The order in which these words is used varies from round to round. In the first round, the words are used in their original order. For rounds 2 through 4, the following permutations are used

$$\begin{aligned} \rho_2(i) &= (1 + 5i) \bmod 16 \\ \rho_3(i) &= (5 + 3i) \bmod 16 \\ \rho_4(i) &= 7i \bmod 16 \end{aligned}$$

## MD4



Precursor to MD5



Design goals of MD4 (which are carried over to MD5)

- Security
- Speed
- Simplicity and compactness
- Favor little-endian architecture



Main differences between MD5 and MD4

- A fourth round has been added.
- Each step now has a unique additive constant.

- The function g in round 2 was changed from  $(bc \vee bd \vee cd)$  to  $(bd \vee cd')$  to make g less symmetric.

- Each step now adds in the result of the previous step. This promotes a faster "avalanche effect".

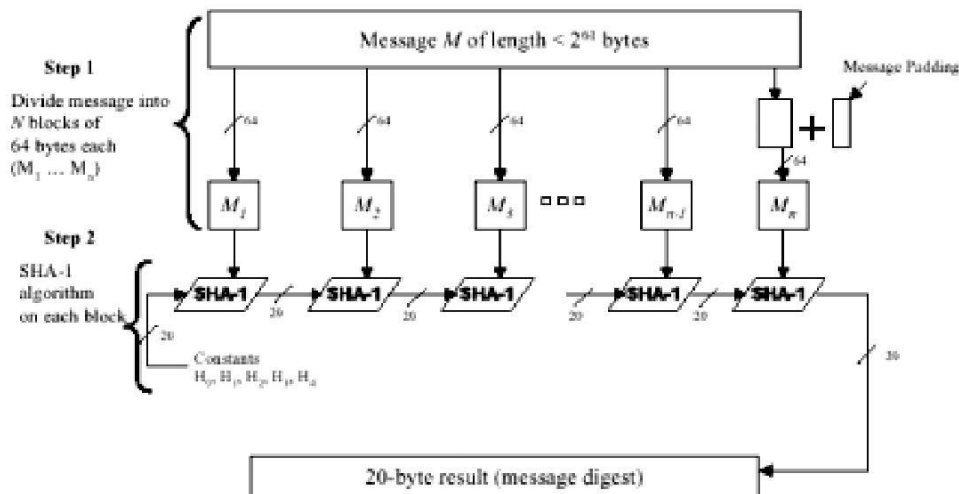
- The order in which input words are accessed in rounds 2 and 3 is changed, to make these patterns less like each other.

- The shift amounts in each round have been approximately optimized, to yield a faster "avalanche effect." The shifts in different rounds are distinct.

## Secure Hash Algorithm:

The secure hash algorithm (SHA) was developed by the National Institute of Standards and Technology (NIST). SHA-1 is the best established of the existing SHA hash functions, and is

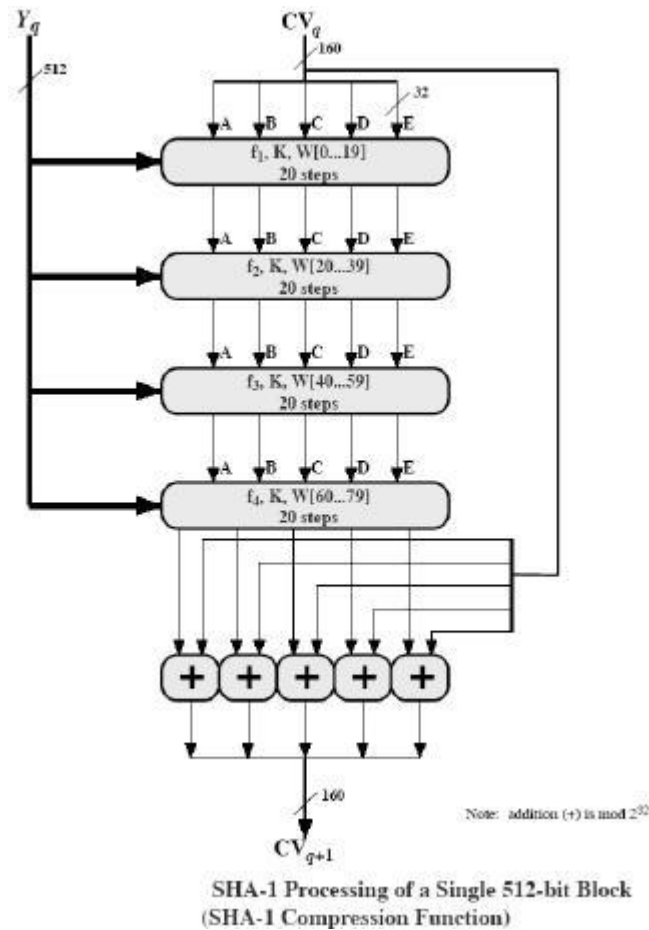
employed in several widely used security applications and protocols. The algorithm takes as input a message with a maximum length of less than  $2^{64}$  bits and produces as output a 160-bit message digest.



The input is processed in 512-bit blocks. The overall processing of a message follows the structure of MD5 with block length of 512 bits and a hash length and chaining variable length of 160 bits. The processing consists of following steps:

- 1.) Append Padding Bits: The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.
- 2.) Append Length: a block of 64 bits containing the length of the original message is added.
- 3.) Initialize MD buffer: A 160-bit buffer is used to hold intermediate and final results on the hash function. This is formed by 32-bit registers A,B,C,D,E. Initial values: A=0x67452301, B=0xEFCDAB89, C=0x98BADCFE, D=0x10325476, E=C3D2E1F0. Stores in big-endian format i.e. the most significant bit in low address.
- 4.) Process message in blocks 512-bit (16-word) blocks: The processing of a single 512-bit block is shown above. It consists of four rounds of processing of 20 steps each. These four rounds have similar structure, but uses a different primitive logical function, which we refer to as f1, f2, f3 and f4. Each round takes as input the current 512-bit block being processed and the 160-bit buffer value ABCDE and updates the contents of the buffer. Each round also makes use of four distinct additive constants  $K_t$ . The output of the fourth round i.e. eightieth step is added to the input to the first round to produce  $CV_{q+1}$ .

5.) Output: After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.



The behavior of SHA-1 is as follows:

$$CV_0 = IV$$

$$CV_{q+1} = SUM_{32}(CV_q, ABCDE_q)$$

$$MD = CV_L$$

Where, IV = initial value of ABCDE buffer

$ABCDE_q$  = output of last round of processing of qth message block

L = number of blocks in the message

$SUM_{32}$  = Addition modulo  $2^{32}$

MD = final message digest value.

### SHA-1 Compression Function:

Each round has 20 steps which replaces the 5 buffer words. The logic present in each one of the 80 present is given as

$$(A, B, C, D, E) \leftarrow (E + f_t(B, C, D) + S^5(A) + W_t + K_t, A, S^{30}(B), C, D)$$

Where, A, B, C, D, E = the five words of the buffer

t = step number;  $0 < t < 79$

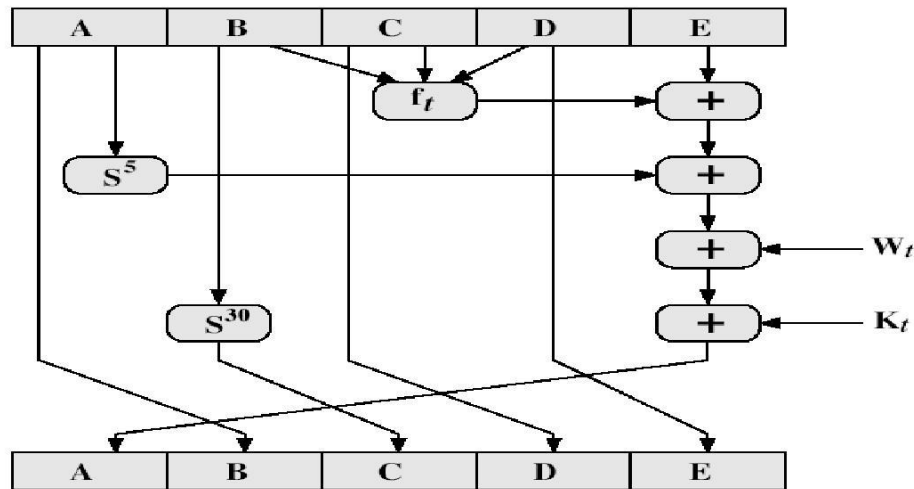
$f_t(B, C, D)$  = primitive logical function for step t

$S^k$  = circular left shift of the 32-bit argument by k bits

$W_t$  = a 32-bit word derived from current 512-bit input block.

$K_t$  = an additive constant; four distinct values are used

+ = modulo addition



Elementary SHA operation (single step)

SHA shares much in common with MD4/5, but with 20 instead of 16 steps in each of the 4 rounds. Note the 4 constants are based on  $\sqrt{2, 3, 5, 10}$ . Note also that instead of just splitting the input block into 32-bit words and using them directly, SHA-1 shuffles and mixes them using rotates & XOR's to form a more complex input, and greatly increases the difficulty of finding collisions. A sequence of logical functions  $f_0, f_1, \dots, f_{79}$  is used in the SHA-1.

Each  $f_t$ ,  $0 \leq t \leq 79$ , operates on three 32-bit words B, C, D and produces a 32-bit word as output.  $f_t(B, C, D)$  is defined as follows: for words B, C, D,

$$f_t(B, C, D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 \leq t \leq 19)$$

$$f_t(B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (20 \leq t \leq 39)$$

$$f_t(B, C, D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \leq t \leq 59)$$

$$f_t(B, C, D) = B \text{ XOR } C \text{ XOR } D \quad (60 \leq t \leq 79).$$



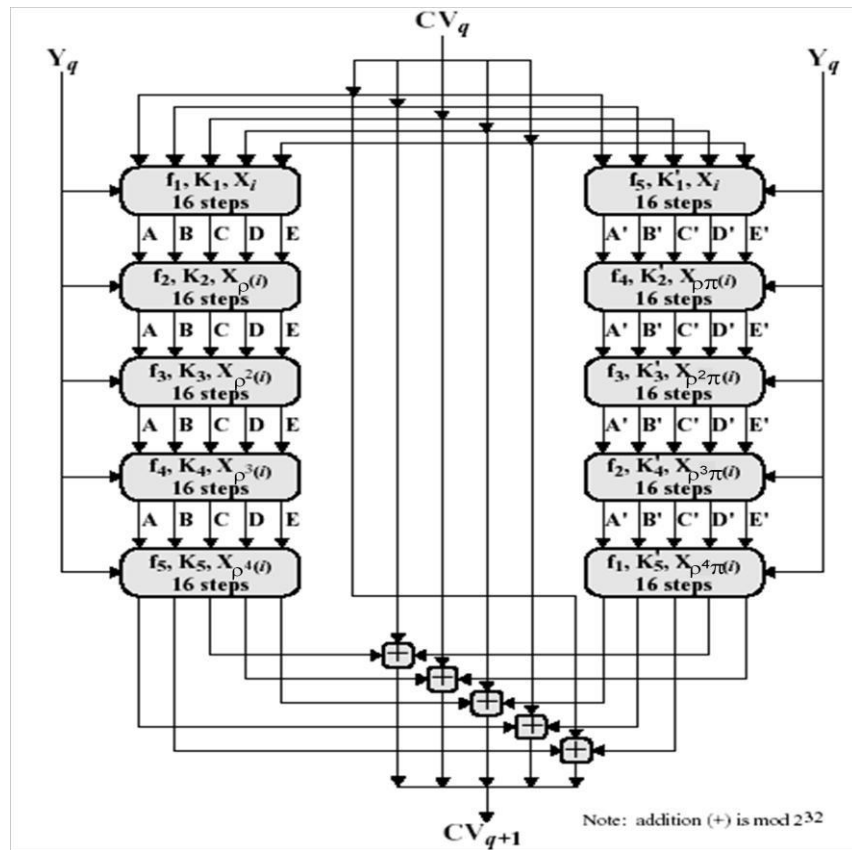
### *Comparison of SHA-1 with MD5*

12. brute force attack is harder (160 vs 128 bits for MD5)
13. not vulnerable to any known attacks (compared to MD4/5)
14. a little slower than MD5 (80 vs 64 steps)
15. both designed as simple and compact
16. optimised for big endian CPU's (vs MD5 which is optimised for little endian CPU's)

## **RIPEMD-160**

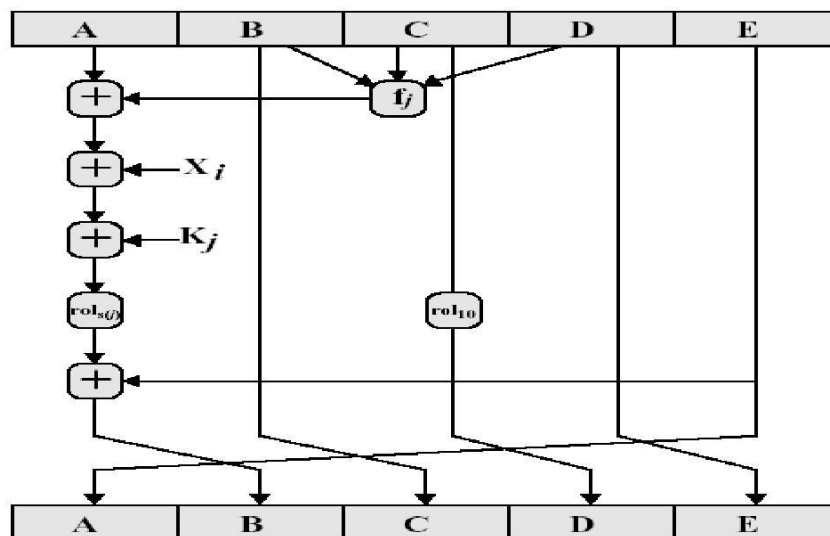
RIPEMD-160 was developed in Europe as part of RIPE project in 96 by researchers involved in attacks on MD4/5. It is somewhat similar to MD5/SHA and uses 2 parallel lines of 5 rounds of 16 steps. Creates a 160-bit hash value. It is slower, but probably more secure, than SHA. The processing consists of the following steps:

- 1.) Append Padding Bits: The message is padded so that length is congruent to 448 modulo 512; padding always added –one bit 1 followed by the necessary number of 0 bits.
- 2.) Append Length: a block of 64 bits containing the length of the original message is added.
- 3.) Initialize MD buffer: A 160-bit buffer is used to hold intermediate and final results on the hash function. This is formed by 32-bit registers A,B,C,D,E. Initial values: A=0x67452301, B=0xEFCDAB89, C=0x98BADCFE, D=0x10325476, E=C3D2E1F0. Unlike SHA, like MD5, RIPEMD-160 uses a little-endian convention.
- 4.) Process message in blocks 512-bit (16-word) blocks: The algorithm consists of 10 rounds of processing of 16 steps each. The 10 rounds are arranged as two parallel lines of five rounds. The processing is depicted below:  
  
The 10 rounds have a similar structure, but uses a different primitive logical function, referred to as f1, f2, f3, f4 and f5. The same functions are used in the reverse order in the right line. Each round also makes use of an additive constant and in total nine distinct constants are used, one of them being zero. The output of the fifth round is added to the chaining variable input to the first round  $CV_q$  to produce  $CV_{q+1}$  and this addition is done independently for each of the five words in buffer of each line with each of the words of  $CV_q$ .
- 5.) Output: After all L 512-bit blocks have been processed, the output from the Lth stage is the 160-bit message digest.



### RIPEMD-160 Compression Function

The compression function is rather more complex than SHA. Operation of single step of RIPEMD-160 is shown below:



One of the 5 primitive logical functions is used in each round; (functions used in reverse order on the right line). Each primitive function takes three 32-bit words as input and produces a 32-bit

word output. It performs a set of bitwise logical operations and the functions are summarized below and the truth table for logical functions is also given below;

Step	Function Name	Value
$0 \leq j \leq 15$	$f_1 = f(j, B, C, D)$	$B \oplus C \oplus D$
$16 \leq j \leq 31$	$f_2 = f(j, B, C, D)$	$(B \wedge C) \vee (B' \wedge D)$
$32 \leq j \leq 47$	$f_3 = f(j, B, C, D)$	$(B \vee C') \oplus D$
$48 \leq j \leq 63$	$f_4 = f(j, B, C, D)$	$(B \wedge D) \vee (C \wedge D')$
$64 \leq j \leq 79$	$f_5 = f(j, B, C, D)$	$B \oplus (C \vee D')$

**Truth Table**

B	C	D	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$
0	0	0	0	0	1	0	1
0	0	1	1	1	0	0	0
0	1	0	1	0	0	1	1
0	1	1	0	1	1	0	1
1	0	0	1	0	1	0	0
1	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0
1	1	1	1	1	0	1	0

The pseudo code given below defines the processing algorithm for one round

```

A := CVq(0); B := CVq(1); C := CVq(2); D := CVq(3); E := CVq(4);
A' := CVq(0); B' := CVq(1); C' := CVq(2); D' := CVq(3); E' := CVq(4);
for j:=0 to 79 do
    T := rols(j)(A + f(j, B, C, D) + Xr(j) + K(j)) + E;
    A := E; E := D; D := rol10(C); C := B; B := T;
    T' := rols'(j)(A' + f(79-j, B', C', D') + Xr'(j) + K'(j)) + E';
    A' := E'; E' := D'; D' := rol10(C'); C' := B'; B' := T';
enddo
CVq+1(0) := CVq(1) + C + D'; CVq+1(1) := CVq(2) + D + E'; CVq+1(2) := CVq(3) + E + A';
CVq+1(3) := CVq(4) + A + B'; CVq+1(4) := CVq(0) + B + C';

```

where

ABCDE (A'B'C'D'E') = 5 words of the buffer for the left (right) line

j = step number; 0..79

f(j, B, C, D) = primitive logical functions used in step j

rol<sub>s(j)</sub> = circular left shift (rotation); s(j) is a function that determines the amount of rotation for a particular step

X<sub>r(j)</sub> = a 32-bit word from the current 512-bit input block; r(j) is a permutation function that selects a particular word

K(j) = additive constants used in step j

RIPEMD-160 is probably the most secure of the hash algorithms. The following are the design criteria taken into consideration by the developers of RIPEMD-160 to get some level of detail that must be considered in designing a strong cryptographic hash function.

- ☐ Use 2 parallel lines of 5 rounds for increased complexity
- ☐ For simplicity the 2 lines are very similar i.e. same logic. But the notable differences are additive constants, order of primitive logical functions and processing of 32-bit words.
- ☐ Step operation very close to MD5
- ☐ Permutation varies parts of message used
- ☐ Circular shifts designed for best results

Comparison of above stated three algorithms is given below in a tabular form:

	MD5	SHA-1	RIPEMD-160
Digest length	128 bits	160 bits	160 bits
Basic unit of processing	512 bits	512 bits	512 bits
Number of steps	64 (4 rounds of 16)	80 (4 rounds of 20)	160 (5 paired rounds of 16)
Maximum message size	$\infty$	$2^{64} - 1$ bits	$\infty$
Primitive logical functions	4	4	5
Additive constants used	64	4	9
Endianness	Little-endian	Big-endian	Little-endian

# HMAC

Interest in developing a MAC, derived from a cryptographic hash code has been increasing mainly because hash functions are generally faster and are also not limited by export restrictions unlike block ciphers. Additional reason also would be that the library code for cryptographic hash functions is widely available. The original proposal is for incorporation of a secret key into an existing hash algorithm and the approach that received most support is HMAC. HMAC is specified as Internet standard RFC2104. It makes use of the hash function on the given message. Any of MD5, SHA-1, RIPEMD-160 can be used.

## *HMAC Design Objectives*

- To use, without modifications, available hash functions
- To allow for easy replaceability of the embedded hash function
- To preserve the original performance of the hash function
- To use and handle keys in a simple way
- To have a well understood cryptographic analysis of the strength of the MAC based on reasonable assumptions on the embedded hash function

The first two objectives are very important for the acceptability of HMAC. HMAC treats the hash function as a “black box”, which has two benefits. First is that an existing implementation of the hash function can be used for implementing HMAC making the bulk of HMAC code readily available without modification. Second is that if ever an existing hash function is to be replaced, the existing hash function module is removed and new module is dropped in. The last design objective provides the main advantage of HMAC over other proposed hash-based schemes. HMAC can be proven secure provided that the embedded hash function has some reasonable cryptographic strengths.

## Steps involved in HMAC algorithm:

- ❑ Append zeroes to the left end of K to create a b-bit string  $K^+$  (ex: If K is of length 160-bits and b = 512, then K will be appended with 44 zero bytes).
- ❑ XOR(bitwise exclusive-OR)  $K^+$  with ipad to produce the b-bit block  $S_i$ .
- ❑ Append M to  $S_i$ .
- ❑ Now apply H to the stream generated in step-3
- ❑ XOR  $K^+$  with opad to produce the b-bit block  $S_0$ .
- ❑ Append the hash result from step-4 to  $S_0$ .
- ❑ Apply H to the stream generated in step-6 and output the result.

### HMAC Algorithm

- Define the following terms

$H$  = embedded hash function

$M$  = message input to HMAC

$Y_i$  =  $i^{\text{th}}$  block of  $M$ ,  $0 \leq i \leq L - 1$

$L$  = number of blocks in  $M$

$b$  = number of bits in a block

$n$  = length of hash code produced by embedded hash function

$K$  = secret key; if key length is greater than  $b$ , the key is input to the hash function to produce an  $n$ -bit key; recommended length  $\geq n$

$K^+$  =  $K$  padded with 0's on the left so that the result is  $b$  bits in length

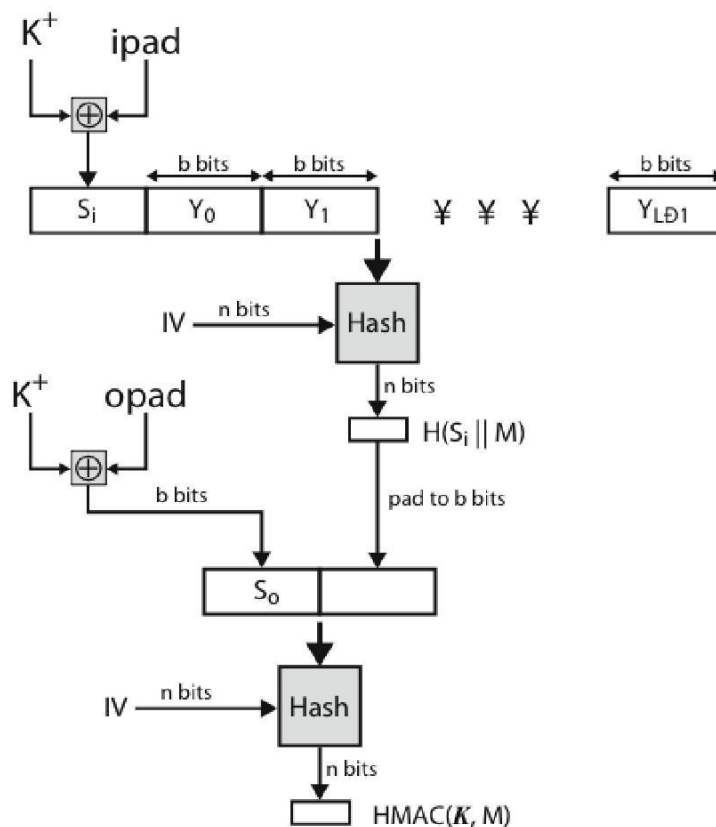
ipad = 00110110 repeated  $b/8$  times

opad = 01011100 repeated  $b/8$  times

- Then HMAC can be expressed as

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

### HMAC Structure:



The XOR with ipad results in flipping one-half of the bits of  $K$ . Similarly, XOR with opad results in flipping one-half of the bits of  $K$ , but different set of bits. By passing  $S_i$  and  $S_0$  through the compression function of the hash algorithm, we have pseudorandomly generated two keys from  $K$ .

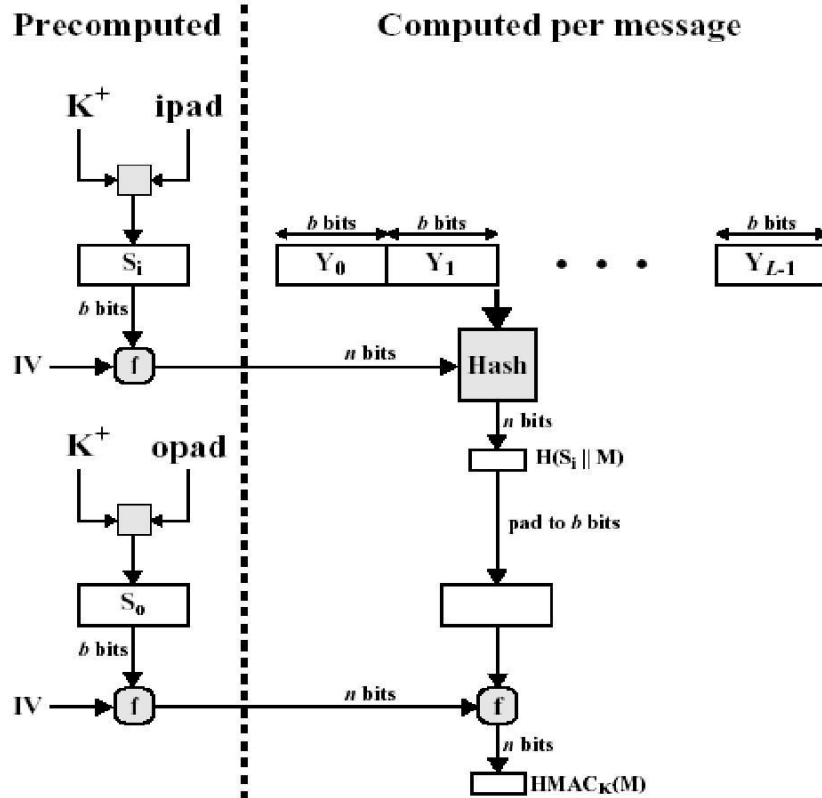
HMAC should execute in approximately the same time as the embedded hash function for long messages. HMAC adds three executions of the hash compression function (for  $S_0$ ,  $S_i$ , and the block produced from the inner hash)

A more efficient implementation is possible. Two quantities are precomputed.

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

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

where  $f$  is the compression function for the hash function which takes as arguments a chaining variable of  $n$  bits and a block of  $b$ -bits and produces a chaining variable of  $n$  bits.



As shown in the above figure, the values are needed to be computed initially and every time a key changes. The precomputed quantities substitute for the initial value (IV) in the hash function. With this implementation, only one additional instance of the compression function is added to the processing normally produced by the hash function. This implementation is worthwhile if most of the messages for which a MAC is computed are short.

Security of HMAC:

The appeal of HMAC is that its designers have been able to prove an exact relationship between the strength of the embedded hash function and the strength of HMAC. The security of a MAC function is generally expressed in terms of the probability of successful forgery with a given amount of time spent by the forger and a given number of message-MAC pairs created with the same key. Have two classes of attacks on the embedded hash function:

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

These attacks are likely to be caused by brute force attack on key used which has work of order  $2^n$ ; or a birthday attack which requires work of order  $2^{(n/2)}$  - but which requires the attacker to observe  $2^{n/2}$  blocks of messages using the same key - very unlikely. So even MD5 is still secure for use in HMAC given these constraints.