Experiment-5

Aim → Implementation of block ciphering process for secured transmission of digital information in computer network.

Objective 1 → An overview on DES (Data Encryption Standard) algorithm.

DES is a symmetric-key block cipher developed by IBM and standardized by NIST in 1997. It was widely used for securing digital information, but it is now considered outdated due to its short key length.

Key features →

Block size = 64 bits

Key size = 64 bits (only 56 bits used, 8 bits for parity)

Rounds = 16

Algorithm Type = Feistel structure

How DES works →

i) Initial Permutation (IP) → Rearranges the 64-bit plaintext according to a fixed table.

ii) 16 rounds of processing → Each round involves: dividing data into left and right halves, expanding the right half (32 → 48 bits), XOR with a 48-bit subkey, substition using S-boxes (48 → 32 bits), permutation, swapping the halves.

iii) Final Permutation (FP) → Applies the inverse of the initial permutation to get the ciphertext.

Key Generation → The original 64-bit key is reduced to 56 bits. Subkeys are generated for each round using shifts & a permutation table.

Decryption → Uses the same process as encryption but applies the subkeys in reverse order.

Limitations → The 56-bit key is vulnerable to brute-force attacks. Not recommended for modern use - replaced by more secure algorithms like AES.

Name: Reed No:

Objective-2 → Execution of DES algorithm for encryption & decryption of digital information.

```
# Initial Permutation Table
IP = [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]

# Final Permutation Table
FP = [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]

# Expansion Table
E = [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]

# S-boxes
S_BOXES = [
[[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]],

[[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]],

[[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]],

[[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]],

[[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]],

[[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]],
```

Name: Regd No:

```
[[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]],

[[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]]
]


# Permutation Function P
P = [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]

# PC-1 for key permutation
PC1 = [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]

# PC-2 for key compression
PC2 = [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]

# Number of left shifts
SHIFT = [1, 1, 2, 2, 2, 2, 2, 2,
1, 2, 2, 2, 2, 2, 2, 1]

# Helper functions
def permute(block, table):
return [block[i-1] for i in table]

def shift_left(k, n):
return k[n:] + k[:n]

def xor(a, b):
return [i ^ j for i, j in zip(a, b)]

def sbox_substitution(block48):
output = []
for i in range(8):
chunk = block48[i*6:(i+1)*6]
row = (chunk[0] << 1) | chunk[5]
col = (chunk[1] << 3) | (chunk[2] << 2) | (chunk[3] << 1) | chunk[4]
val = S_BOXES[i][row][col]
bin_val = [int(x) for x in format(val, '04b')]
```

```
output.extend(bin_val)
return output

def generate_keys(key64):
key56 = permute(key64, PC1)
C = key56[:28]
D = key56[28:]
keys = []
for i in range(16):
C = shift_left(C, SHIFT[i])
D = shift_left(D, SHIFT[i])
combined = C + D
round_key = permute(combined, PC2)
keys.append(round_key)
return keys

def des_round(L, R, key):
expanded_R = permute(R, E)
temp = xor(expanded_R, key)
sbox_out = sbox_substitution(temp)
permuted = permute(sbox_out, P)
result = xor(L, permuted)
return R, result

def des_encrypt(block64, keys):
block = permute(block64, IP)
L, R = block[:32], block[32:]
for i in range(16):
L, R = des_round(L, R, keys[i])
final_block = R + L # Note the swap
return permute(final_block, FP)

def des_decrypt(block64, keys):
block = permute(block64, IP)
L, R = block[:32], block[32:]
for i in range(15, -1, -1):
L, R = des_round(L, R, keys[i])
final_block = R + L # Note the swap
return permute(final_block, FP)

# Convert string to 64-bit binary
def string_to_bitlist(s):
return [int(bit) for char in s for bit in format(ord(char), '08b')]

# Convert 64-bit binary to string
def bitlist_to_string(b):
return ''.join(chr(int(''.join(map(str, b[i:i+8])), 2)) for i in range(0,
len(b), 8))

# Example usage
plaintext = "ABCDEFGH" # 8 characters = 64 bits
keytext = "12345678" # 8 characters = 64 bits

plain_bits = string_to_bitlist(plaintext)
key_bits = string_to_bitlist(keytext)

subkeys = generate_keys(key_bits)
cipher_bits = des_encrypt(plain_bits, subkeys)
decrypted_bits = des_decrypt(cipher_bits, subkeys)

print("Original:", plaintext)
print("Encrypted bits:", cipher_bits)
print("Decrypted:", bitlist_to_string(decrypted_bits))
Output: -
```

```
Original: ABCDEFGH
Encrypted bits: [1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1]
Decrypted: ABCDEFGH
```

Conclusion → In this experiment, we understood the internal working of DES, including key generation, the Feistel Structure, S-box transformations, and round-based encryption. While DES is no longer secure for modern applications due to its short key length, it remains an essential foundation for understanding the principles of block cipher design and cryptographic security.

Exercises →

Using DES find the following for the given 8-bit plaintext 10010111 and 10-bit key 1010000010

1) Find the permuted key using the P10 table given as 3 5 2 7 4 10 1 9 8 6 and the round 1 key (k1) using the P8 table 6 3 7 4 8 5 10 9.

Ans→ 10 bit key → 1010000010

P10 Table → [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

key 1 0 1 0 0 0 0 0 1 0

P10 output 1 0 0 0 0 0 1 1 0 0 → Permuted key

Left = 10000 Right = 01100

After LS-1, Left shift (10000) = 00001 Left shift (01100) = 11000

Combined after shift = 0000111000

Applying P8 table →

~~P8~~ index → ~~1 3 7 4 8 5 10 9~~

1 2 3 4 5 6 7 8 9 10

key → 0 0 0 0 1 1 1 0 0 0

P8 combined → 1 0 1 0 0 1 0 0 → Round 1 key (k1)

2) Find the output of initial permutation as L0 and R0 using the given IP table 2 6 3 1 4 8 5 7.

Ans→ 8 bit plaintext → 10010111 IP table → 2 6 3 1 4 8 5 7

IP output → 01011101

L0 = 0101 , R0 = 1101

3) Find the output of expansion permutation on RO.

Ans→ RO → 1101

EP→ 4 1 2 3 2 3 4 1

Output → 1 1 1 0 1 0 1 1 EP(RO) → 11101011

4) Find the output of XOR (EP(RO), K1).

Ans→ EP(RO) output → 11101011

K1 (key) → 10100100

XOR (EP(RO), K1) → 01001111

5) Find the out put of the given S-boxes.

Ans→ XOR result → 01001111

Left half → 0100, Right half → 1111

$S_0 =$

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

$S_1 =$

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

So, lookup → row 00 → 0, column = 10 which gives 11.

S_1 lookup → row = 11, colum = 11 which gives 11

Combined S_0 & S_1 → 1111

Applying P_4 permutation, final output → 1111

6) Find the output of the permutation table using the P4 table 2 4 3 1.

Ans→ XOR output → 0100 1111

Rightmost 4 gives → 1111

P_4 permutation → 2 4 3 1

So, the P_4 output becomes 1111.