Name: Sanaullah

Program: MS(cs)

Roll Number: 14971

Paper: Advance Network Security

Teacher: Dr.Sheeraz Ahmad

Abstract

Modern Technology like Wi-Fi network helps us to join right away with humans somewhere and at any time. Security of Wireless community is the fundamental mission faced by way of today's world. It is where cryptography plays an essential position to furnish security to the Wi-Fi network. Numerous encryption calculations are handy to impenetrable the information. This paper deals with frequently utilized symmetric encryption calculation which is DES Algorithm. Test consequences are given to illustrate the execution of this calculation.

INTRODUCTION

Cryptography is an art of conveying messages in coded structure which is understood solely with the aid of the supposed recipient. The recipient in flip decodes to examine the message. The transfer of facts via public network with security problems can be blanketed with cryptography. There are quite a few widespread symmetric and asymmetric algorithms which are highly secured and time tested.

Types of Cryptography:

DES:

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

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

TDES:

Triple DES (aka 3DES, 3-DES, TDES) is based on the DES (Data Encryption Standard) algorithm, therefore it is very easy to modify existing software to use Triple DES. It also has the advantage of proven reliability and a longer key length that eliminates many of the attacks that can be used to reduce the amount of time it takes to break DES. However, even this more powerful version of DES may not be strong enough to protect data for very much longer (due in particular to the small block size). As such, the DES algorithm itself has become obsolete and is no longer used.

AES:

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.

The features of AES are as follows −

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

Blowfish:

Blowfish is a 64-bit block cipher invented by Bruce Schneider. Blowfish was designed for fast ciphering on 32-bit microprocessors. Blowfish is also compact and has a variable key length which can be increased to 448 bits. Data Encryption Standard (DES) 167 Blowfish is suitable for applications where the key does not change frequently like communication links or fi le encryptions. However for applications like packet switching or as an one-way hash function, it is unsuitable. Blowfish is not ideal for smart cards, which requires even more compact ciphers. Blowfish is faster than DES when implemented on 32-bit microprocessors.

RSA (Rivest–Shamir–Adleman)

Is one of the first public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and distinct from the decryption key which is kept secret (private). In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". The acronym RSA is the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who publicly described the algorithm in 1977.

3.1 DATA ENCRYPTION STANDARD

The DES has the following steps involved.

a) 64 bit plain text is taken as input and initial permutation is done on the input by re-arranging the bits to get the permuted input.

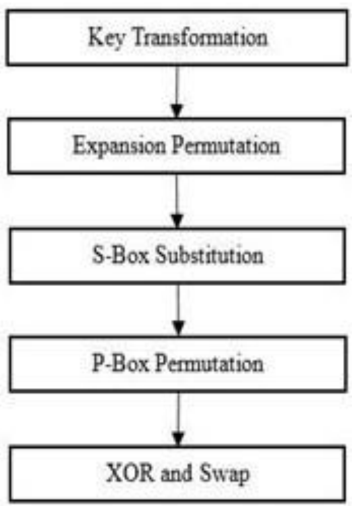
The next step involves 16 rounds of the same function along with permutation and substitution.

b) The 16^{t}h output contains 64 bits as a result of function of input plain text and key.

c) The output of left and right side are swapped producing the preoutis.

d) e. The preoutis have gone through IP, i.e Opposite of initial permutation to produce 64 bit cipher text.

SINGLE ROUND DES



Key Transformation:

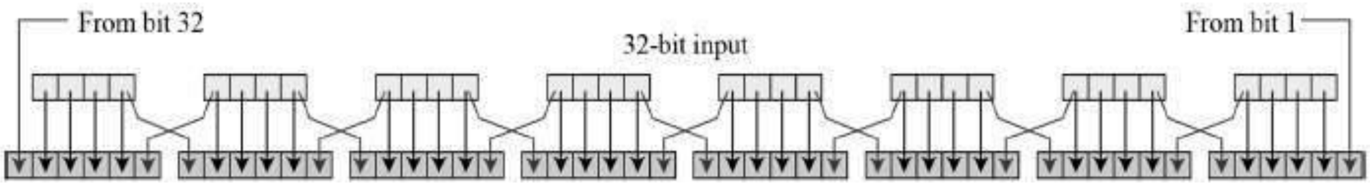
We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called as key transformation. For this the 56 bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.
For example, if the round number 1, 2, 9 or 16 the shift is done by only position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is show in figure.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

Expansion Permutation

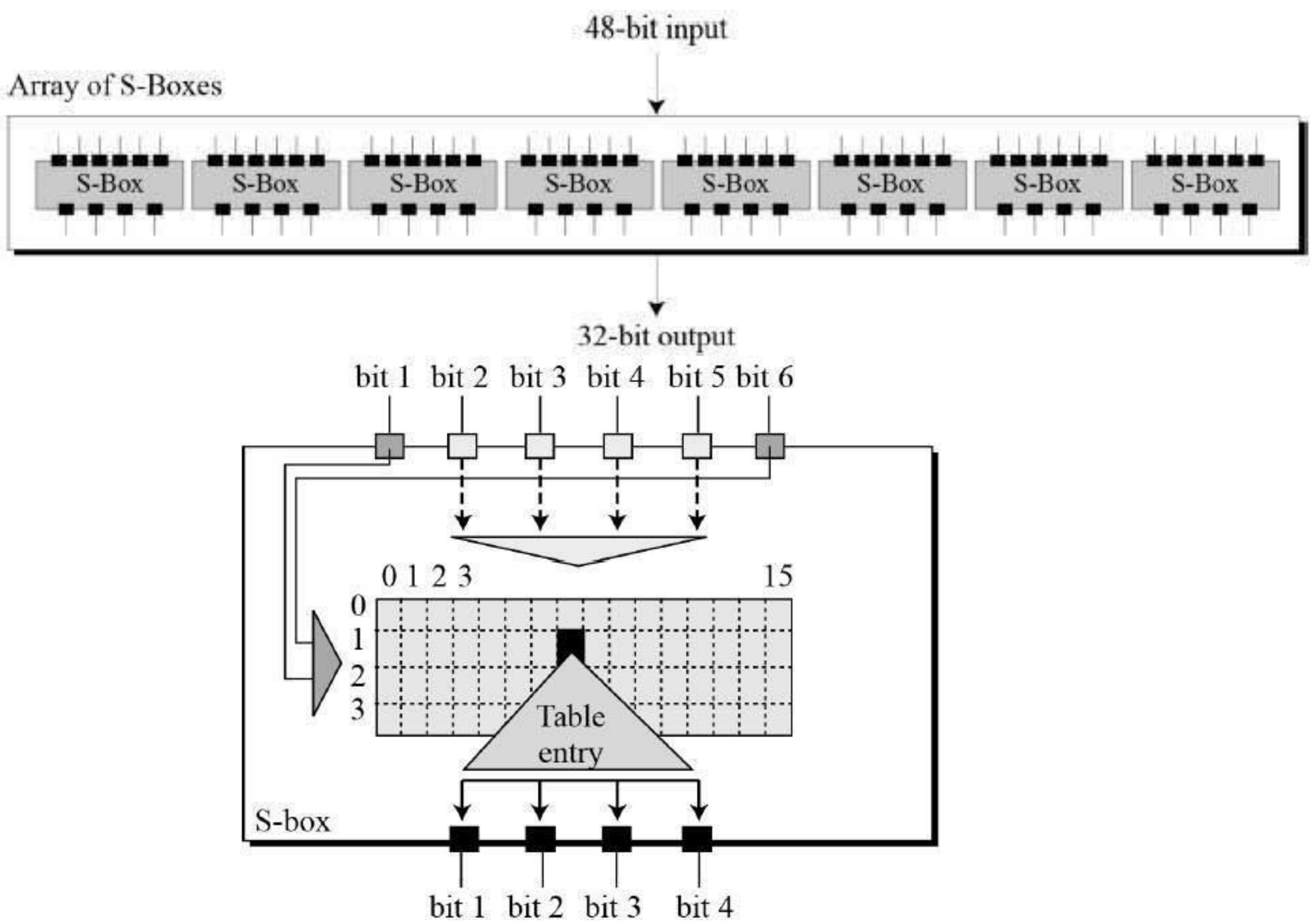
Recall that after initial permutation, we had two 32-bit plain text areas called as Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called as expansion permutation. This happens as the 32 bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4 bit block of the previous step is then expanded to a corresponding 6 bit block, i.e., per 4 bit block, 2 more bits are added.



S-box (substitution-box)

Is a basic component of symmetric key algorithms which performs substitution. In block ciphers, they are typically used to obscure the relationship between the key and the ciphertext — Shannon's property of confusion.

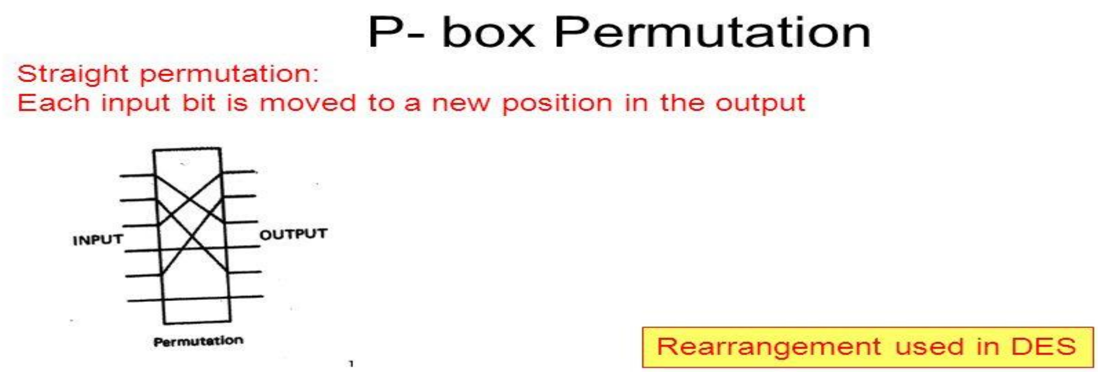
In general, an S-box takes some number of input bits, *m*, and transforms them into some number of output bits, *n*, where *n* is not necessarily equal to *m*.[1] An $m \times n$ S-box can be implemented as a lookup table with 2^m words of *n* bits each. Fixed tables are normally used, as in the Data Encryption Standard (DES), but in some ciphers the tables are generated dynamically from the key (e.g. the Blowfish and the Twofish encryption algorithms).



Permutation box (or **P-box**)

Is a method of bit-shuffling used to permute or transpose bits across S-boxes inputs, retaining diffusion while transposing.[1]

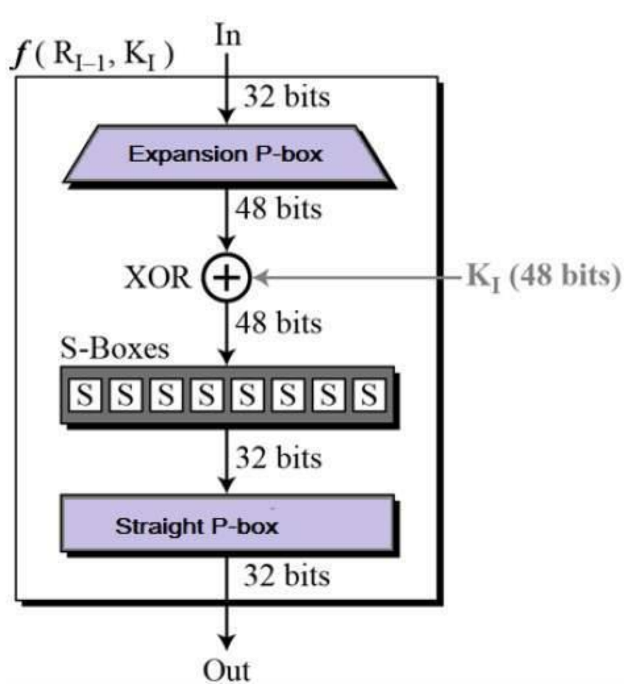
In block ciphers, the S-boxes and P-boxes are used to make the relation between the plaintext and the ciphertext difficult to understand (see Shannon's property of confusion). P-boxes are typically classified as *compression*, *expansion*, and *straight*, depending on whether the number of output bits is less than, greater than, or equal to the number of input bits. Only straight P-boxes are invertible.



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

XOR (Whitener).

After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.

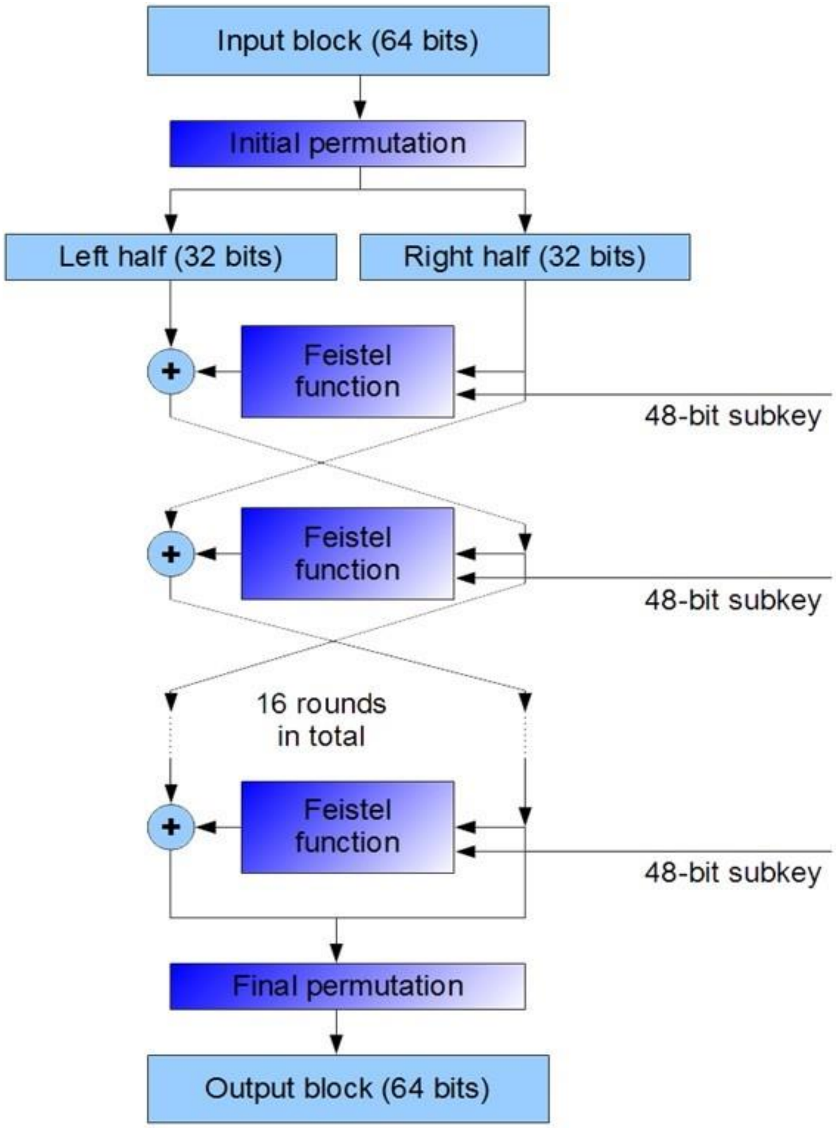


3.2 DES-Feistel Function:

Encryption Process:

The encryption process uses the Feistel structure consisting multiple rounds of processing of the plaintext, each round consisting of a “substitution” step followed by a permutation step.

Feistel Structure is shown in the following illustration.



- The input block to each round is divided into two halves that can be denoted as L and R for the left half and the right half.
- In each round, the right half of the block, R, goes through unchanged. But the left half, L, goes through an operation that depends on R and the encryption key. First, we apply an encrypting function ‘f’ that takes two input − the key K and R. The function produces the output f(R,K). Then, we XOR the output of the mathematical function with L.

- In real implementation of the Feistel Cipher, such as DES, instead of using the whole encryption key during each round, a round-dependent key (a subkey) is derived from the encryption key. This means that each round uses a different key, although all these subkeys are related to the original key.
- The permutation step at the end of each round swaps the modified L and unmodified R. Therefore, the L for the next round would be R of the current round. And R for the next round be the output L of the current round.
- Above substitution and permutation steps form a ‘round’. The numbers of rounds are specified by the algorithm design.
- Once the last round is completed then the two sub blocks, ‘R’ and ‘L’ are concatenated in this order to form the cipher text block.

The difficult part of designing a Feistel Cipher is selection of round function ‘f’. In order to be unbreakable scheme, this function needs to have several important properties that are beyond the scope of our discussion.

Decryption Process

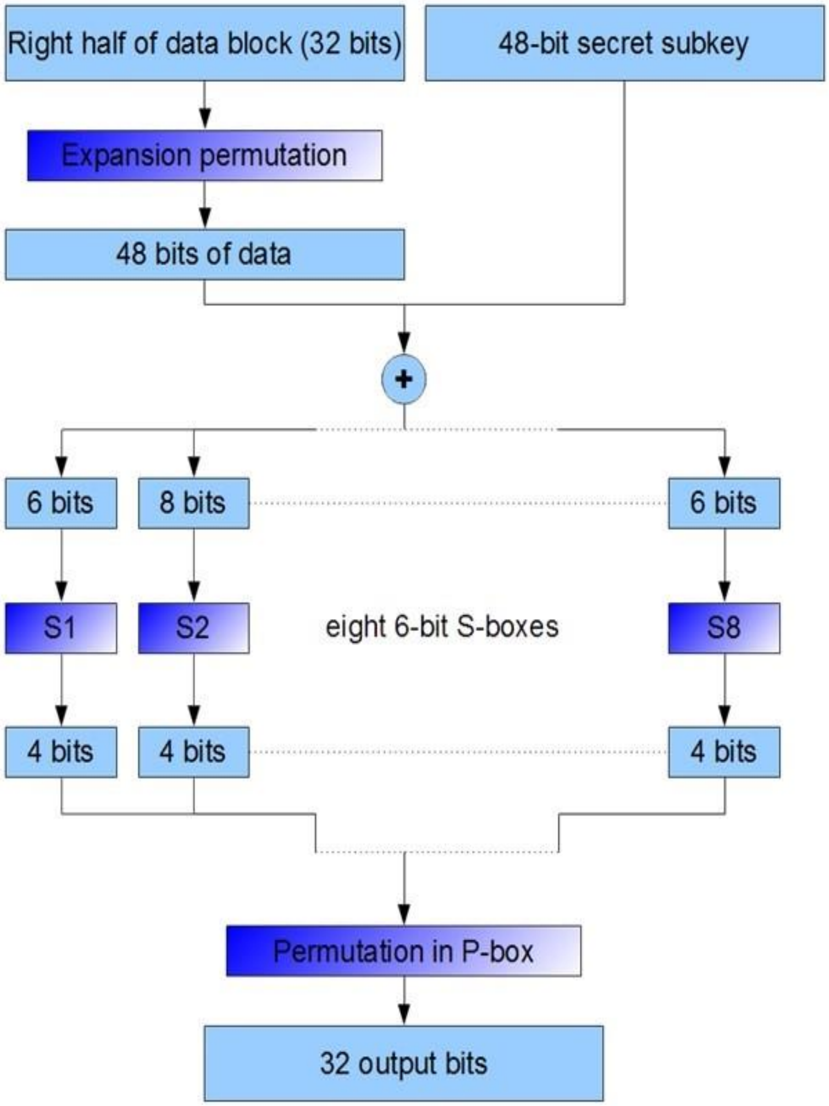
The process of decryption in Feistel cipher is almost similar. Instead of starting with a block of plaintext, the ciphertext block is fed into the start of the Feistel structure and then the process thereafter is exactly the same as described in the given illustration.

The process is said to be almost similar and not exactly same. In the case of decryption, the only difference is that the subkeys used in encryption are used in the reverse order.

The final swapping of ‘L’ and ‘R’ in last step of the Feistel Cipher is essential. If these are not swapped then the resulting ciphertext could not be decrypted using the same algorithm.

Figure 4:

The F-function, depicted in Figure 4, operates on half a block (32 bits) at a time and consists of four stages:

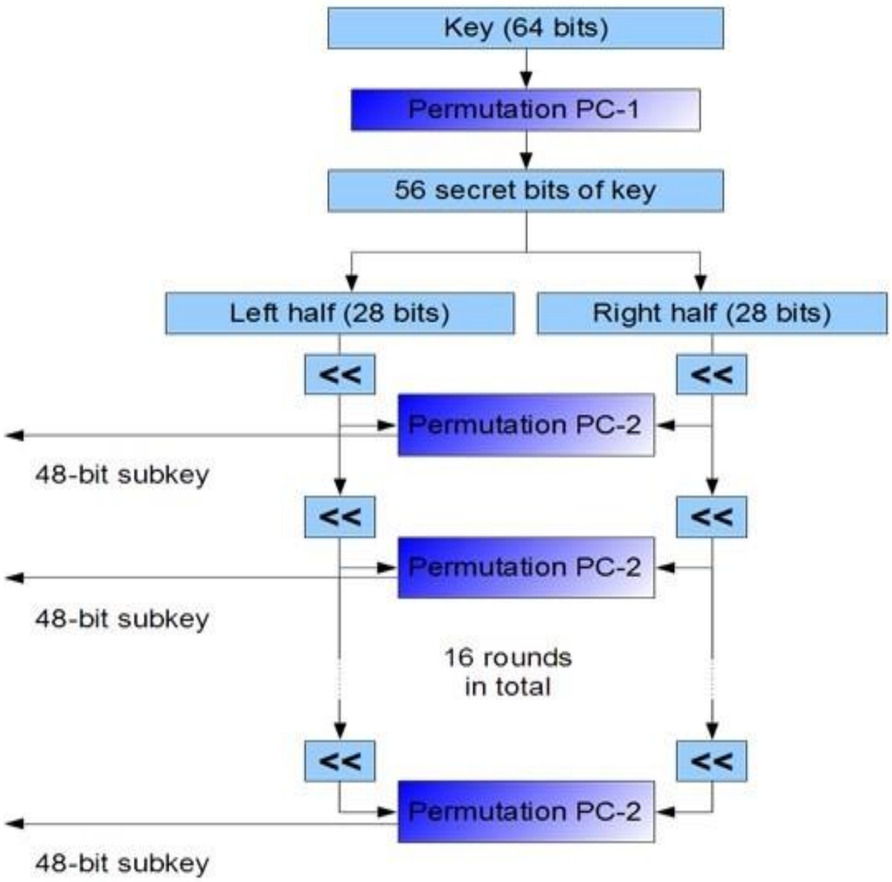


1. Expansion: the 32-bit half-block is expanded to 48 bits using the expansion permutation, denoted E in the diagram, by duplicating half of the bits. The output consists of eight 6-bit ($8 \times 6 = 48$ bits) pieces, each containing a copy of 4 corresponding input bits, plus a copy of the immediately adjacent bit from each of the input pieces to either side.
2. Key mixing: the result is combined with a subkey using an XOR operation. Sixteen 48-bit subkeys—one for each round—are derived from the main key using the key schedule (described below).
3. Substitution: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the S-boxes, or substitution boxes. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes provide the core of the security of DES—without them, the cipher would be linear, and trivially breakable.
4. Permutation: finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the P-box. This is designed so that, after permutation, the bits from the output of each S-box in this round are spread across four different S-boxes in the next round.

The alternation of substitution from the S-boxes, and permutation of bits from the P-box and E-expansion provides so-called "confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical cipher.

Figure 5:

Key-Schedule:



Illustrates the *key schedule* for encryption—the algorithm which generates the subkeys. Initially, 56 bits of the key are selected from the initial 64 by *Permuted Choice 1* (*PC-1*)—the remaining eight bits are either discarded or used as parity check bits. The 56 bits are then divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one and two bits (specified for each round), and then 48 subkey bits are selected by *Permuted Choice 2* (*PC-2*)—24 bits from the left half, and 24 from the right. The rotations (denoted by "<<<" in the diagram) mean that a different set of bits is used in each subkey; each bit is used in approximately 14 out of the 16 subkeys.

The key schedule for decryption is similar—the subkeys are in reverse order compared to encryption. Apart from that change, the process is the same as for encryption. The same 28 bits are passed to all rotation boxes.

3.4: Initial Permutation

It is done on every block of input data in the beginning stage of encryption.

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

Answer:

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

Example: Applying the initial permutation to the block of text **M**, given previously, we get
M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010
Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.
Next divide the permuted block **IP** into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.
Example: From **IP**, we get L_0 and R_0
L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
We now proceed through 16 iterations, for $1 <= n <= 16$, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2)**. Then for **n** going from 1 to 16 we calculate
$L_n = R_{n-1}$
$R_n = L_{n-1} + f(R_{n-1},K_n)$
This results in a final block, for $n = 16$, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f .
Example: For $n = 1$, we have
K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010
$R_1 = L_0 + f(R_0,K_1)$
It remains to explain how the function f works. To calculate f, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1} . We'll call the use of this selection table the function **E**. Thus $\mathbf{E}(R_{n-1})$ has a 32 bit input block, and a 48 bit output block.

3.5: Permutation Key PC-1

From 64-bit key only 56 bits are selected. The key is then divided as left half and right half. Bit shifting is done on every part. (Every eight bit can be used for parity control which is excluded from encryption.

Answer:

In general, a 64-bit key is used as input for DES, of which only 56-bits are used. 16 subkeys, with 48-bit each, will then be created from this 56-bits.
The first step is to permute the key using the PC-1 table above. This is, the first bit of our 56-bit permutation key will be the 57th bit of our original key, and so on.

Left half						
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
Right half						
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

Input Key: **00110110 00110100 01100010 01101001 01110100 01001011 01100101 01111001**

Would Become: 00000000 11111100 11011111 10010000 00100101 01010011 10101000 00110000

Next we divide the key in two parts, left $\mathbf{C_0}$ and right $\mathbf{D_0}$.

C_0: 00000000 11111100 11011111 10010000

D_0: 00100101 01010011 10101000 00110000

Iteration Number	**Number of lift shift**
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

To do a left shift we move each bit one place to the left, except for the first bit which goes to the end of the block.

In our example, we would have the following 16 keys

C_0: 00000000 11111100 11011111 10010000 **D_0: 00100101 01010011 10101000 00110000**

Would originate:

C_1: 00000001 11111001 10111111 00100000 **D_1: 01001010 10100111 01010000 01100000**

C_2: 00000011 11110011 01111110 01000000 **D_2: 10010101 01001110 10100000 11000001**

C_3: 00001111 11001101 11111001 00000000 **D_3: 01010101 00111010 10000011 00100001**

C_4: 00111111 00110111 11100100 00000001 **D_4: 01010100 11101010 00001100 10010001**

C_5: 11111100 11011111 10010000 00000001 **D_5: 01010011 10101000 00110010 01010001**

C_6: 11110011 01111110 01000000 00110001 **D_6: 01001110 10100000 11001001 01010000**

C_7: 11001101 11111001 00000000 11110000 **D_7: 00111010 10000011 00100101 01010001**

C_8: 00110111 11100100 00000011 11110001 **D_8: 11101010 00001100 10010101 01000000**

C_9: 01101111 11001000 00000111 11100000 **D_9: 11010100 00011001 00101010 10010001**

C_{10}: 10111111 00100000 00011111 10010001 **D_{10}:01010000 01100100 10101010 01110001**

C_{11}: 11111100 10000000 01111110 01100001 **D_{11}:01000001 10010010 10101001 11010000**

C_{12}: 11110010 00000001 11111001 10110001 **D_{12}:00000110 01001010 10100111 01010000**

C_{13}: 11001000 00000111 11100110 11110000 **D_{13}:00011001 00101010 10011101 01000001**

C_{14}: 00100000 00011111 10011011 11110000 **D_{14}:01100100 10101010 01110101 00000000**

C_{15}: 10000000 01111110 01101111 11000000 **D_{15}:10010010 10101001 11010100 00010001**

C_{16}: 00000000 11111100 11011111 10010000 **D_{16}:00100101 01010011 10101000 00110000**

3.6 : Permutation Expansion

Every round feistel function is initiated by expansion. The right half of data is expanded form 32 to 48 bits.

Answer:

The Permutation Expansion is used to expand the 32-bit input to a round's F function into a 48-bit block. The E function is fairly straightforward and is implemented as shown in the Table below.

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

As shown in the Table, the first two bits of each eight-bit block are the same as the last two bits of the previous block (wrapping around to the last block in the case of the first row). The remaining 4 columns are the bits of the input in order starting with the second bit.

As an example, let's expand the 32-bit string: 11010100 00000100 11010100 11110011. For the purposes of this example, we'll break the steps into two parts: the left two columns repeated from the previous round and the right four columns which are unique. The symbol ':' is used to represent a range that is inclusive and wraps around, i.e. 30:2 is (30, 31, 1, 2).

In: 11010100 00000100 11010100 11110011

Left Right

P[32: 1]=11 P[2: 5]=1010

P[4: 5]=10 P[6: 9]=1000

P[8: 9]=00 P[10:13]=0000

P[12:13]=00 P[14:17]=1001

P[16:17]=01 P[18:21]=1010

P[20:21]=10 P[22:25]=1001

P[24:25]=01 P[26:29]=1110

P[28:29]=10 P[30: 1]=0111

Out: 111010 101000 000000 001001 011010 101001 011110 100111

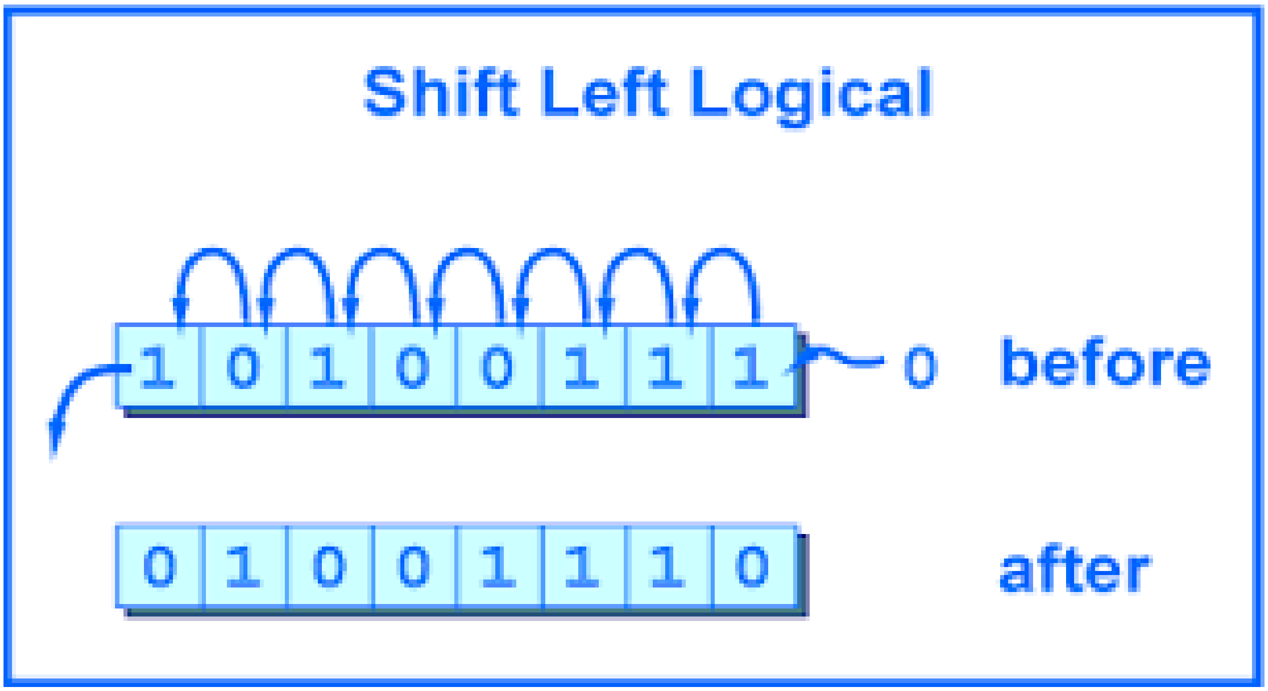
Binary Shifting

The 48 bits are rotated left by one or 2 bits.

No. of cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Amount of bits	1	1	2	2	2	2	2	2	1	2	2	2	2	2	1	16

Binary shifting is just as its name suggests; we are shifting or moving binary values left or right. Each 1 or 0 is called a bit; which is short for Binary digIT.

Arithmetic shifts can be useful as efficient ways to perform multiplication or division of signed integers by powers of two. Shifting left by *n* bits on a signed or unsigned binary number has the effect of multiplying it by 2^n. Shifting right by *n* bits on a two's complement *signed* binary number has the effect of dividing it by 2^n, but it always rounds down (towards negative infinity). This is different from the way rounding is usually done in signed integer division (which rounds towards 0). This discrepancy has led to bugs in more than one compiler.



Permutation Key PC-2

From the 56 bit subkey which is output of a given round of feistel function, only 48 bit subkey are selected.

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

Key C_1D_1: 00000000 11111100 11011111 10010000 00100101 01010011 10101000 00110000 will become:

K1= 00000011 00101110 00100011 00110000

The other keys are:

K_2= 00011100 00111001 00000101 00101000

K_3= 00001011 00110100 00000100 00010111

K_4= 00001000 00111011 00100010 00000011

K_5= 00000111 00110110 00111100 00000101

K_6= 00110101 00010001 00110001 00001010

K_7= 00101100 00110000 00100000 00001001

K_8= 00111001 00011101 00100100 00011111

K_9= 00101101 00100111 00101001 00000101

K_{10}= 00110011 00001100 00100110 00001100

K_{11}= 00111001 00011011 00100110 00100000

K_{12}= 00101010 00001101 00010100 00011110

K_{13}= 00010101 00101011 00000010 00001001

K_{14}= 00001011 00001011 00011110 00100100

K_{15}= 00110101 00011000 00100111 00011010

K_{16}= 00101100 00100111 00111000 00000000

S-Blocks

- 48 bit input is divided into 6 bit input of 8 blocks.
- From each 6 bit the first and the last bit is taken as a row value and the remaining 4 bits are taken as a column value. • The resultant S- box value is a 4 bit output. For eg, for a 6

bit input 001110 the row value is 0(00) and the column value is 7(0111) and the resultant S1 box is 8 whose 4 bit output is 1000.

Answer:

The first and last bits of ***B*** represent in base 2 a number in the decimal range 0 to 3 (binary 00 to 11). Let that number be ***i***.
The 4 bits in the middle of ***B*** represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be ***j***.
Look up in the table the number in the ***i***-th row and ***j***-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input ***B***. For example, for input block ***B*** = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence S_1(011011) = 0101.

The tables defining the functions $S_1,...,S_8$ are the following:

S_1

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

S_2

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

S_3

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

S_4

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

S_5

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

S_6

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

S_7

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

S_8

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

In our example we obtain as the output of the eight S boxes:

$\boldsymbol{K_1 \oplus E(R_0)}$ = 100000 110010 111111 011101 101100 000010 100000 001010

$\mathbf{S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)=}$

0100 1000 1100 1110 0111 0001 0001 1111

Permutation P

The output block of 32 bit from S -box undergo P-Permutation

Answer:

The Permutation p in DES is another permutation function. It takes a thrity-two bit block as input and outputs a thirty-two bit block. The permutation is shown in the Table below.

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

As shown, the permutation for the P function is not as structured as other permutation functions in DES. However, the permutation is not random and is the same for all rounds of DES.

As an example, we'll use the S-Box output from our example in the previous section: 10101101 11101110 10010100 01110010.

In: 10101101 11101110 10010100 01110010

P[16]=0, P[7]=0, P[20]=1, P[21]=0, P[29]=0, P[12]=0, P[28]=1, P[17]=1

P[1]=1, P[15]=1, P[23]=0, P[26]=1, P[5]=1, P[18]=0, P[31]=1, P[10]=1

P[2]=0, P[8]=1, P[24]=0, P[14]=1, P[32]=0, P[27]=1, P[3]=1, P[9]=1

P[19]=0, P[13]=1, P[30]=0, P[6]=1, P[22]=1, P[11]=1, P[4]=0, P[25]=0

Out: 00100011 11011011 01010111 01011100

Final Permutation

This is done for every block of data which is the inverse of IP.

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

The final permutation occurs after the sixteen rounds of DES are completed. It is the inverse of the initial permutation and is shown in the Table below.

As an example, let's undo the initial permutation. Its output was 01001100 00110011 01010110 10011110 00111010 00100110 10100000 10001110.

The Final Permutation is applied as follows:

In: 01001100 00110011 01010110 10011110 00111010 00100110 10100000 10001110

P[40]=0, P[8]=0, P[48]=0, P[16]=1, P[56]=0, P[24]=0, P[64]=0, P[32]=0

P[39]=1, P[7]=0, P[47]=1, P[15]=1, P[55]=0, P[23]=1, P[63]=1, P[31]=1

P[38]=0, P[6]=1, P[46]=1, P[14]=0, P[54]=0, P[22]=1, P[62]=1, P[30]=1

P[37]=1, P[5]=1, P[45]=0, P[13]=0, P[53]=0, P[21]=0, P[61]=1, P[29]=1

P[36]=1, P[4]=0, P[44]=0, P[12]=1, P[52]=0, P[20]=1, P[60]=0, P[28]=1

P[35]=1, P[3]=0, P[43]=1, P[11]=1, P[51]=1, P[19]=0, P[59]=0, P[27]=0

P[34]=0, P[2]=1, P[42]=0, P[10]=0, P[50]=0, P[18]=1, P[58]=0, P[26]=0

P[33]=0, P[1]=0, P[41]=0, P[9]=0, P[49]=1, P[17]=0, P[57]=1, P[25]=1

Out: 00010000 10110111 01100111 11000011 10010101 10111000 01000100 00001011

Conclusion

In this paper we have explained mathematical technique for DES algorithm and have also given examples for the same. The essential concern in DES algorithm protection is about two areas such as nature of algorithm and key size. It is clear that DES can be broken using 255 encryptions. However, nowadays most purposes use either 3DES with two keys or 3DES with three keys. These two a couple of DES versions make DES resistant to brute- force attacks.

Using own sentence convert to cipher text using DES

Encryption:

Plain Text:

Iqra National University Hayatabad

Key: **School64**

Cipher Text:

2a 57 87 53 fe f3 7f 43 1d 68 e9 43 86 c4 01 44
c8 0f 89 e0 d4 42 36 f1 1e 03 3e 18 52 d1 ff 57
b9 69 10 24 fa 55 ec af

Decryption:

Cipher Text:

2a 57 87 53 fe f3 7f 43 1d 68 e9 43 86 c4 01 44
c8 0f 89 e0 d4 42 36 f1 1e 03 3e 18 52 d1 ff 57
b9 69 10 24 fa 55 ec af

Key: **School64**

Plan Text:

Iqra National University Hayatabad