**INFORMATION SECURITY LAB**

**BCA-VI SEMESTER**

**LAB SHEET 5**

**Name: Tanmay Dangat**

**PRN NO: 20220801070**

**Objective**

Implementation of DES Algorithm: Data Encryption Standard (DES)

**Data Encryption Standard** is a symmetric block cipher which takes the input of 64-bit plain text along with 64-bit key and process it, to generate the 64-bit ciphertext.

🔹 Key Features:

✔ Uses a 56-bit key (plus 8 parity bits = 64 bits total).

✔ Encrypts 64-bit blocks of data.

✔ Operates in 16 rounds of encryption.

✔ Uses substitution & permutation

**Tasks**

1. Run the **Data Encryption Standard** Python program in your IDE.

2. **Test encryption & decryption** with different messages.

3. Paste your Python code & output below with the date & time of execution

# DES Implementation in Python

# 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 (Inverse of Initial Permutation)

IP\_INV = [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)

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]

# Permutation Table (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]

# Substitution Boxes (S-boxes)

S\_BOX = [

    # S1

    [[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]],

    # S2

    [[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]],

    # S3

    [[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]],

    # S4

    [[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]],

    # S5

    [[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]],

    # S6

    [[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]],

    # S7

    [[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]],

    # S8

    [[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]]

]

# Permuted Choice 1 (PC-1)

PC\_1 = [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]

# Permuted Choice 2 (PC-2)

PC\_2 = [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 for each round

SHIFT\_SCHEDULE = [1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1]

def hex\_to\_bin(hex\_str, num\_bits):

    """Convert hexadecimal string to binary string with leading zeros"""

    return bin(int(hex\_str, 16))[2:].zfill(num\_bits)

def bin\_to\_hex(bin\_str):

    """Convert binary string to hexadecimal string"""

    return hex(int(bin\_str, 2))[2:].upper()

def str\_to\_bin(text):

    """Convert string to binary representation"""

    return ''.join(format(ord(c), '08b') for c in text)

def bin\_to\_str(bin\_str):

    """Convert binary string to text"""

    return ''.join(chr(int(bin\_str[i:i+8], 2)) for i in range(0, len(bin\_str), 8))

def pad\_text(text):

    """Pad text to be a multiple of 64 bits (8 bytes)"""

    pad\_len = 8 - (len(text) % 8)

    return text + chr(pad\_len) \* pad\_len

def unpad\_text(text):

    """Remove padding from text"""

    pad\_len = ord(text[-1])

    return text[:-pad\_len]

def permute(block, table):

    """Permute the input block using the specified table"""

    return ''.join(block[i-1] for i in table)

def left\_shift(bits, n):

    """Left shift the bits by n positions"""

    return bits[n:] + bits[:n]

def generate\_subkeys(key):

    """Generate the 16 subkeys for each round of DES"""

    # Convert key to binary and permute using PC-1

    key\_bin = hex\_to\_bin(key, 64)

    key\_perm = permute(key\_bin, PC\_1)

    # Split into left and right halves

    left = key\_perm[:28]

    right = key\_perm[28:]

    subkeys = []

    for shift in SHIFT\_SCHEDULE:

        # Left shift both halves

        left = left\_shift(left, shift)

        right = left\_shift(right, shift)

        # Combine and permute with PC-2 to get subkey

        combined = left + right

        subkey = permute(combined, PC\_2)

        subkeys.append(subkey)

    return subkeys

def xor(bits1, bits2):

    """Perform XOR operation on two bit strings"""

    return ''.join(str(int(b1) ^ int(b2)) for b1, b2 in zip(bits1, bits2))

def f\_function(right, subkey):

    """The Feistel function used in each round of DES"""

    # Expand right half from 32 to 48 bits

    expanded = permute(right, E)

    # XOR with subkey

    xored = xor(expanded, subkey)

    # Apply S-boxes

    s\_box\_output = ''

    for i in range(8):

        # Get 6-bit chunk

        chunk = xored[i\*6 : (i+1)\*6]

        row = int(chunk[0] + chunk[5], 2)

        col = int(chunk[1:5], 2)

        # Get S-box value (4 bits)

        val = S\_BOX[i][row][col]

        s\_box\_output += format(val, '04b')

    # Permute with P table

    return permute(s\_box\_output, P)

def des\_round(left, right, subkey):

    """Perform one round of DES encryption"""

    new\_left = right

    new\_right = xor(left, f\_function(right, subkey))

    return new\_left, new\_right

def des\_encrypt\_block(block, subkeys):

    """Encrypt a single 64-bit block using DES"""

    # Initial permutation

    block = permute(block, IP)

    # Split into left and right halves

    left = block[:32]

    right = block[32:]

    # Perform 16 rounds

    for i in range(16):

        left, right = des\_round(left, right, subkeys[i])

    # Combine and perform final permutation

    combined = right + left

    ciphertext = permute(combined, IP\_INV)

    return ciphertext

def des\_decrypt\_block(block, subkeys):

    """Decrypt a single 64-bit block using DES"""

    # Initial permutation

    block = permute(block, IP)

    # Split into left and right halves

    left = block[:32]

    right = block[32:]

    # Perform 16 rounds in reverse order

    for i in range(15, -1, -1):

        left, right = des\_round(left, right, subkeys[i])

    # Combine and perform final permutation

    combined = right + left

    plaintext = permute(combined, IP\_INV)

    return plaintext

def des\_encrypt(plaintext, key):

    """Encrypt plaintext using DES"""

    # Generate subkeys

    subkeys = generate\_subkeys(key)

    # Convert plaintext to binary

    if isinstance(plaintext, str):

        plaintext = str\_to\_bin(pad\_text(plaintext))

    else:

        plaintext = hex\_to\_bin(plaintext, 64)

    # Split into 64-bit blocks

    blocks = [plaintext[i:i+64] for i in range(0, len(plaintext), 64)]

    # Encrypt each block

    ciphertext = ''

    for block in blocks:

        # Pad block if it's less than 64 bits

        if len(block) < 64:

            block = block.ljust(64, '0')

        ciphertext += des\_encrypt\_block(block, subkeys)

    # Return as hex string

    return bin\_to\_hex(ciphertext)

def des\_decrypt(ciphertext, key):

    """Decrypt ciphertext using DES"""

    # Generate subkeys

    subkeys = generate\_subkeys(key)

    # Convert ciphertext to binary

    ciphertext = hex\_to\_bin(ciphertext, len(ciphertext)\*4)

    # Split into 64-bit blocks

    blocks = [ciphertext[i:i+64] for i in range(0, len(ciphertext), 64)]

    # Decrypt each block

    plaintext = ''

    for block in blocks:

        plaintext += des\_decrypt\_block(block, subkeys)

    # Convert to text and unpad

    plaintext = bin\_to\_str(plaintext)

    return unpad\_text(plaintext)

# Test the DES implementation

if \_\_name\_\_ == "\_\_main\_\_":

    # Example usage

    key = "133457799BBCDFF1"  # 64-bit key (56 bits + 8 parity bits)

    plaintext = "Hello DES"

    print("Original Text:", plaintext)

    # Encrypt

    ciphertext = des\_encrypt(plaintext, key)

    print("Encrypted (hex):", ciphertext)

    # Decrypt

    decrypted = des\_decrypt(ciphertext, key)

    print("Decrypted Text:", decrypted)

    # Test with different messages

    test\_messages = [

        "Secret!",

        "12345678",

        "The quick brown fox jumps over the lazy dog",

        "DES is a block cipher"

    ]

    print("\nTesting with different messages:")

    for msg in test\_messages:

        print("\nMessage:", msg)

        encrypted = des\_encrypt(msg, key)

        print("Encrypted:", encrypted)

        decrypted = des\_decrypt(encrypted, key)

        print("Decrypted:", decrypted)

        print("Success:", msg == decrypted)

**OUTPUT:**

Original Text: Hello DES

Encrypted (hex): 5123BA50CC7555833F6139AE5DDDE941

Decrypted Text: Hello DES

Testing with different messages:

Message: Secret!

Encrypted: 65FFBF7E50BFC8B3

Decrypted: Secret!

Success: True

Message: 12345678

Encrypted: 8B96B79529CCA218FDF2E174492922F8

Decrypted: 12345678

Success: True

Message: The quick brown fox jumps over the lazy dog

Encrypted: 7A53B8DC9C17E38EA6FF2538A5F46258C51998DCB1E30327D6DEB14B0264C8A31BDB2BC1B453825968B9BBB7A23F17EB

Decrypted: The quick brown fox jumps over the lazy dog

Success: True

Message: DES is a block cipher

Encrypted: BD467217F05B22A8EBC733CB21EE70016FE5D13657738B54

Decrypted: DES is a block cipher

Success: True