Name: Sagar Patil

Roll no: 8061

Assignment No.1

Title: Implementation of S-DES (Data Encryption Standard)

Problem Definition:

Implementation of S-DES

Prerequisite:

Basics of Computer networking and Python

Software Requirements:

Python 3

Hardware Requirements:

PIV, 2GB RAM, 500 GB HDD

Learning Objectives:

Learn Data Encryption Standard Algorithm (DES)

Outcomes:

After completion of this assignment students are able to understand the Data Encryption Standard.

Theory Concepts:

Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a Symmetric-key block cipher issued by the national Institute of Standards & Technology (NIST).

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

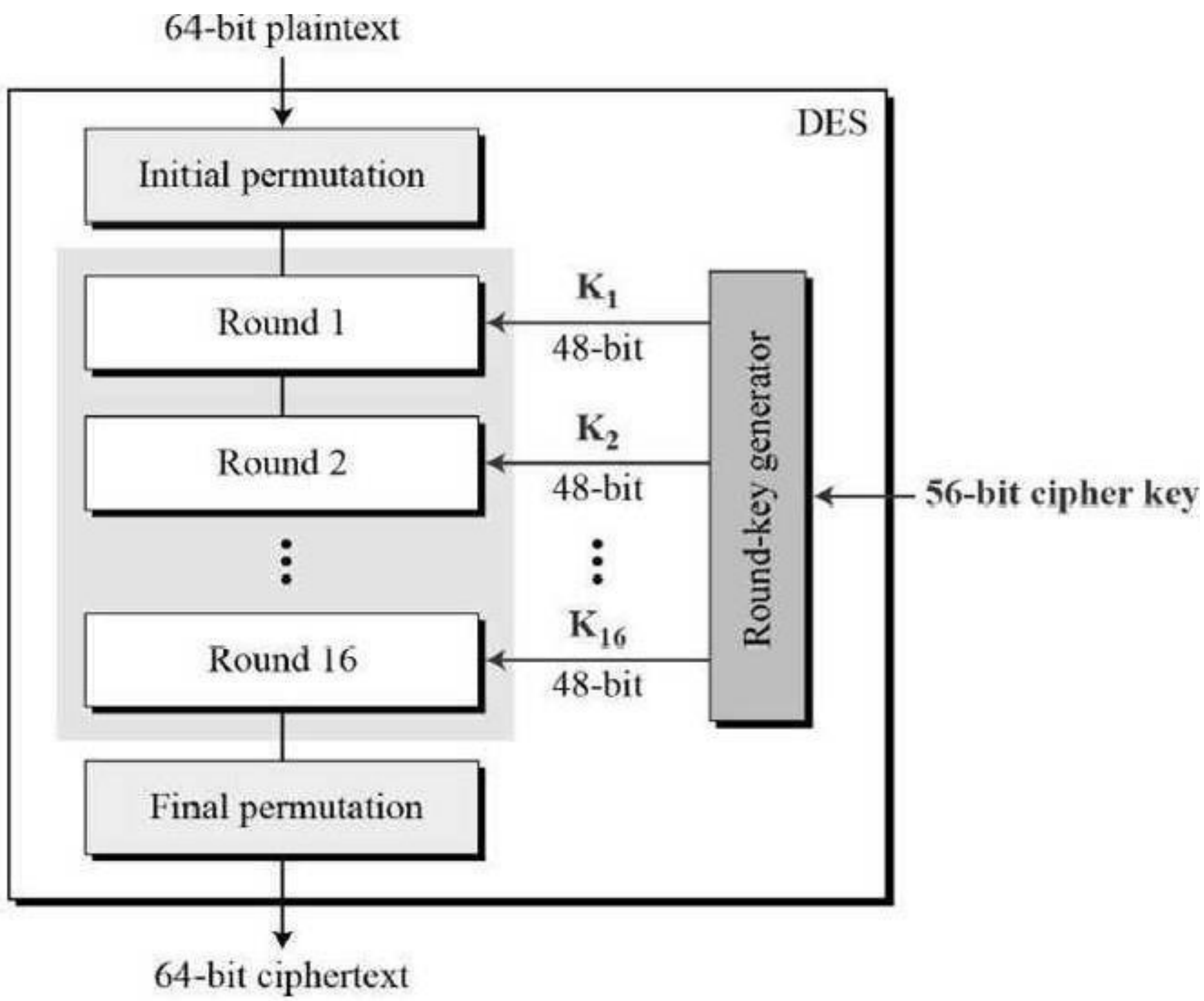


Figure 5.1: General Structure of DES

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

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

Initial and Final Permutation

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

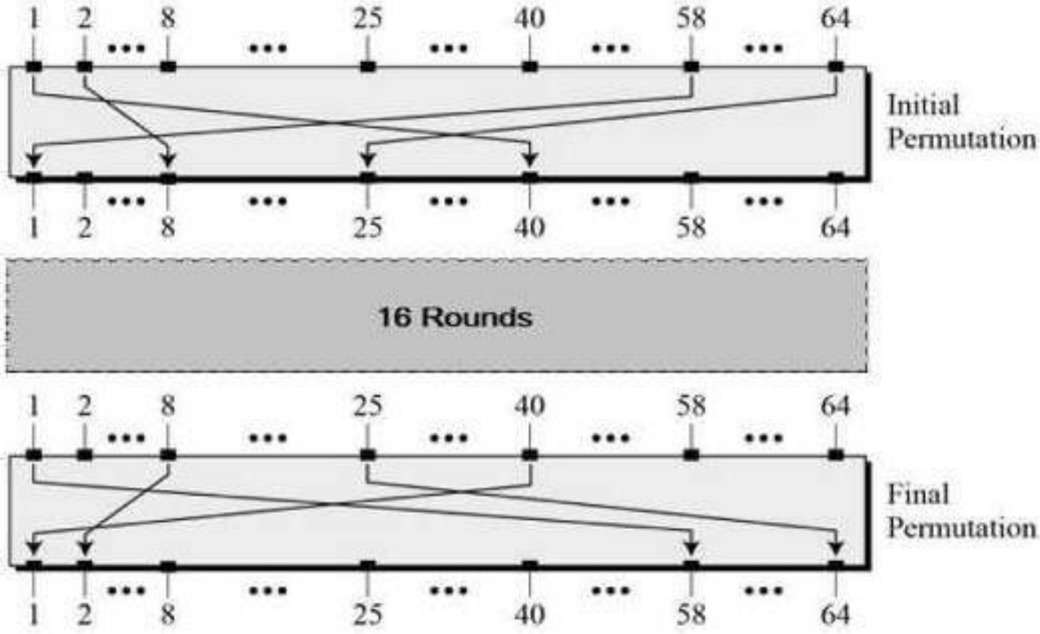


Figure : initial and final permutations

Round Function

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

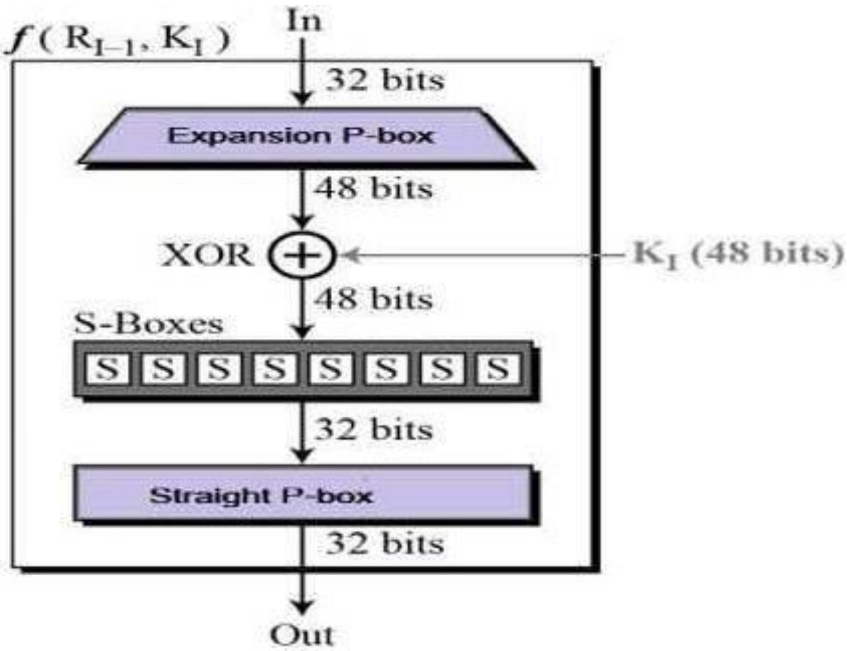
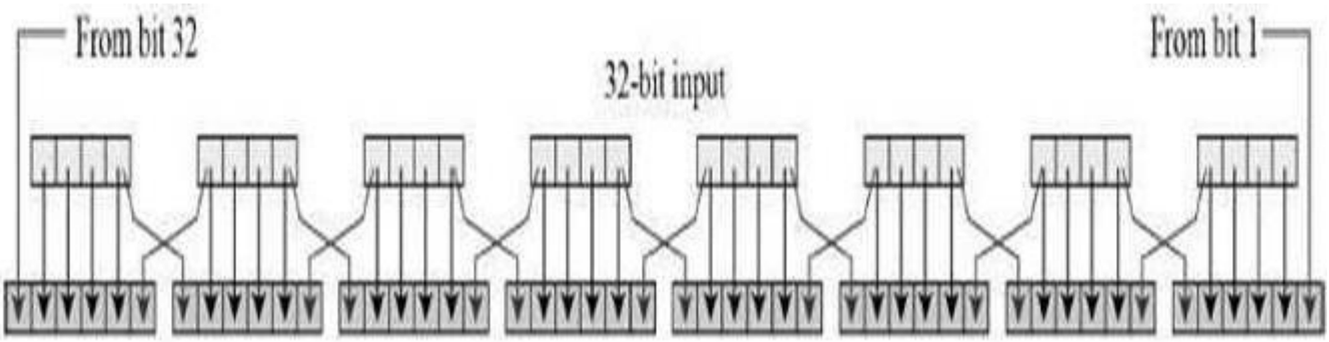


Figure : Round Functions

Expansion Permutation Box

Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits.



Permutation logic is graphically depicted in the following illustration:

Figure : Permutation logic

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

Table 5.1 Permutation logic

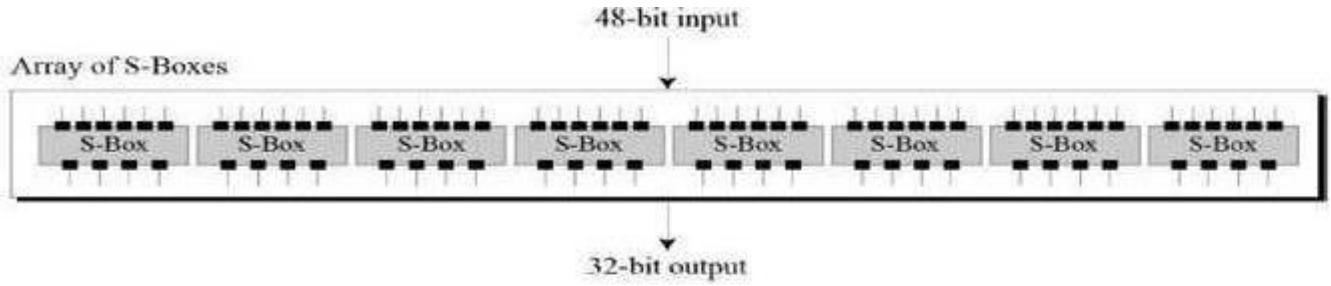
32	01	02	03	04	05
04	05	06	07	08	09
08	09	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	31	31	32	01

XOR(Whitener)

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

Substitution Boxes

The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and



a 4-bit output. Refer the following illustration –

Figure : S-Boxes

The S-box rule is illustrated below –

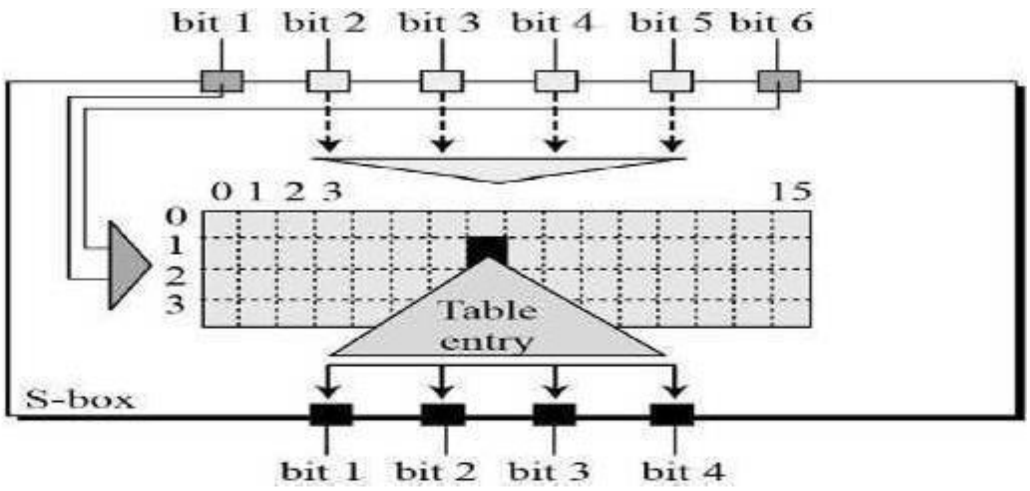


Figure 5.6 S-Box Rules

There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.

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

Table 5.2 Straight Permutation

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

Key Generation

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

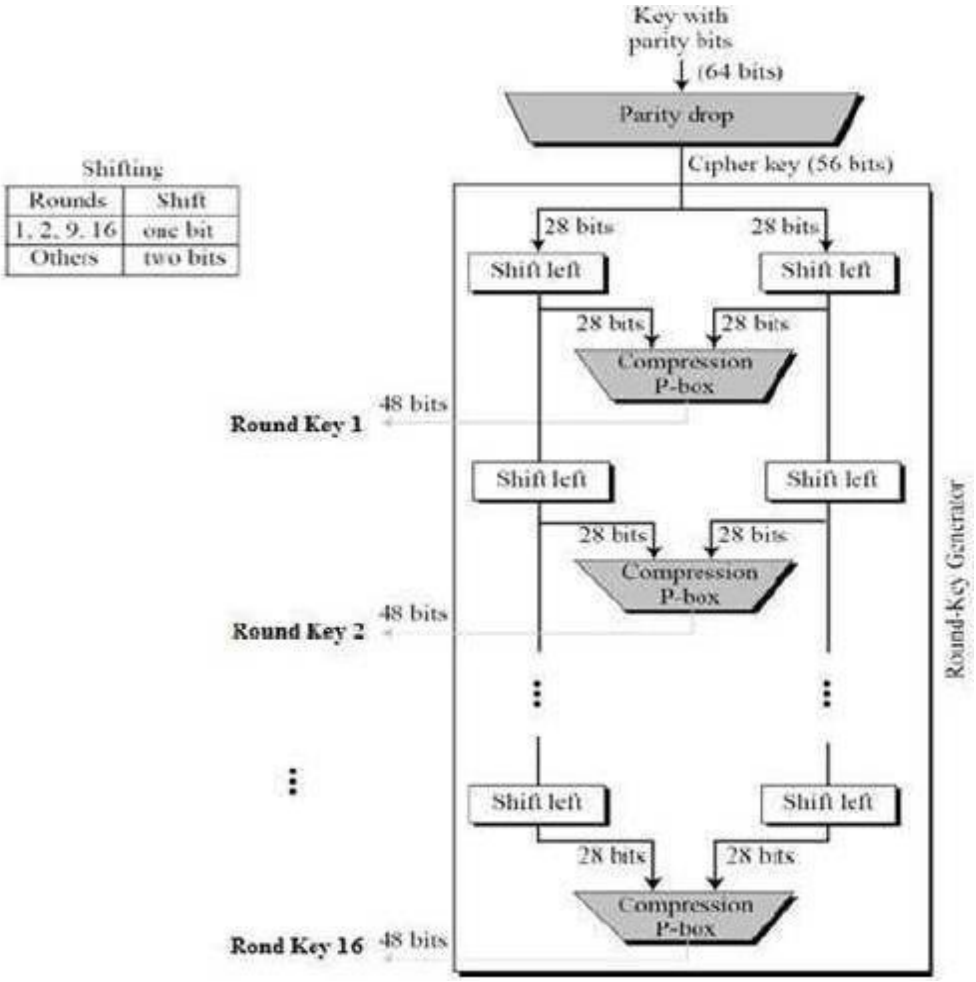


Figure : the process of key generation

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

DES Analysis

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

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

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

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

Conclusion: Successfully learned and implemented algorithm of S-DES.

Code:

```
P10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)
P8 = (6, 3, 7, 4, 8, 5, 10, 9)
P4 = (2, 4, 3, 1)

IP = (2, 6, 3, 1, 4, 8, 5, 7)
IPi = (4, 1, 3, 5, 7, 2, 8, 6)

E = (4, 1, 2, 3, 2, 3, 4, 1)

S0 = [
        [1, 0, 3, 2],
        [3, 2, 1, 0],
        [0, 2, 1, 3],
        [3, 1, 3, 2]
     ]

S1 = [
        [0, 1, 2, 3],
        [2, 0, 1, 3],
        [3, 0, 1, 0],
        [2, 1, 0, 3]
     ]

def permutation(pattern, key):
    permuted = ""

    for i in pattern:
        permuted += key[i-1]

    return permuted

def generate_first(left, right):
    left = left[1:] + left[:1]
    right = right[1:] + right[:1]
    key = left + right

    return permutation(P8, key)

def generate_second(left, right):
    left = left[3:] + left[:3]
    right = right[3:] + right[:3]
    key = left + right

    return permutation(P8, key)

def transform(right, key):
    extended = permutation(E, right)
    xor_cipher = bin(int(extended, 2) ^ int(key, 2))[2:].zfill(8)
```

```
    xor_left = xor_cipher[:4]
    xor_right = xor_cipher[4:]

    new_left = Sbox(xor_left, S0)
    new_right = Sbox(xor_right, S1)

    return permutation(P4, new_left + new_right)

def Sbox(data, box):
    row = int(data[0] + data[3], 2)
    column = int(data[1] + data[2], 2)

    return bin(box[row][column])[2:].zfill(4)

def encrypt(left, right, key):
    cipher = int(left, 2) ^ int(transform(right, key), 2)

    return right, bin(cipher)[2:].zfill(4)

key = input("Enter a 10-bit key: ")
if len(key) != 10:
    raise Exception("Check the input")

plaintext = input("Enter 8-bit plaintext: ")
if len(plaintext) != 8:
    raise Exception("Check the input")

p10key = permutation(P10, key)
print("First Permutation")
print(p10key)
left_key = p10key[:len(p10key)//2]
print("Left key",left_key)
right_key = p10key[len(p10key)//2:]
print("Right key",right_key)
first_key = generate_first(left_key, right_key)
print("*****")
print("First key")
print(first_key)
second_key = generate_second(left_key, right_key)
print("*****")
print("Second key")
print(second_key)
initial_permutation = permutation(IP, plaintext)
print("Initial Permutation",initial_permutation)
left_data = initial_permutation[:len(initial_permutation)//2]
right_data = initial_permutation[len(initial_permutation)//2:]

left, right = encrypt(left_data, right_data, first_key)
left, right = encrypt(left, right, second_key)

print("Ciphertext:", permutation(IPi, left + right))
```

Output:

```
Enter a 10-bit key: 1010101010
Enter 8-bit plaintext: 10101010
First Permutation
1101001100
Left key 11010
Right key 01100
*****
```

First key
11100100
*****
Second key
01010011
Initial Permutation 00110011
Ciphertext: 10101011