Rashtriya Raksha University

School of Information Technology, Artificial Intelligence & Cyber Security (SITAICS)

At- Lavad, Dahegam, Gandhinagar, Gujarat-382305

Practical File

(Introduction to Cryptography)

Name:	Sarthak Sanay
Enrollment No:	230031101611051
Subject Name:	Introduction to Cryptography
Subject Code:	G4A19ITC
Program:	B.Tech CSE (with specialization in Cyber Security)
Year:	2nd year (Semester-IV)

This is certifying that Mr. Sarthak Sanay has satisfactorily completed all experiments in the practical work prescribed by SITAICS in the ITC laboratory.

Dr. Ashish Revar
SUBJECT INCHARGE

PRACTICAL - 7

AIM: TO IMPLEMENT THE DES (DATA ENCRYPTION STANDARD) ALGORITHM

BRIEF :-

The Data Encryption Standard (DES) is a symmetric-key block cipher developed by IBM in the early 1970s and later adopted by the U.S. National Institute of Standards and Technology (NIST) as a federal encryption standard in 1977. It operates on 64-bit blocks of data using a 56-bit key (excluding 8 parity bits). DES works by dividing the plaintext into two halves and then processing them through 16 rounds of complex operations, including substitution, permutation, and bitwise logical operations based on the key. Its core structure follows the Feistel cipher model, which ensures that encryption and decryption processes are similar, enhancing efficiency.

Despite its historical importance, DES is now considered insecure due to advances in computing power. Its relatively short key length makes it vulnerable to brute-force attacks, with real-world examples successfully cracking DES-encrypted messages in under a day. As a result, DES has been largely replaced by stronger encryption standards such as Triple DES (3DES) and the Advanced Encryption Standard (AES). However, DES remains a foundational algorithm in the field of cryptography, widely studied for educational purposes and as a basis for understanding modern encryption systems.

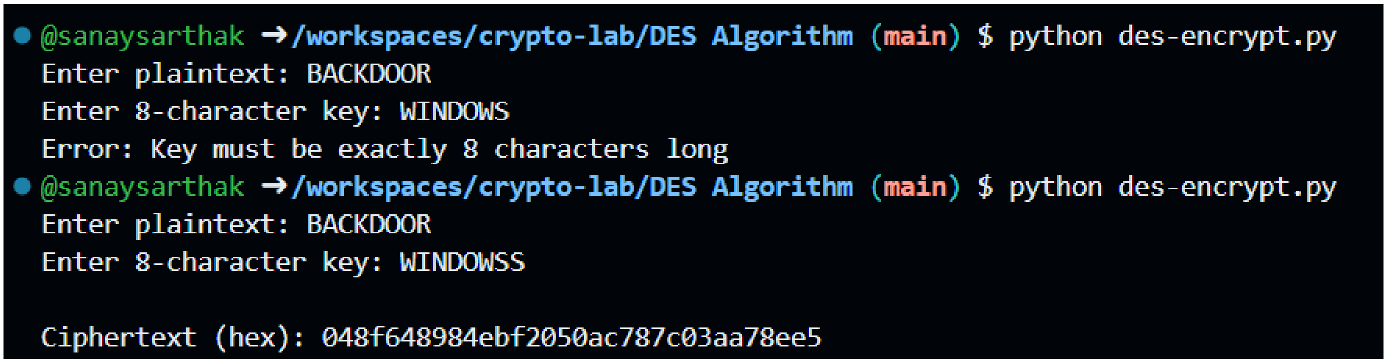
ALGORITHM / PSEUDOCODE FOR ENCRYPTION :-

```
print menu
read plaintext
read key

if key length ≠ 8 then
    error “Key must be 8 characters”
    exit

// Pad plaintext to a multiple of 8 bytes (PKCS5 style)
pad_len = 8 − (length(plaintext) mod 8)
plaintext += chr(pad_len) × pad_len

// Generate the 16 round keys
round_keys = []
key_bits = permute(string_to_bits(key), PC1)
L, R = split(key_bits, 28)
for each shift in shift_schedule do
    L = left_shift(L, shift)
    R = left_shift(R, shift)
    round_keys.append( permute(L + R, PC2) )
end for

ciphertext = “”

// Process each 8-byte block
for each block in split(plaintext, 8 bytes) do
    bits = string_to_bits(block)
    bits = permute(bits, IP)

    L = bits[0..31]
    R = bits[32..63]

    // 16 Feistel rounds
    for i = 1 to 16 do
        E = permute(R, E_BOX)
        X = xor(E, round_keys[i])
        S = apply_sboxes(X)
        P = permute(S, P_BOX)
        L, R = R, xor(L, P)
    end for

    // Swap and final permutation
    preout = R + L
    cipher_bits = permute(preout, IP_INVERSE)
    ciphertext += bits_to_string(cipher_bits)
end for

hex_output = bytes_to_hex(ciphertext)
print hex_output
```

CODE FOR ENCRYPTION :-

```
# Implementation of DES (Data Encryption Standard) Algorithm in Python

# DES Tables
ip_table = [
    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
]

pc1_table = [
    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
]

pc2_table = [
    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
]

e_box_table = [
    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
]

p_box_table = [
    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
]

ip_inverse_table = [
    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
]

shift_schedule = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

s_boxes = [
    # S-box 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-box 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-box 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-box 4
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 51],
        [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-box 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-box 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-box 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-box 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]
    ]
]

def string_to_bin(text):
    return ''.join(format(ord(char), '08b') for char in text)

def bin_to_string(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0,
len(binary), 8))

def permute(input_block, table):
    return ''.join(input_block[i-1] for i in table)

def left_shift(data, shifts):
    return data[shifts:] + data[:shifts]

def xor(a, b):
    return ''.join('1' if x != y else '0' for x, y in zip(a, b))

def apply_sbox(expanded_block):
    output = ""
    for i in range(8):
        block = expanded_block[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        output += format(s_boxes[i][row][col], '04b')
    return output

def generate_round_keys(key):
    key = string_to_bin(key)

    key = permute(key, pc1_table)
    left = key[:28]
    right = key[28:]

    round_keys = []

    for i in range(16):
        left = left_shift(left, shift_schedule[i])
        right = left_shift(right, shift_schedule[i])
        combined = left + right
        round_key = permute(combined, pc2_table)
        round_keys.append(round_key)

    return round_keys

def des_round(left, right, round_key):
    expanded = permute(right, e_box_table)
    xored = xor(expanded, round_key)
    substituted = apply_sbox(xored)
    permuted = permute(substituted, p_box_table)
    new_right = xor(left, permuted)
    return right, new_right

def pad_text(text):
    pad_length = 8 - (len(text) % 8)
    return text + chr(pad_length) * pad_length

def des_encrypt(plaintext, key):
    if len(key) != 8:
        raise ValueError("Key must be exactly 8 characters long")
    plaintext = pad_text(plaintext)
    round_keys = generate_round_keys(key)

    ciphertext = ""

    for i in range(0, len(plaintext), 8):
        block = plaintext[i:i+8]
        block_bin = string_to_bin(block)
        block_bin = permute(block_bin, ip_table)
        left = block_bin[:32]
        right = block_bin[32:]
        for j in range(16):
            left, right = des_round(left, right, round_keys[j])
        left, right = right, left
        combined = left + right
        encrypted_block = permute(combined, ip_inverse_table)
        ciphertext += bin_to_string(encrypted_block)
    return ciphertext

if __name__ == "__main__":
    plaintext = input("Enter plaintext: ")
    key = input("Enter 8-character key: ")
    try:
        ciphertext = des_encrypt(plaintext, key)
        hex_ciphertext = ''.join(format(ord(c), '02x') for c in
ciphertext)
        print(f"\nCiphertext (hex): {hex_ciphertext}")

    except ValueError as e:
        print(f"Error: {e}")
```

OUTPUT FOR ENCRYPTION :-



```
● @sanaysarthak →/workspaces/crypto-lab/DES Algorithm (main) $ python des-encrypt.py
  Enter plaintext: BACKDOOR
  Enter 8-character key: WINDOWS
  Error: Key must be exactly 8 characters long
● @sanaysarthak →/workspaces/crypto-lab/DES Algorithm (main) $ python des-encrypt.py
  Enter plaintext: BACKDOOR
  Enter 8-character key: WINDOWSS

  Ciphertext (hex): 048f648984ebf2050ac787c03aa78ee5
```

ALGORITHM / PSEUDOCODE FOR DECRYPTION :-

```
BEGIN

  DISPLAY menu
  INPUT ciphertext (in hex)
  INPUT key (must be 8 characters)

  IF LENGTH(key) ≠ 8 THEN
    DISPLAY "Key must be 8 characters long"
    EXIT
  ENDIF

  raw ← hex_to_string(ciphertext)
  key_bits ← permute(string_to_bits(key), PC1)
  L, R ← split(key_bits, 28)

  round_keys ← EMPTY LIST
  FOR each shift IN shift_schedule DO
    L ← left_shift(L, shift)
    R ← left_shift(R, shift)
    round_key ← permute(L + R, PC2)
    APPEND round_key TO round_keys
  ENDFOR

  plaintext ← ""

  FOR each block IN split(raw, 8 bytes) DO
    bits ← string_to_bits(block)

    bits ← permute(bits, IP)
    L ← bits[0..31]
    R ← bits[32..63]

    FOR i FROM 15 DOWNTO 0 DO
      L, R ← des_round(L, R, round_keys[i])
    ENDFOR

    pre_output ← R + L
    decrypted_bits ← permute(pre_output, IP_INVERSE)
    plaintext ← plaintext + bits_to_string(decrypted_bits)
  ENDFOR

  plaintext ← unpad_text(plaintext)

  DISPLAY "Decrypted Plaintext: ", plaintext

END
```

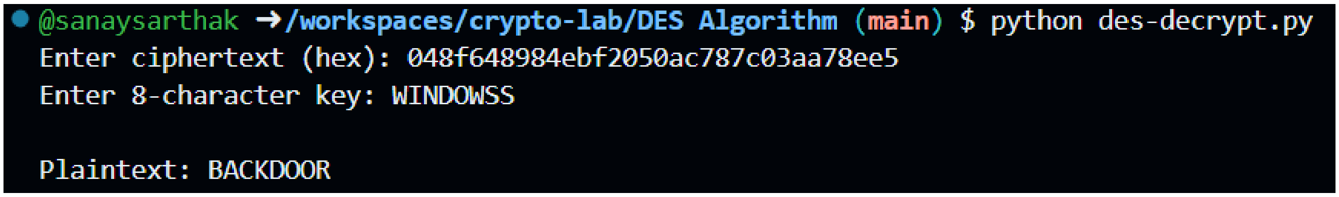
CODE FOR DECRYPTION :-

```
# Implementation of DES (Data Encryption Standard) Algorithm in Python

# DES Tables
ip_table = [
    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
]

pc1_table = [
    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
]

pc2_table = [
    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
]

e_box_table = [
    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
]

p_box_table = [
    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
]

ip_inverse_table = [
    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
]

shift_schedule = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

s_boxes = [
    # S-box 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-box 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-box 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-box 4
    [
        [7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 51],
        [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-box 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-box 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-box 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-box 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]
    ]
]

def string_to_bin(text):
    return ''.join(format(ord(char), '08b') for char in text)

def bin_to_string(binary):
    return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0,
len(binary), 8))

def hex_to_string(hex_text):
    return ''.join(chr(int(hex_text[i:i+2], 16)) for i in range(0,
len(hex_text), 2))

def permute(input_block, table):
    return ''.join(input_block[i-1] for i in table)

def left_shift(data, shifts):
    return data[shifts:] + data[:shifts]

def xor(a, b):
    return ''.join('1' if x != y else '0' for x, y in zip(a, b))

def apply_sbox(expanded_block):
    output = ""
    for i in range(8):
        block = expanded_block[i*6:(i+1)*6]
        row = int(block[0] + block[5], 2)
        col = int(block[1:5], 2)
        output += format(s_boxes[i][row][col], '04b')
    return output
```

```
def generate_round_keys(key):
    key = string_to_bin(key)
    key = permute(key, pc1_table)
    left = key[:28]
    right = key[28:]
    round_keys = []
    for i in range(16):
        left = left_shift(left, shift_schedule[i])
        right = left_shift(right, shift_schedule[i])
        combined = left + right
        round_key = permute(combined, pc2_table)
        round_keys.append(round_key)
    return round_keys

def des_round(left, right, round_key):
    expanded = permute(right, e_box_table)
    xored = xor(expanded, round_key)
    substituted = apply_sbox(xored)
    permuted = permute(substituted, p_box_table)
    new_right = xor(left, permuted)
    return right, new_right

def unpad_text(text):
    pad_value = ord(text[-1])
    if pad_value > 0 and pad_value <= 8:
        for i in range(1, pad_value + 1):
            if ord(text[-i]) != pad_value:
                return text
        return text[:-pad_value]
    return text

def des_decrypt(hex_ciphertext, key):
    if len(key) != 8:
        raise ValueError("Key must be exactly 8 characters long")
    ciphertext = hex_to_string(hex_ciphertext)
    round_keys = generate_round_keys(key)
    plaintext = ""
    for i in range(0, len(ciphertext), 8):
        block = ciphertext[i:i+8]
        block_bin = string_to_bin(block)
        block_bin = permute(block_bin, ip_table)
        left = block_bin[:32]
        right = block_bin[32:]
        for j in range(15, -1, -1):
            left, right = des_round(left, right, round_keys[j])
        left, right = right, left
        combined = left + right
        decrypted_block = permute(combined, ip_inverse_table)
        plaintext += bin_to_string(decrypted_block)
    return unpad_text(plaintext)

if __name__ == "__main__":
```

```
hex_ciphertext = input("Enter ciphertext (hex): ")
key = input("Enter 8-character key: ")
try:
    hex_ciphertext = hex_ciphertext.replace(" ", "")
    plaintext = des_decrypt(hex_ciphertext, key)
    print(f"\nPlaintext: {plaintext}")
except ValueError as e:
    print(f"Error: {e}")
```

OUTPUT FOR DECRYPTION :-



```
● @sanaysarthak ➜ /workspaces/crypto-lab/DES Algorithm (main) $ python des-decrypt.py
  Enter ciphertext (hex): 048f648984ebf2050ac787c03aa78ee5
  Enter 8-character key: WINDOWSS

  Plaintext: BACKDOOR
```