BLOCK CIPHERS & SYMMETRIC KEY CRYPTOGRAPHY

Syllabus: Traditional Block Cipher Structure, DES, Block Cipher Design Principles, AES-Structure, Transformation functions, Key Expansion, Blowfish, CAST-128, IDEA, Block Cipher Modes of Operations

Simplified DES: Simplified DES, developed by Professor Edward Schaefer of Santa Clara University, is an educational rather than a secure encryption algorithm. It has similar properties and structure to DES with much smaller parameters.

The S-DES encryption algorithm takes an 8-bit block of plain text and a 10-bit key as input and produces an 8-bit block of cipher text as output. The S-DES decryption algorithm takes an 8-bit block of cipher text and the same 10-bit key used to produce that cipher text as input and produces the original 8-bit block of plaintext as shown in the following figure. The encryption algorithm involves five functions:

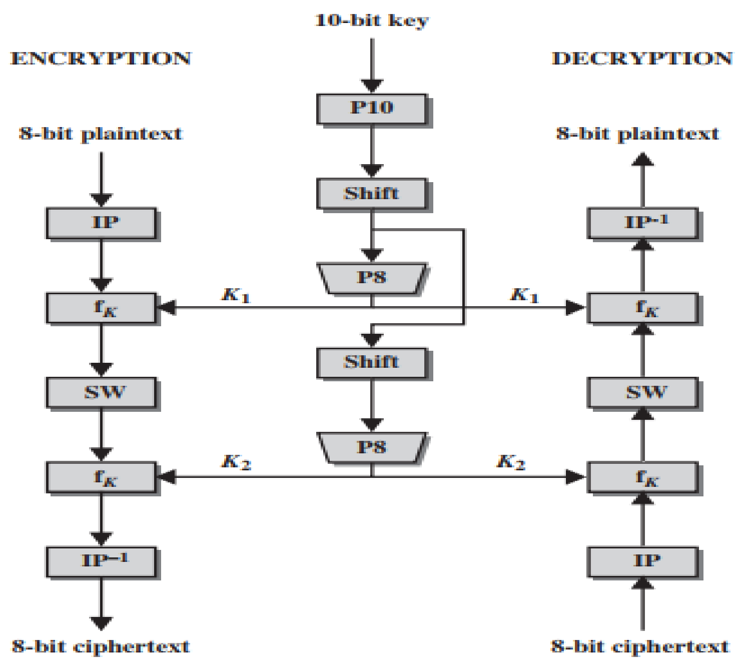
i) An initial permutation (IP);

ii) A complex function labelled fK, which involves both permutation and substitution operations and depends on a key input;

iii) A simple permutation function that switches (SW) the two halves of the data;

iv) The function fK again;

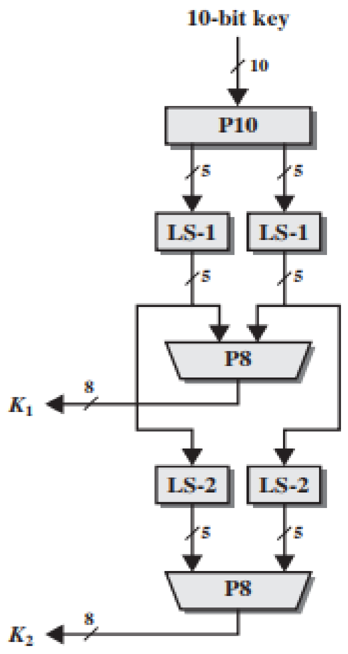
v) And finally a permutation function that is the inverse of the initial permutation (IP–1).



Simplified DES Scheme

The use of multiple stages of permutation and substitution results in a more complex algorithm, which increases the difficulty of cryptanalysis.

S-DES Key Generation: The complex function fK takes both 6 bit plain text and key as input. It uses a 10-bit key from which two 8-bit sub keys are generated as shown in the following figure.



Key Generation for Simplified DES

In this case, the key is first subjected to a permutation (P10). Then a circular left shift or left shift (LS) operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first sub key (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second sub key (K2).

We can concisely express the encryption algorithm as a composition of functions:

$$IP^{-1}(fK2(SW(fK1(IP(plain\ text)))))$$

which can also be written as: Cipher text = IP^{-1}(fK2(SW(fK1(IP(plaintext)))))

where K1 = P8(Shift(P10(key))) and

K2 = P8(Shift(Shift(P10(key))))

Decryption is also shown in the above figure and is essentially the reverse of encryption:

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit sub keys are produced for use in particular stages of the encryption and decryption algorithm. Figure G.2 depicts the stages followed to produce the sub keys. First, permute the key in the following fashion. Let the 10-bit key be designated as (k1, k2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as: P10(k1, k2, k3, k4, k5, k6, k7, k8, k9, k10) = (k3, k5, k2, k7, k4, k10, k1, k9, k8, k6). P10 can be concisely defined by the display:

P10									
3	5	2	7	4	10	1	9	8	6

This table is read from left to right; each position in the table gives the identity of the input bit that produces the output bit in that position. So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on.

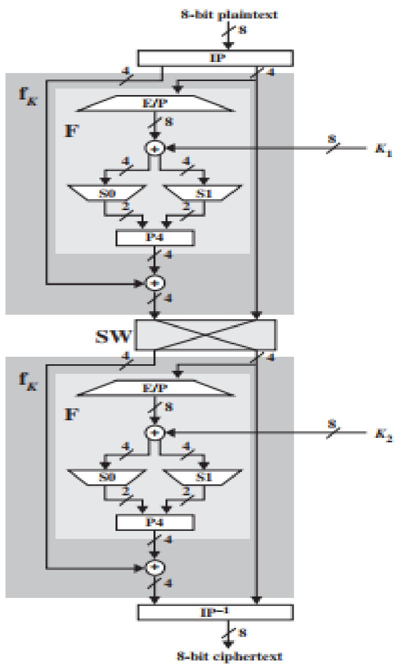
For example, the key (1010000010) is permuted to (1000001100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In the given example, the result is (00001 11000).

Next we apply P8, 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 sub key1 (K1). In our example, this yields (10100100) We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2. In our example, the result is (01000011).

S-DES Encryption: Encryption involves 5 different functions.



Simplified DES Encryption Detail

1. The input to the algorithm is the 8-bit plain text which is first permuted using the following IP function

IP							
2	6	3	1	4	8	5	7

2. The most complex function of s-des is F_k which is a combination of both substitution and permutation function and is summarized in the following equation. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to fK, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

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

where SK is a subkey and (+) is the bit-by-bit exclusive-OR function. For example, suppose the output of the IP stage is (10111101) and F(1101, SK) = (1110) for some key SK. Then fK(10111101) = (01011101) because (1011) ! (1110) = (0101). We now describe the mapping F. The input is a 4-bit number (n1,n2,n3,n4). The first operation is an expansion/permutation operation:

E/P							
4	1	2	3	2	3	4	1

Therefore it is clear that

$$\begin{array}{c|cc|c} n_4 & n_1 & n_2 & n_3 \\ n_2 & n_3 & n_4 & n_1 \end{array}$$

The 8-bit sub key k1=(k_{11},k_{12},k_{13},k_{14},k_{15},k_{16},k_{17},k_{18}) is added to this function, then

$$\begin{array}{c|cc|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}$$

Let us rename these 8 bits as

$$\begin{array}{c|cc|c} P_{0,0} & P_{0,1} & P_{0,2} & P_{0,3} \\ P_{1,0} & P_{1,1} & P_{1,2} & P_{1,3} \end{array}$$

The first 4 bits (first row) are compared into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are compared into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 3 & 2 \\ 1 & 3 & 2 & 1 & 0 \\ 2 & 0 & 2 & 1 & 3 \\ 3 & 3 & 1 & 3 & 2 \end{array} \qquad S1 = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 2 & 3 \\ 1 & 2 & 0 & 1 & 3 \\ 2 & 3 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 & 3 \end{array}$$

The first and fourth input bits (end bits) are treated as a 2-bit number that specify a row index of the S-box, and the second and third input bits (middle bits) specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output.

Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

3. Switch Function (swap): The function fK only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of fK operates on a different 4 bits.
4. Again the same complex function is repeated with the sub key K_2
5. Apply inverse Initial permuation(IP^{-1}) on the output of complex function fk2 which give an 8-bit cipher text.

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

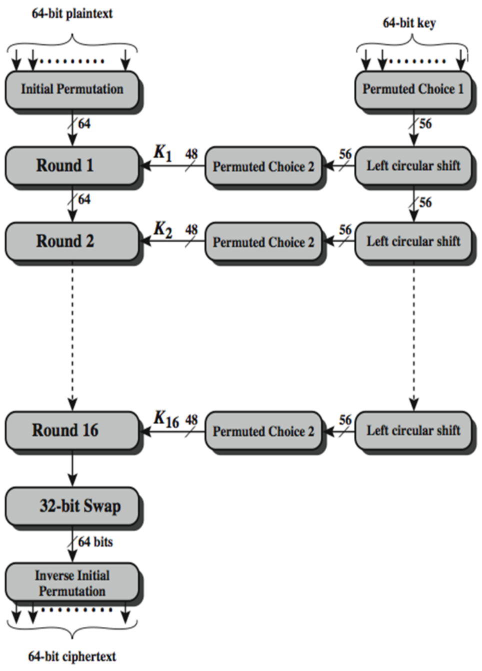
Data Encryption Standard:

The most widely used encryption scheme adopted in 1977 by National Bureau of Standards, now it is NIST as FIPS46(Federal Information Processing Standard)

The components of DES algorithm are:

Plain Text Block Size	:64 bits
Key size	:56 bits
No of Rounds	:16
No of Sub keys	:2, where each sub key is of length 48 bits

In DES, there are two inputs to an encryption function that is plain text and key. Inititailly the 64 bit plain text is taken an input to the Initial Permutation Function (IP) which rearranges the bits to produce 64 bit permuted sequence. This is followed by 16 rounds to produce cipher text.



These 16 rounds include both substitution and permutation functions. The output of the last round consists of 64 bits. The left and right halves of output are swapped to produce the output. Finally this is passed through inverse initial permutation (IP^{-1}) to produce cipher text. The right hand part of the above diagram shows the way in which the 56-bit key is used. Initially given key is passes through a permutation function (PC-I). Then for each of the 16 rounds, a sub key k_i is generated with a Left shift and permuted choice II (PC-II) function.

Details of single Round:

1. Arrange the given 64-bit plain text in the following format

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	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

2. 64-bit plain text will be taken as input to IP function which rearranges positions of bits and out puts again 64 bits.

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

3. The left and right halves of 64-bit value are treated as separate 32-bit quantities. The right hand side 32 bit will be taken as input to Expansion/Permutation function and it outputs a 48 bit data.

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

4. These 48 bits will be taken as an input to XOR function with the corresponding round sub key and outputs a 48 bit data.
5. Then these are compared in 8 s-boxes. Each s-box takes a 6-bit input and outputs a 4-bit. Each row of s-box contains all 16 possibilities i.e from 0 to 15. To represent these values we need a 4 bit information. Therefore 8 s-boxes output 32 bits of data.

S_1	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S_2	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S_3	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S_4	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_5	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S_6	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S_7	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S_8	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

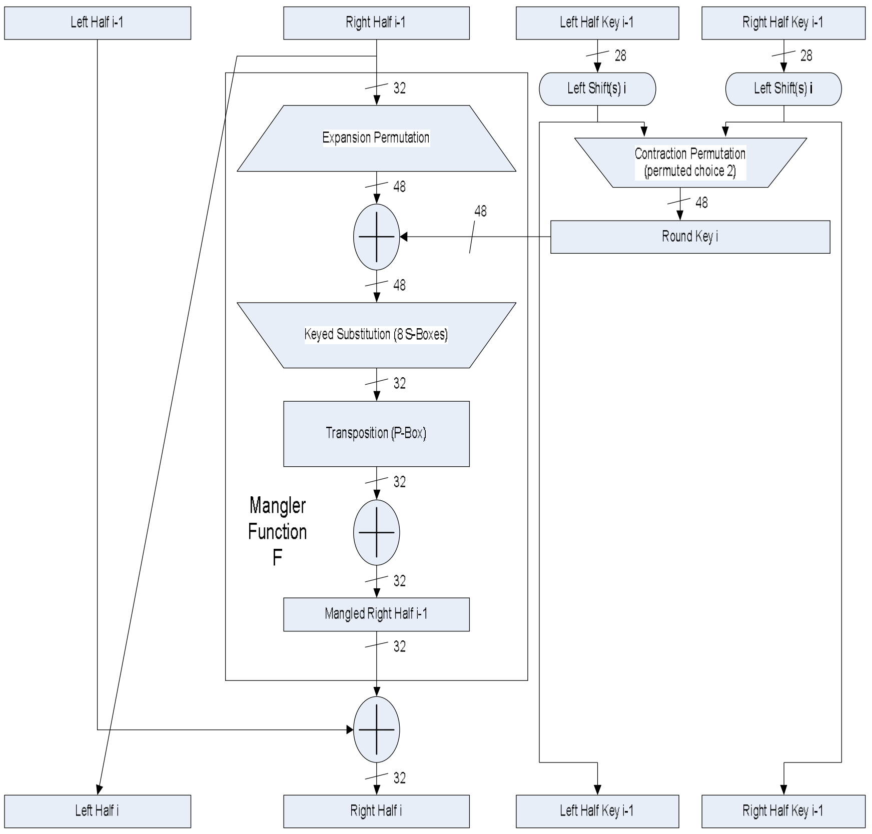
6. Now, apply P-32 function on the output of four s-boxes.

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

7. Apply an XOR operation with left hand side 32 bits and on the output of P-32

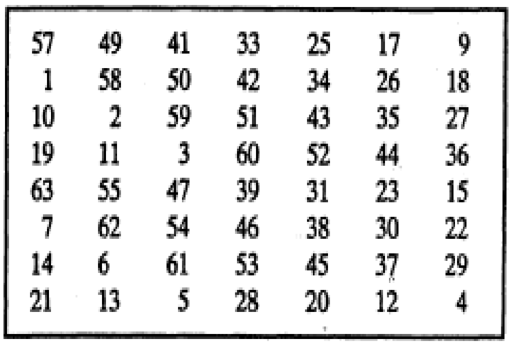
40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

8. Then apply 32 bit swap between left hand side data and right hand data.
9. Finally apply Inverse Initial Permutation on the output of 32-bit swap to get cipher text. The overall procedure can be summarized as:



Sub-key Generation: the given 56 bit key is extended to 64 bit by adding a parity bit for each 7-bits. Let us assume that these 64-bits are arranged in the following way

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	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64



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

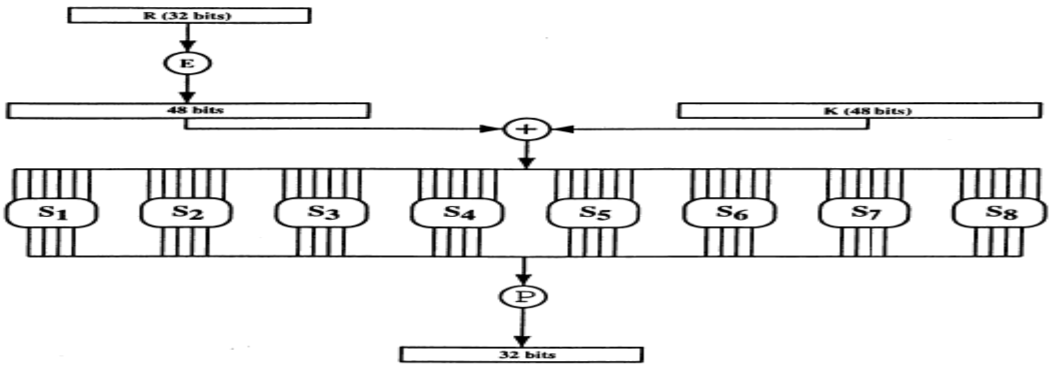
- Apply PC-I on the extended 64 bits
- On the output of PC-I divide into two equal halves C_{i-1} and D_{i-1}
- Apply LCS-1 or LCS-2 on these two halves based on the round count

Round number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits rotated	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

- Then apply PC-II operation to get a 48 bit sub key for the corresponding round

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

Role of S-Boxes: Each S-box accepts 6-bits as input and produces 4 bits as o/p. The first and last bits signify row index and middle 4 bits signify column index. Each row of s-box contains all the 16 possible combinations. The decimal value selected by the row and colum is converted to binary and is considered as the output of s-box. In this way 8 s-boxes produce 1 32 bit output.



Avalanche Effect: A small change in either the Plain text or key should produce a significant change in cipher text. That is a change in 1 bit of plain text or key should produce a change in many bits of the cipher text. The two plain texts that differ in 1-bit are used

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

10000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

With the following key

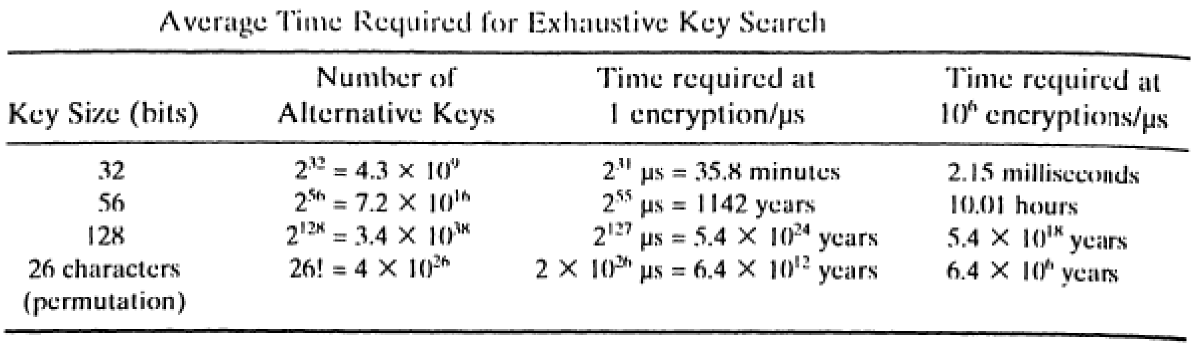
0000001 1001011 0100100 1100010 0011100 0011000 0011100 0110010

(a) Change in Plaintext		(b) Change in Key	
Round	Number of bits that differ	Round	Number of bits that differ
0	1	0	0
1	6	1	2
2	21	2	14
3	35	3	28
4	39	4	32
5	34	5	30
6	32	6	32
7	31	7	35
8	29	8	34
9	42	9	40
10	44	10	38
11	32	11	31
12	30	12	33
13	30	13	28
14	26	14	26
15	29	15	34
16	34	16	35

Strength of DES : The strength of DES depends on key size and the nature of algorithm.

1. **The Use of 56-Bit Keys**: With a key length of 56 bits, there are 2^{56} possible keys, which is approximately 7.2 * 1016 keys. Therefore brute-force attack appears impractical. Assuming that, on average, half the key space has to be searched, a single machine performing one DES encryption per microsecond would take more than a thousand years to break the cipher.

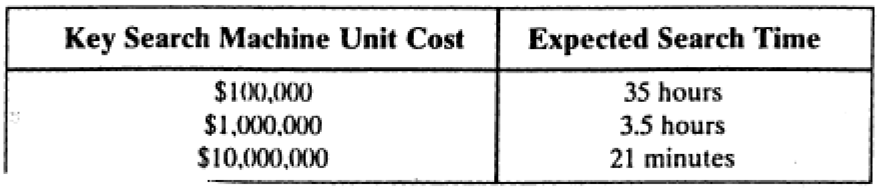
Average Time Required for Exhaustive Key Search



Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	2^{31} μs = 35.8 minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	2^{55} μs = 1142 years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	2^{127} μs = 5.4×10^{24} years	5.4×10^{18} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	2×10^{26} μs = 6.4×10^{12} years	6.4×10^6 years

However, the assumption of one encryption per microsecond is overly conservative. As far back as 1977, Diffie and Hellman postulated that the technology existed to build a parallel machine with 1 million encryption devices, each of which could perform one encryption per microsecond [DIFF77]. This would bring the average search time down to about 10 hours. The authors estimated that the cost would be about $20 million in 1977 dollars.

Weiner reports on the design of a chip that uses pipelined techniques to achieve a key search rate 50 million keys per second. Using 1993 costs, he designed a module that costa one lakh dollars and contains 5760 key search chips. With this design the following results are obtained.



Key Search Machine Unit Cost	Expected Search Time
$100,000	35 hours
$1,000,000	3.5 hours
$10,000,000	21 minutes

2. **The Nature of the DES Algorithm**

Another concern is the possibility that cryptanalysis is possible by exploiting the characteristics of the DES algorithm. The focus of concern has been on the eight substitution tables, or S-boxes, that are used in each iteration. Because the design criteria for these boxes, and indeed for the entire algorithm, were not made public, there is a suspicion that the boxes were constructed in such a way that cryptanalysis is possible for an opponent who knows the weaknesses in the S-boxes. This assertion is tantalizing, and over the years a number of regularities and unexpected behaviours of the S-boxes have been discovered. Despite this, no one has so far succeeded in discovering the supposed fatal weaknesses in the S-boxes.

3. **Timing Attacks**: We discuss timing attacks in more detail in Part Two, as they relate to public-key algorithms. However, the issue may also be relevant for symmetric ciphers. In essence, a timing attack is one in which information about the key or the plaintext is obtained by observing how long it takes a given implementation to perform decryptions on various cipher texts. A timing attack exploits the fact that an encryption or decryption algorithm often takes slightly different amounts of time on different inputs.

Block Cipher Design Principles: There are three critical aspects in designing block ciphers: number of rounds, Design of function F and key scheduling

DES design Criteria: S-box design and P function design criteria is as follows:

1. N output of any s-box should be too close to a linear function of the input bits.
2. Each row of an s-box should contain all 16 possibilities
3. If two inputs to an s-box differ in exactly one bit, the outputs differ in at least two bits
4. If two inputs to an s-box differ in two middle bits exactly then their output mst differ in at least two bits.
5. If two inputs to an s-box differ in their first two bits and are identical in their last two bits, the two outputs must not be the same.
6. For any non-zero 6-bit difference between inputs, no more than 8 of the 32 pairs of inputs exhibit the same difference in their outputs.

Criteria for function P:

1. The four output bits from each s-box at round I are distributed so that two of them affect middle bits of round (i+1) and the other two affect end bits.
2. The four output bits from each s-box affect six different s-boxes on the next round , and no two affect the same s-box

3. For two s-boxes j,k if an output bit from Sj affects a middle bit os Sk on the next round, then an out put bit fro, Sk cannot affect a moddle bit of Sj.

Number of Rounds: The greater the number of rounds , the more difficult it is to perform analysis, even for a week function F.

Design of Complex Function F: This function relies on the use of s-boxes. The function F provides elements of confusion. Therefore it is very difficult to unscramble the substitution performed by F. Several other criteria should be considered in designing F is:

1. Strict Avalanche Criteria: Any output j of an s-box should change with probability ½ when any single input bit 'I' is inverted for all I,j.
2. Bit Independence Criteria: The output bits j and k should change independently when considering design that do not include s-boxes.

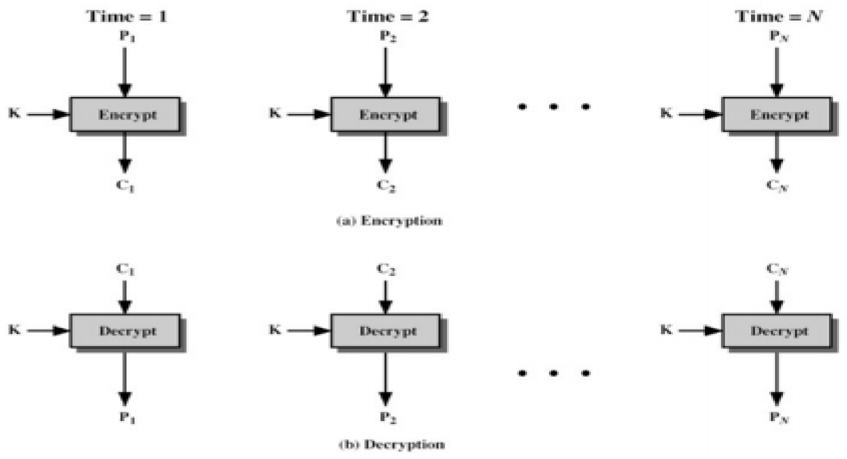
S-box Design: The guaranteed avalanche criterion for s-boxes is as follows: An s-box satisfies GA of order n if, for a 1-bit input change atleast n output bits change. S-boxes are generated in the following way

1. Random: use some pseudorandom number generation for generating each row of an s-box
2. Random with Testing: choose s-box entries randomly and then test the results against various criteria and throw away that do not pass
3. Man-made: using manual approach s-boxes elements are generated with simple mathematics
4. Math-made: generate s-boxes according to mathematical principles.

Key-Schedule Algorithm: This is the final area of block cipher design, and one that has received less attention than s-box design is the key generation. This is used to generate sub keys corresponding to each round.

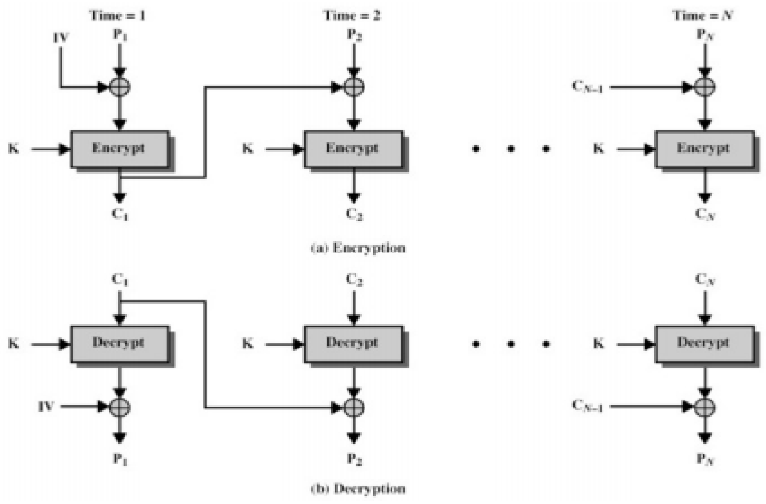
Modes of Block Ciphers: We have seen previously that five modes of operation are used when applying block ciphers in a variety of applications. This section will give a more detailed view of how these modes operate.

Electronic Codebook Mode (ECB) : This first mode is the simplest of all five modes. Figure 7.10 shows the scheme where it can be seen that a block of plaintext (which is the same size in each case) is encrypted with the same key K. The term codebook is used because, for a given key, there is a unique cipher text for every block of plaintext. Therefore we can imagine a gigantic codebook in which there is an entry for every possible plaintext pattern showing its corresponding cipher text. If the message is longer than the block length then the procedure is to break the message into blocks of the required length padding the last block if necessary.

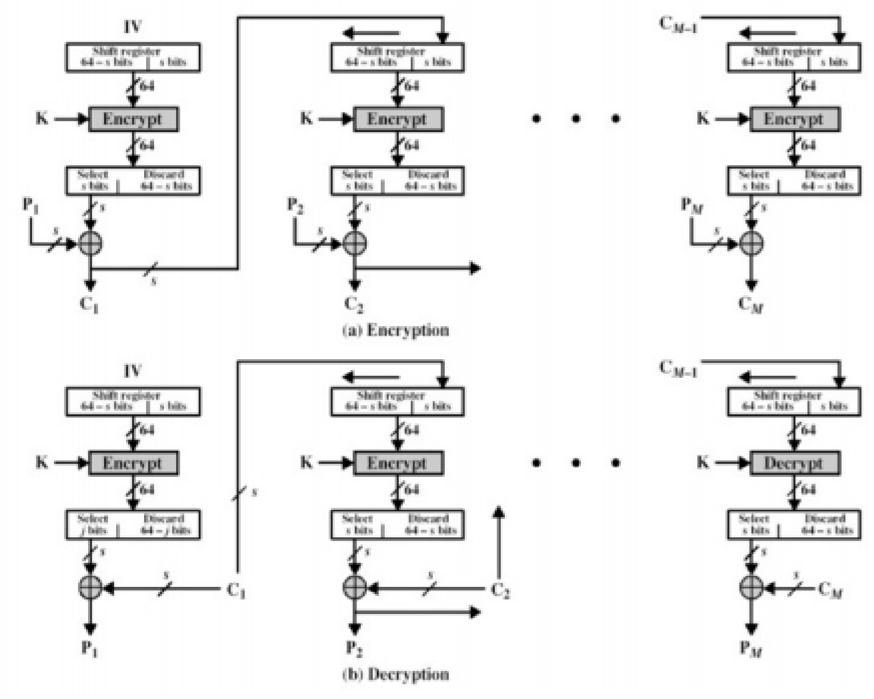


As with encryption, decryption is performed one block at a time, always using the same key. The ECB method is ideal for small amounts of data such as an encryption key however for larger messages if the same plaintext block appears more than once then the same cipher text is produced. This may assist an attacker.

Cipher Block Chaining (CBC) Mode: Cipher Block Chaining allows this by XORing each plaintext with the cipher text from the previous round (the first round using an Initialisation Vector (IV)). As before, the same key is used for each block. Decryption works as shown in the figure because of the properties of the XOR operation, i.e. IV ⊕ IV ⊕ P = P where IV is the Initialisation Vector and P is the plaintext. Obviously the IV needs to be known by both sender and receiver and it should be kept secret along with the key for maximum security.

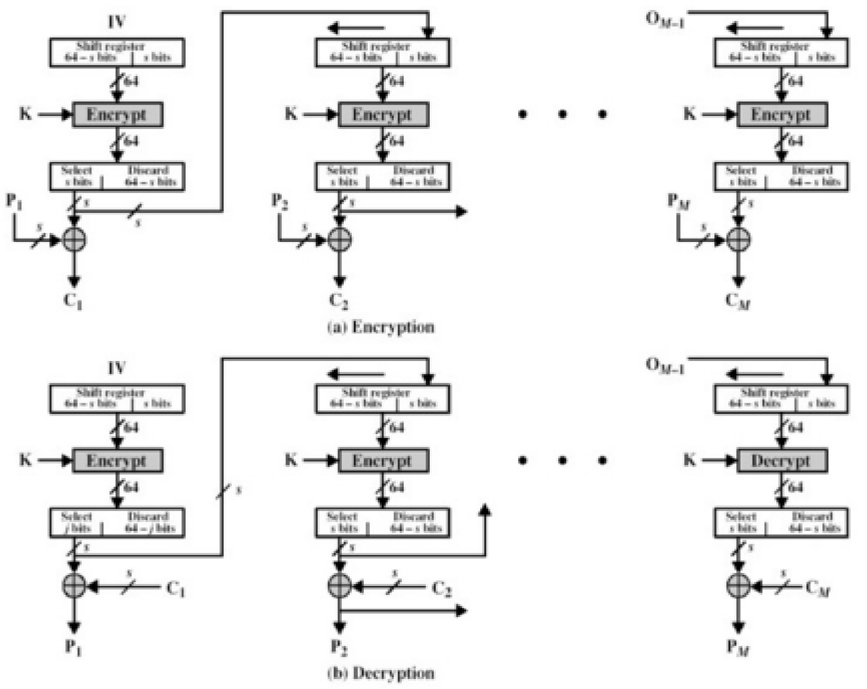


Cipher Feedback Mode: The Cipher Feedback and Output Feedback allows a block cipher to be converted into a stream cipher. This eliminates the need to pad a message to be an integral number of blocks. It also can operate in real time. Here it is assumed that the unit of transmission is s bits; a common value is s = 8. As with CBC, the units of plaintext are chained together, so that the cipher text of any plaintext unit is a function of all the preceding plaintext (which is split into s bit segments). The input to the encryption function is a shift register equal in length to the block cipher of the algorithm (although the diagram shows 64 bits, which is block size used by DES, this can be extended to other block sizes such as the 128 bits of AES).

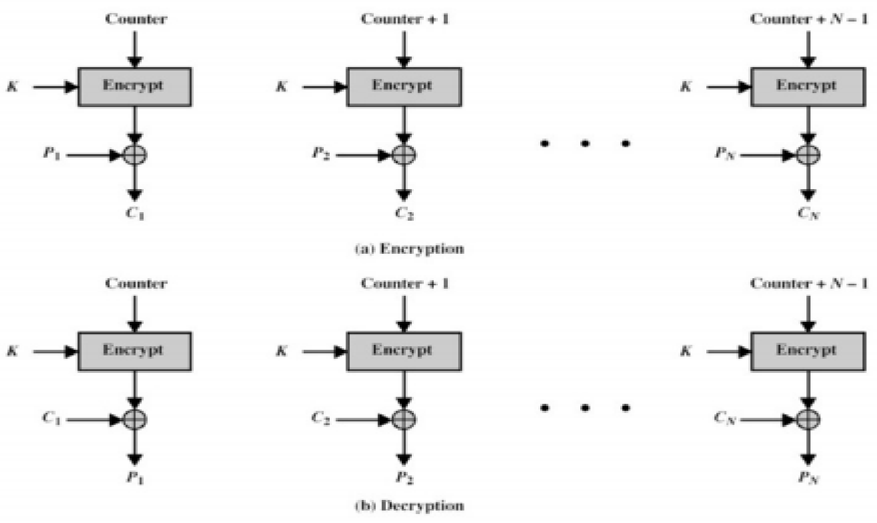


This is initially set to some Initialisation Vector (IV). The leftmost s bits of the output of the encryption function are XORed with the first segment of plaintext P1 (also s bits) to produce the first unit of cipher text C1 which is then transmitted. In addition, the contents of the shift register are shifted left by s bits and C1 is placed in the rightmost (least significant) s bits of the shift register. This process continues until all plaintext units have been encrypted. Decryption is similar.

Output Feedback Mode (OFB): The Output Feedback Mode is similar in structure to that of CFB, As can be seen, it is the output of the encryption function that is fed back to the shift register in OFB, whereas in CFB the cipher text unit is fed back to the shift register. One advantage of the OFB method is that bit errors in transmission do not propagate. For example, if a bit error occurs in C1 only the recovered value of P1 is affected; subsequent plaintext units are not corrupted. With CFB, C1 also serves as input to the shift register and therefore causes additional corruption downstream.



Counter Mode: This is a newer mode that was not listed initially with the above four. Interest in this mode has increased a good deal lately. A counter, equal to the plaintext block size is used. The only requirement stated in the standard is that the counter value must be different for each plaintext block that is encrypted. Typically, this counter is initialised to some value and then incremented by 1 for each subsequent block (modulo 2b where b is the block size). For encryption, the counter is encrypted and then XORed with the plaintext to produce the cipher text block; there is no chaining. For decryption, the same sequence of counter values is used, with each encrypted counter XORed with a cipher text block to recover the corresponding plaintext block. This mode contains a number of advantages including hardware efficiency, software efficiency, provable security and simplicity.

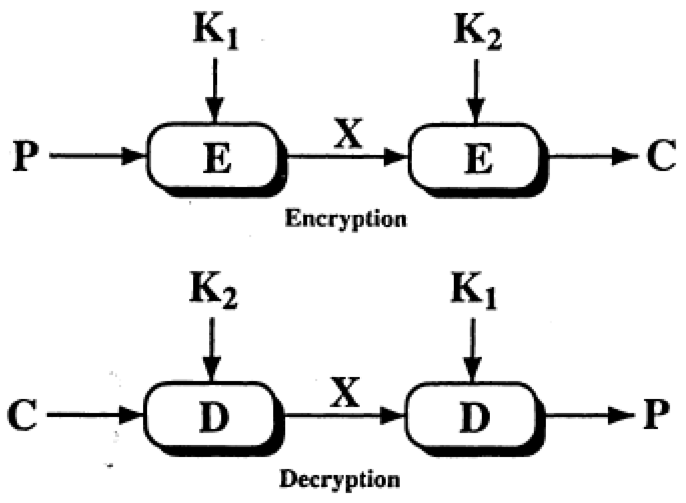


Double DES : Given a plain text P and two encryption keys K_1 and K_2, cipher text C

$$C=Ek_2[Ek_1[P]]$$

Decryption requires that the keys need to be applied in reverse order

$$P= Dk_1[Dk_2[P]]$$



(a) Double Encryption

For any two possible keys k1 and k2 it is possible to find a k3 such that

$$EK2[e=EK1[P]]=EK3[p]$$

Meet in the Middle Attack(MITM): This is a type of attack that can exponentially reduce the number of brute force permutations required to decrypt text that is encrypted with more than one key. It makes much easier for an intruder to gain access to data.

$$C=Ek_2[Ek_1[P]]$$

$$X=EK1[P]=DK2[C]$$

Triple DES(with two keys): To counter meet in the middle attack, this algorithm uses three stages of encryption with two different keys.

$$C=EK1[DK2[EK1[P]]]$$

$$P=DK1[EK2[DK1[C]]]$$

This was proposed by Tuchman EDE. It is relatively popular and alternative to DES.

Triple DES with three keys: It has an effective key length of 168 bits.

$$C=EK3[DK2[EK1[P]]]$$

$$P=DK1[EK2[DK3[C]]]$$

International Data Encryption Algorithm: A symmetric block cipher developed by Xuejia Lai and James Massey of Swiss Federal Institute of Technology.

Plain text block size – 64 bits

Key Size – 128 bits

No of Rounds -8 + one output transformation round

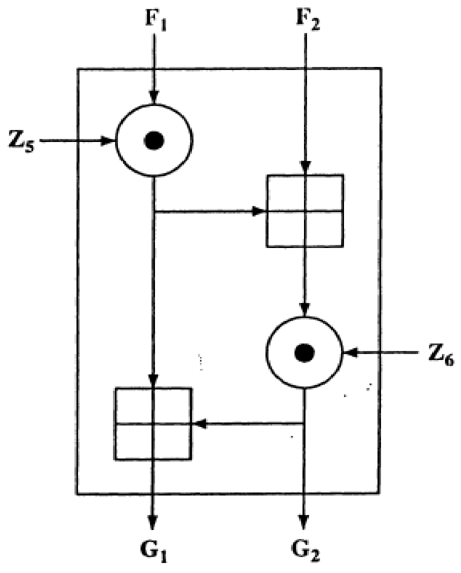
Sub keys – Each rounds needs six sub keys of length 16 bits. The output transformation rounds needs another 4 sub keys. So the total number of sub keys are 8x6=48+4=52 sub keys. They are represented with Z1,Z2,....Z52

Algorithm consists of the following functions.

Confusion: Cipher text should depend on plain text and key in a complicated and involved way. This is achieved by mixing three different operations. Each operation is performed on two 16-bit inputs.

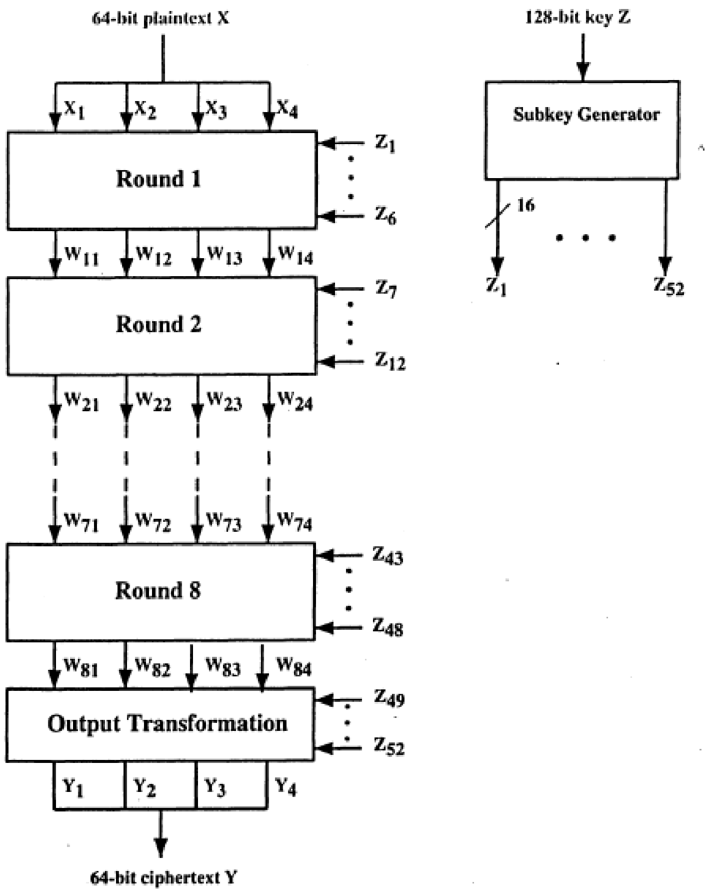
1. Bit by bit XOR
2. Addition of integer modulo 2^{16}
3. Multiplication of integer modulo 2^{16}

Diffusion: Each lain text bit influence every cipher text bit and key bit should influence cipher text bit. This is achieved by multiplication/addition (M/A) structure.



Multiplication/Addition (MA) Structure.

IDEA Encryption: It consists of eight (8) rounds followed by an output transformation round. It divides the input into four 16-bit blocks. Each round takes 4 16-bit blocks as an input and produces 4 16-bit blocks, which are concatenated to form cipher text. Each round uses six 16-bit sub keys, whereas the last output round uses only 4 sub keys, for a total of 52 sub keys.

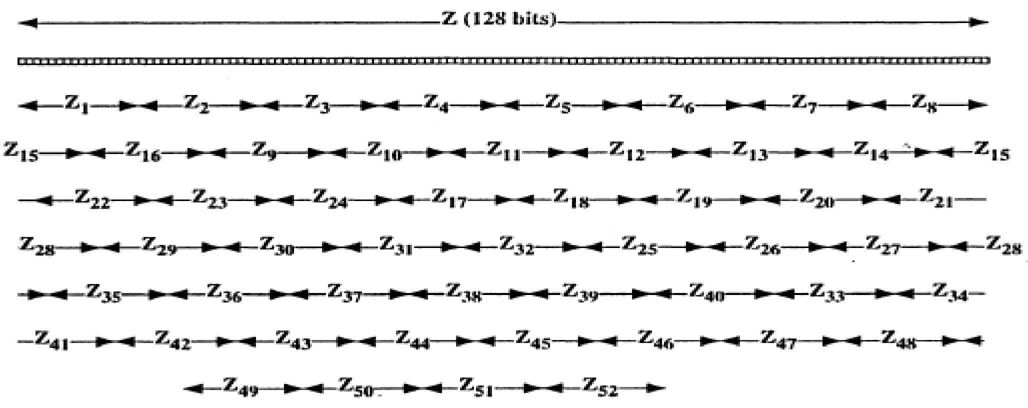


Sub key generation: 52, 16-bit sub keys are generated from the given 128 bit key. The procedure is described as follows

- First 8 sub keys are generated using the given 128 bit key that is from Z1 to Z8.

  Z1 : First 16 bits (most significant bits)

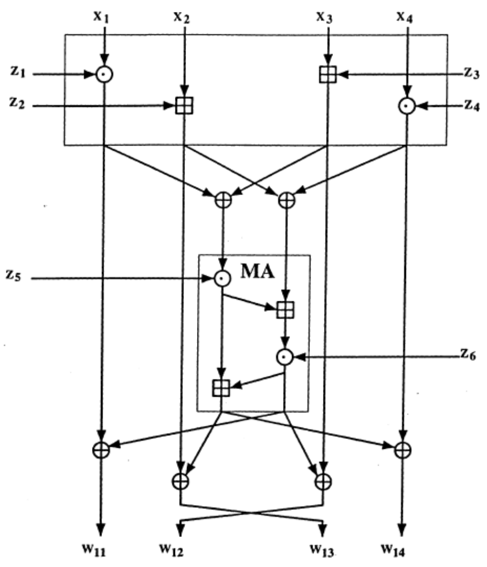
  Z2: Next 16 bits -----Z8: Last 16 bits (least significant bits)
- Apply a circular left shift of 25 bits on the fiven key and select another set of 8 sub keys from Z9 to Z16
- Continue step 2 till you get 52 sub keys


Z (128 bits)
Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8
Z15 Z16 Z9 Z10 Z11 Z12 Z13 Z14 Z15
Z22 Z23 Z24 Z17 Z18 Z19 Z20 Z21
Z28 Z29 Z30 Z31 Z32 Z25 Z26 Z27 Z28
Z35 Z36 Z37 Z38 Z39 Z40 Z33 Z34
Z41 Z42 Z43 Z44 Z45 Z46 Z47 Z48
Z49 Z50 Z51 Z52

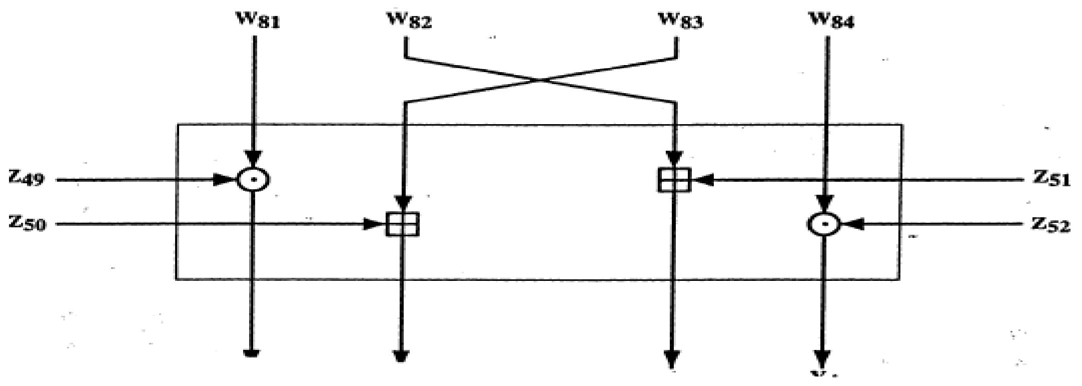
IDEA Subkeys.

Single Round Details:

- The round begins with a transformation that combines 4 16-bit input sub blocks with 4 16-bit sub keys using $\boxplus$ and $\odot$operations.
- The four output blocks of this transformation are then combined using $\boxplus$ to form 2 16-bit blocks that are taken as input to MA structure.
- MA structure also takes two sub keys as input and combines these inputs to produce two 16-bit outputs.
  - Finally the four output blocks from upper transformation are combined with two output blocks of MA structure using XOR to produce 4 output blocks in every round



The output transformation round also uses the same functionality of transformation round. The only difference is the second and third inputs are interchanged before this round. Means undoing the interchange after the 8th round. The reason for this extra interchange is so that the decryption has same structure as encryption.



Blowfish: A symmetric block cipher developed by Bruce Shiner. It encrypts data on 32 bit microprocessors at a rate of 19 clock cycles per byte.

Block size-64 bits

Key size- 32 bits to 448 bits(1 to 14 words)

No of Rounds – 16

Sub Key and S-Box Generation: It uses 32 to 448 bits key. This is used to generate 18 32-bit sub key and four 8x32 s-boxes which contains 1024 entries. The size of each s-box element is 32-bit. The keys are stored in a k-array K1,K2,K3, … ,Kj (1<=j<=14) ,

The sub keys are stored in P-array P1,P2,P3, … P18. There are four s-boxes, each with 256 32-bit entries:

S_{10},S_{11},.... $S_{1,255}$

S_{20},S_{21},....$S_{2,255}$

S_{30},S_{31},....$S_{3,255}$

S_{40},S_{41},....$S_{4,255}$

To generate these P-array and s-boxes the following procedure is adopted:

1. Initialize the p-array and then four s-boxes using the fractional part of PI. So, the left most 32-bits of PI becomes P1 and so on.

   Ex: in hexadecimal

   P1=243F6!88

   P2=...

   S_{4254}=578FdFE3

   S_{4255}=3AC372E6

2. Perform bitwise XOR of p-array and k-array i.e,

   P1=P1$\oplus$K1

   P2=P1$\oplus$K2

   .....

   P14=P14$\oplus$K14

   P15=P15$\oplus$K1

   P16=P16$\oplus$K2

   P17=P17$\oplus$K3

   P18=P16$\oplus$K4 (since K-array has only 14 keys)

3. Encrypt 64-bit block of all 0's using current p-array and s-boxes to replace P1&P2 with the output of encryption.

4. Encrypt the output of step-3 using current p-array and s-arrays to replace P3 and P4 with resulting cipher text.
5. Continue the above step to update all p and s-box elements.

P1P2=Ep,s[0]

P3P4=Eps[P1||P2]

…

P17P18=Eps[P115||P16]

$S_{10}S_{11}$=Eps[P17||P18]

…

S_{4254},S_{4255}=Eps[S_{4252}||S_{4253}]

Encryption-Decryption: Uses two primitive operations

1. Addition Integer Modulo 2^{32} $\boxplus$
2. Bitwise XOR $\oplus$

The plain text is divided into two 32-bit halves LE0 and RE0. We use Lei, REi to refer each round output. For i=1 to 16 do

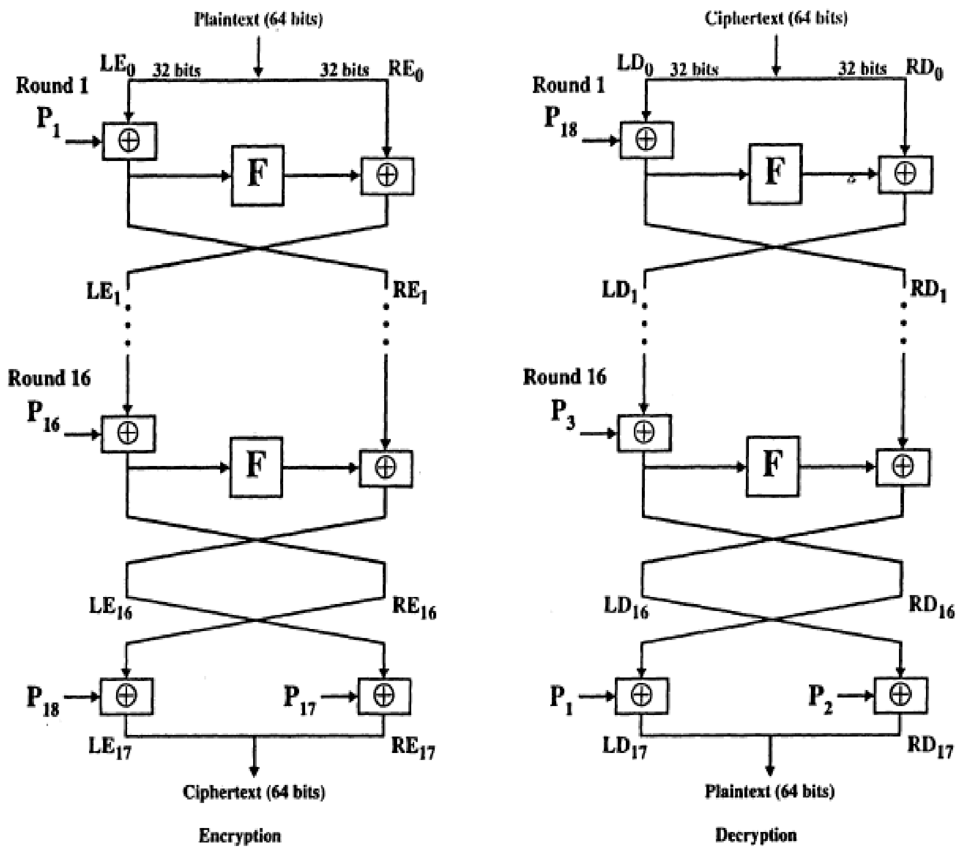
$$\begin{aligned} RE_i &= LE_{i-1} \oplus P_i; \\ LE_i &= F[RE_i] \oplus RE_{i-1}; \\ LE_{17} &= RE_{16} \oplus P_{18}; \\ RE_{17} &= LE_{16} \oplus P_{17}; \end{aligned}$$

The function Fid divided into 4 parts where each part uses 8 bit input and these bytes are represented as a,d,c,d.

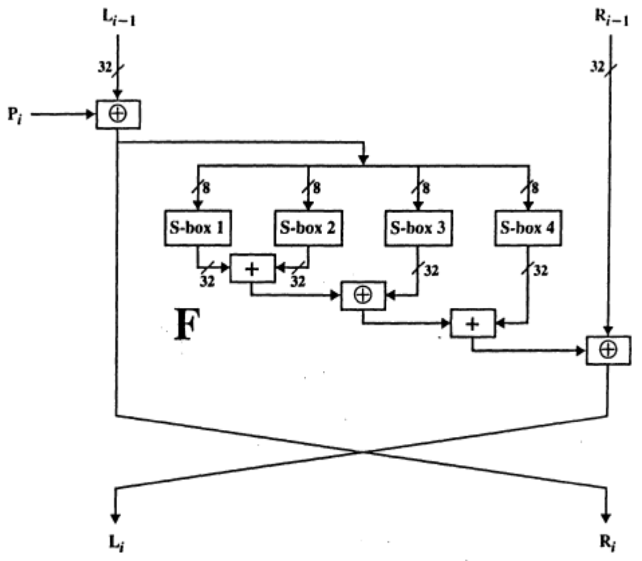
F[a,b,c,d]=((S1a $\boxplus$ S1b) $\oplus$ S1c) $\boxplus$ S1d)

For decryption, for i=1 to 16 do

$$\begin{aligned} RD_i &= LD_{i-1} \oplus P_{19-i}; \\ LD_i &= F[RD_i] \oplus RD_{i-1}; \\ LD_{17} &= RD_{16} \oplus P_1; \\ RD_{17} &= LD_{16} \oplus P_2; \end{aligned}$$



Blowfish Encryption and Decryption.



Detail of Single Blowfish Round.

CAST-128: This algorithm was developed by Carlisle Adams and Stafford Tavares. The components are: Block size-64 bit

Key size- 40 bits-128 bits(in 8-bit increment)

No of Rounds 16

Each round uses two sub-keys (1-masking sub key and 1-rotating sub key. Therefore the total sub keys are 32 out of which 16 are masking and 16 are rotating sub keys.e ach sub key of length 32-bits Kmi and a 5-bit Kri. The function F depends on each round.

Encryption: It uses four primitive operations.

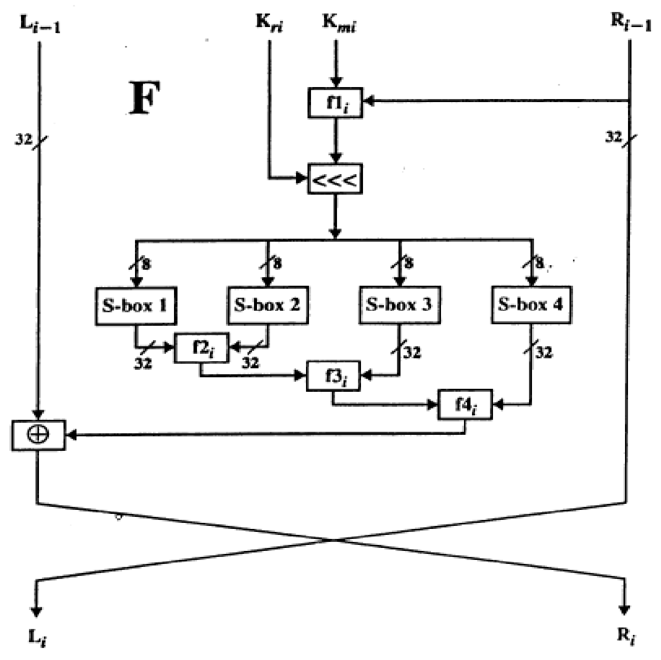
1. Addition Integer Modulo 2^{32} +
2. Subtraction Integer Modulo 2^{32} -
3. XOR
4. Left Circular rotation x<<<Y (rotation of word x by y number of bits)

Details of Single Round: The plain text is divided into two 32-bit halves L0 and R0. The cipher text is formed by swapping the output of 16th round that is,

```
$L_0 \| R_0 = \text{Plaintext}$
for $i$ = 1 to 16 do
        $L_i = R_{i-1}$;
        $R_i = L_{i-1} \oplus F_i[R_{i-1}, Km_i, Kr_i]$;
$\text{Ciphertext} = R_{16} \| L_{16}$
```

Function F uses four 8x32 s-boxes, the <<< function, and 4 functions that depend on round number. These functions are labelled as f1,f2,f3 and f4. We use function I to refer to the Intermediate 32-bit value after left circular rotation, and labels them as Ia,Ib,Ic,Id and they refer as I

Rounds 1, 4, 7, 10, 13, 16	$I = ((Km_i + R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] \oplus S2[Ib]) - S3[Ic]) + S4[Id]$
Rounds 2, 5, 8, 11, 14	$I = ((Km_i \oplus R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] - S2[Ib]) + S3[Ic]) \oplus S4[Id]$
Rounds 3, 6, 9, 12, 15	$I = ((Km_i - R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] + S2[Ib]) \oplus S3[Ic]) - S4[Id]$



Detail of Single CAST-128 Round.

S-boxes and Sub Key Generation: CAST 128 uses eight 8x32 s=boxes. Four of these, s-box1 through s-box4 are used in encryption/decryption. The remaining four S5 to S8 are used in sub key generation. Each s-box contains 256 elements. Each s-box takes an 8-bit input which signifies the index of element. All these s-boxes contain fixed values. These values are generated with bent functions.

1. Label the bytes of 128-bit key as

   X0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF

   X0 is the most significant byte

   ..

   XF is the least significant byte

   Km1---Km16 → sixteen 32-bit masking sub keys-one per round

   Kr1.....Kr16 → sixteen 2-bit rotating sub keys – one per round

Z0--------------ZF Intermediate bytes

K1--------------K32m Intermediate 32-nit words.

For i=1 to 16 do

Kmi=Ki

Kri=K16+i

```
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K1  = S5[z8] ^ S6[z9] ^ S7[z7] ^ S8[z6] ^ S5[z2]
K2  = S5[zA] ^ S6[zB] ^ S7[z5] ^ S8[z4] ^ S6[z6]
K3  = S5[zC] ^ S6[zD] ^ S7[z3] ^ S8[z2] ^ S7[z9]
```

```
K4  = S5[zE] ^ S6[zF] ^ S7[z1] ^ S8[z0] ^ S8[zC]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K5  = S5[x3] ^ S6[x2] ^ S7[xC] ^ S8[xD] ^ S5[x8]
K6  = S5[x1] ^ S6[x0] ^ S7[xE] ^ S8[xF] ^ S6[xD]
K7  = S5[x7] ^ S6[x6] ^ S7[x8] ^ S8[x9] ^ S7[x3]
K8  = S5[x5] ^ S6[x4] ^ S7[xA] ^ S8[xB] ^ S8[x7]
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K9  = S5[z3] ^ S6[z2] ^ S7[zC] ^ S8[zD] ^ S5[z9]
K10 = S5[z1] ^ S6[z0] ^ S7[zE] ^ S8[zF] ^ S6[zC]
K11 = S5[z7] ^ S6[z6] ^ S7[z8] ^ S8[z9] ^ S7[z2]
K12 = S5[z5] ^ S6[z4] ^ S7[zA] ^ S8[zB] ^ S8[z6]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K13 = S5[x8] ^ S6[x9] ^ S7[x7] ^ S8[x6] ^ S5[x3]
K14 = S5[xA] ^ S6[xB] ^ S7[x5] ^ S8[x4] ^ S6[x7]
K15 = S5[xC] ^ S6[xD] ^ S7[x3] ^ S8[x2] ^ S7[x8]
K16 = S5[xE] ^ S6[xF] ^ S7[x1] ^ S8[x0] ^ S8[xD]
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K17 = S5[z8] ^ S6[z9] ^ S7[z7] ^ S8[z6] ^ S5[z2]
K18 = S5[zA] ^ S6[zB] ^ S7[z5] ^ S8[z4] ^ S6[z6]
K19 = S5[zC] ^ S6[zD] ^ S7[z3] ^ S8[z2] ^ S7[z9]
K20 = S5[zE] ^ S6[zF] ^ S7[z1] ^ S8[z0] ^ S8[zC]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K21 = S5[x3] ^ S6[x2] ^ S7[xC] ^ S8[xD] ^ S5[x8]
K22 = S5[x1] ^ S6[x0] ^ S7[xE] ^ S8[xF] ^ S6[xD]
K23 = S5[x7] ^ S6[x6] ^ S7[x8] ^ S8[x9] ^ S7[x3]
K24 = S5[x5] ^ S6[x4] ^ S7[xA] ^ S8[xB] ^ S8[x7]
z0z1z2z3 = x0x1x2x3 ^ S5[xD] ^ S6[xF] ^ S7[xC] ^ S8[xE] ^ S7[x8]
z4z5z6z7 = x8x9xAxB ^ S5[z0] ^ S6[z2] ^ S7[z1] ^ S8[z3] ^ S8[xA]
z8z9zAzB = xCxDxExF ^ S5[z7] ^ S6[z6] ^ S7[z5] ^ S8[z4] ^ S5[x9]
zCzDzEzF = x4x5x6x7 ^ S5[zA] ^ S6[z9] ^ S7[zB] ^ S8[z8] ^ S6[xB]
K25 = S5[z3] ^ S6[z2] ^ S7[zC] ^ S8[zD] ^ S5[z9]
K26 = S5[z1] ^ S6[z0] ^ S7[zE] ^ S8[zF] ^ S6[zC]
K27 = S5[z7] ^ S6[z6] ^ S7[z8] ^ S8[z9] ^ S7[z2]
K28 = S5[z5] ^ S6[z4] ^ S7[zA] ^ S8[zB] ^ S8[z6]
x0x1x2x3 = z8z9zAzB ^ S5[z5] ^ S6[z7] ^ S7[z4] ^ S8[z6] ^ S7[z0]
x4x5x6x7 = z0z1z2z3 ^ S5[x0] ^ S6[x2] ^ S7[x1] ^ S8[x3] ^ S8[z2]
x8x9xAxB = z4z5z6z7 ^ S5[x7] ^ S6[x6] ^ S7[x5] ^ S8[x4] ^ S5[z1]
xCxDxExF = zCzDzEzF ^ S5[xA] ^ S6[x9] ^ S7[xB] ^ S8[x8] ^ S6[z3]
K29 = S5[x8] ^ S6[x9] ^ S7[x7] ^ S8[x6] ^ S5[x3]
```

```
K30 = S5[xA] ^ S6[xB] ^ S7[x5] ^ S8[x4] ^ S6[x7]
K31 = S5[xC] ^ S6[xD] ^ S7[x3] ^ S8[x2] ^ S7[x8]
K32 = S5[xE] ^ S6[xF] ^ S7[x1] ^ S8[x0] ^ S8[xD]
```

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

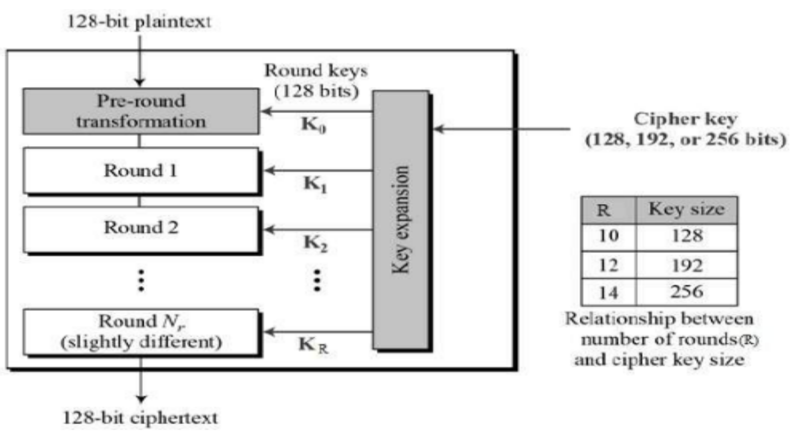
A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

The features of AES are as follows:

1. Symmetric key symmetric block cipher
2. 128-bit data, 128/192/256-bit keys,
3. Stronger and faster than Triple-DES
4. Provide full specification and design details
5. Software implementable in C and Java.

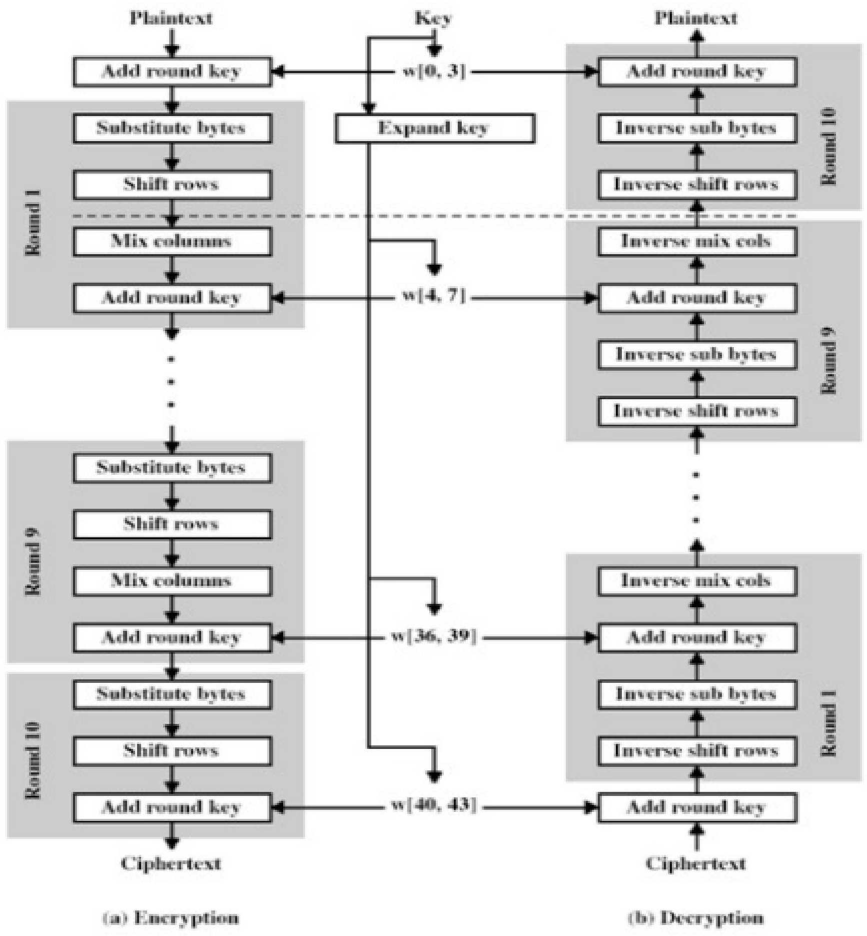
AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix. Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key. The schematic of AES structure is given in the following illustration –



R	Key size
10	128
12	192
14	256

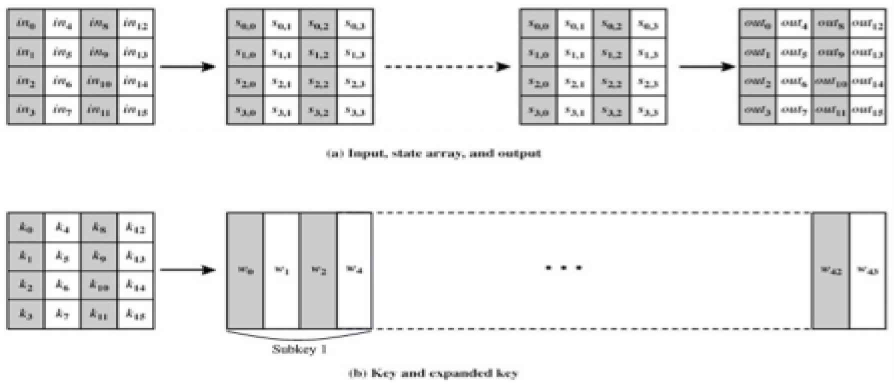
Relationship between number of rounds(R) and cipher key size



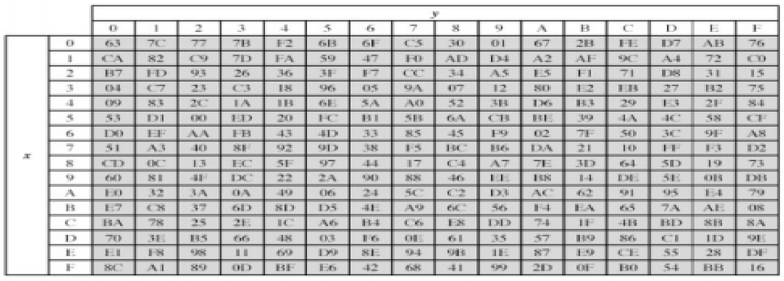
Encryption Process: Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below –

Byte Substitution (Sub Bytes):

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

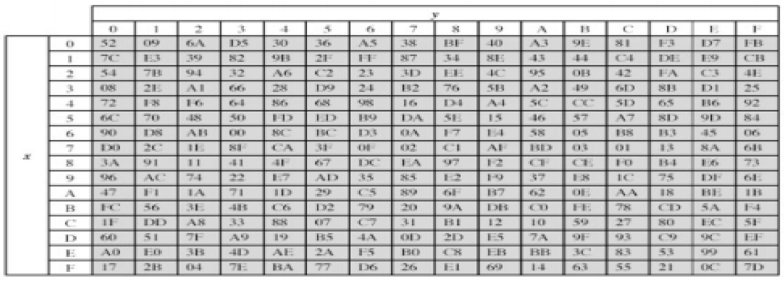


(a) S-box

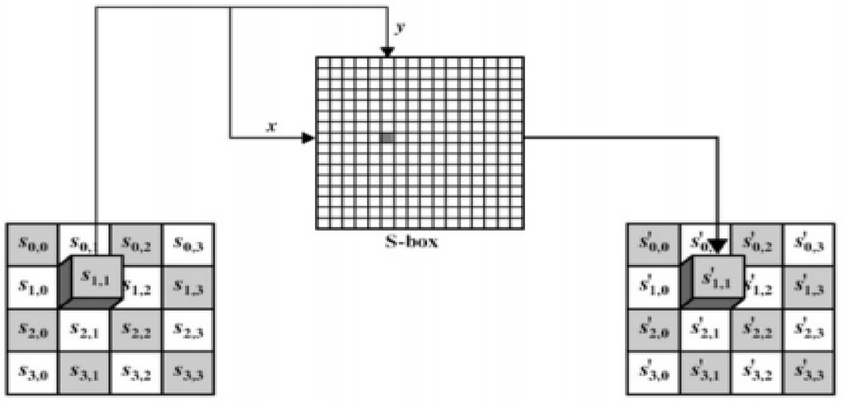


x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED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

(b) Inverse S-box



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


y
x
S-box
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s'_{0,0}$ $s'_{0,1}$ $s'_{0,2}$ $s'_{0,3}$
$s'_{1,0}$ $s'_{1,1}$ $s'_{1,2}$ $s'_{1,3}$
$s'_{2,0}$ $s'_{2,1}$ $s'_{2,2}$ $s'_{2,3}$
$s'_{3,0}$ $s'_{3,1}$ $s'_{3,2}$ $s'_{3,3}$

Shift rows

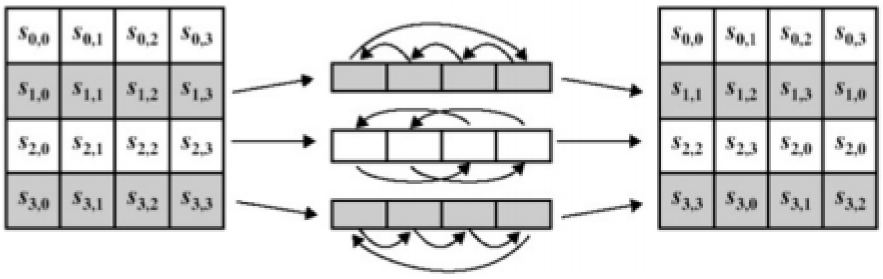
Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows –

First row is not shifted.

Second row is shifted one (byte) position to the left.

Third row is shifted two positions to the left.

Fourth row is shifted three positions to the left.

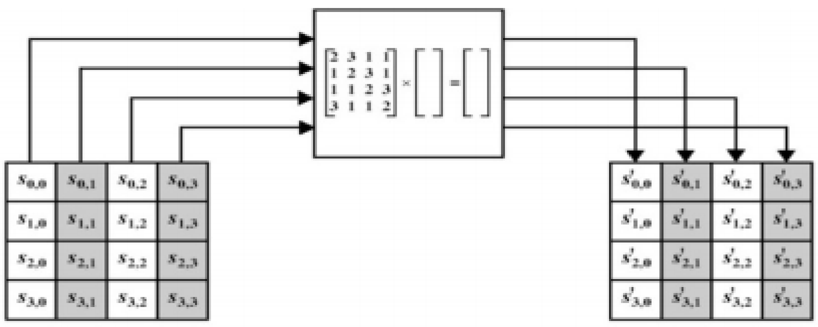

$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,0}$ $s_{1,1}$ $s_{1,2}$ $s_{1,3}$
$s_{2,0}$ $s_{2,1}$ $s_{2,2}$ $s_{2,3}$
$s_{3,0}$ $s_{3,1}$ $s_{3,2}$ $s_{3,3}$
$s_{0,0}$ $s_{0,1}$ $s_{0,2}$ $s_{0,3}$
$s_{1,1}$ $s_{1,2}$ $s_{1,3}$ $s_{1,0}$
$s_{2,2}$ $s_{2,3}$ $s_{2,0}$ $s_{2,0}$
$s_{3,3}$ $s_{3,0}$ $s_{3,1}$ $s_{3,2}$

The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

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

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

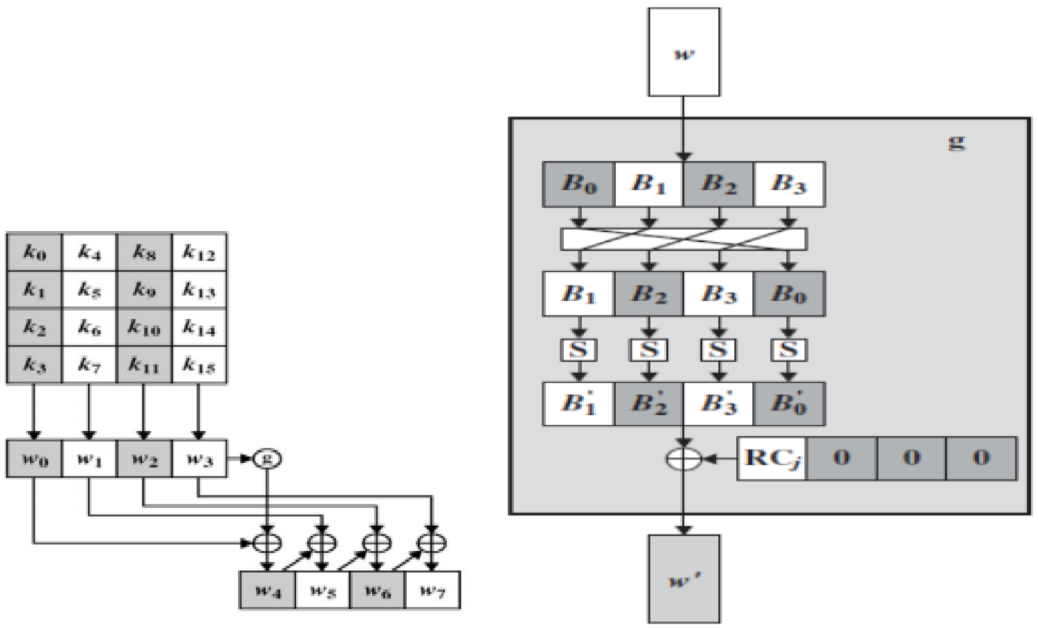


Add round key

In this stage (known as Add Round Key) the 128 bits of state are bitwise XORed with the 128 bits of the round key. The operation is viewed as a column wise operation between the 4 bytes of a state column and one word of the round key. This transformation is as simple as possible which helps in efficiency but it also effects every bit of state. If this is the last round then the output is the cipher text. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

AES Key Expansion: The AES key expansion algorithm takes as input a 4-word key and produces a linear array of 44 words. Each round uses 4 of these words as shown in figure 7.2. Each word contains 32 bytes which means each sub key is 128 bits long. Figure 7.7 show pseudo code for generating the expanded key from the actual key.

```
KeyExpansion (byte key[16], word w[44])
{
    word temp
    for ($i = 0; i < 4; i++$)    w[$i$] = (key[$4*i$], key[$4*i+1$], key[$4*i+2$], key[$4*i+3$]);
    for ($i = 4; i < 44; i++$)
    {
        temp = w[$i-1$];
        if ($i$ mod $4 = 0$)    temp = SubWord (RotWord(temp)) $\oplus$ Rcon[$i/4$];
        w[$i$] = w[1 – 4] $\oplus$ temp;
    }
}
```


k_0 k_4 k_8 k_{12}
k_1 k_5 k_9 k_{13}
k_2 k_6 k_{10} k_{14}
k_3 k_7 k_{11} k_{15}
w_0 w_1 w_2 w_3
g
w_4 w_5 w_6 w_7
w
g
B_0 B_1 B_2 B_3
B_1 B_2 B_3 B_0
S S S S
B_1' B_2' B_3' B_0'
RC_j 0 0 0
w'

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]. The function g consists of the following sub functions:

1. Rot Word performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].

2. Sub Word performs a byte substitution on each byte of its input word, using the s-box described earlier.

3. The result of steps 1 and 2 is XORed with round constant, Rcon[j]

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

The round constant is a word in which the three rightmost bytes are always 0. Thus 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• RC[j – 1] and with multiplication defined over the field GF(2 8). The key expansion was designed to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the way in which round keys are generated in different rounds

Decryption Process

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

Add round key

Mix columns

Shift rows

Byte substitution

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

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

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