DEPARTMENT OF INFORMATION TECHNOLOGY

COURSE CODE: DJS22ITL504 **DATE:** 06 – 10 – 24

COURSE NAME: Cryptography and Network Security Laboratory **CLASS:** TYBTech

SAPID: 60003220250

EXPERIMENT NO. 4

CO/LO: Design secure system using appropriate security mechanism

AIM / OBJECTIVE:

Implementation and analysis of a S-DES on Plain Text / Image.

DESCRIPTION OF EXPERIMENT:

1. **Take 2 hex values. Use any SHA calculator to compute hash value of (Name+Sap) and apply S-DES on computed hash.**
2. **Apply S-DES on image.**

QUESTIONS:

Show encryption and decryption using key for:

input=E (rollno % 16)....Eg: for roll no=10....plaintext=EA

Discuss the cryptanalysis of of S-DES.

Analyze the performance of S-DES wrt avalanche effect

SOURCE CODE:

a.) <u>S-DES on Plain Text</u>

```
import random


# Permutation Tables and S-Boxes
PermutationP10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
PermutationP8 = [6, 3, 7, 4, 8, 5, 10, 9]
```

```
Initial_Permutation_IP = [2, 6, 3, 1, 4, 8, 5, 7]

Expanded_Permutation_EP = [4, 1, 2, 3, 2, 3, 4, 1]

PermutationP4 = [2, 4, 3, 1]

Inverse_of_Inital_Permutation_IP_inv = [4, 1, 3, 5, 7, 2, 8, 6]

S_Box_0 = [[1, 0, 3, 2], [3, 2, 1, 0], [0, 2, 1, 3], [3, 1, 3, 2]]

S_Box_1 = [[0, 1, 2, 3], [2, 0, 1, 3], [3, 0, 1, 0], [2, 1, 0, 3]]


# Variables to store generated 8-bit keys

key1_8bits = [0] * 8

key2_8bits = [0] * 8


def binary(value, bits=8):

    return format(value, f'0{bits}b')


def swap_bits(arr, size):

    left = arr[:size]

    right = arr[size:]

    return right + left


def binary_to_decimal(bits):

    return int(''.join(map(str, bits)), 2)


def permute(bits, table):

    return [bits[i - 1] for i in table]


def function_(bits, key):

    left_4bits = bits[:4]
```

```
    right_4bits = bits[4:]

    ep = permute(right_4bits, Expanded_Permutation_EP)
    xor_result = [ep[i] ^ key[i] for i in range(8)]

    left_4bits_1 = xor_result[:4]
    right_4bits_1 = xor_result[4:]

    row = binary_to_decimal([left_4bits_1[0], left_4bits_1[3]])
    column = binary_to_decimal([left_4bits_1[1], left_4bits_1[2]])
    value = S_Box_0[row][column]
    str_left = [int(x) for x in binary(value, 2)]

    row = binary_to_decimal([right_4bits_1[0], right_4bits_1[3]])
    column = binary_to_decimal([right_4bits_1[1], right_4bits_1[2]])
    value = S_Box_1[row][column]
    str_right = [int(x) for x in binary(value, 2)]

    r_ = str_left + str_right
    r_p4 = permute(r_, PermutationP4)
    left_4bits = [left_4bits[i] ^ r_p4[i] for i in range(4)]

    return left_4bits + right_4bits

def decryption_of_ciphertext(ciphertext_binary):
    tmp = permute(ciphertext_binary, Initial_Permutation_IP)
    print("Decryption - After Initial Permutation:", ".join(map(str, tmp)))
```

```
    arr1 = function_(tmp, key2_8bits)
    print("Decryption - After Function with Key2:", ".join(map(str, arr1)))
    after_swap = swap_bits(arr1, len(arr1) // 2)
    print("Decryption - After Swap:", ".join(map(str, after_swap)))

    arr2 = function_(after_swap, key1_8bits)
    print("Decryption - After Function with Key1:", ".join(map(str, arr2)))

    return permute(arr2, Inverse_of_Inital_Permutation_IP_inv)

def encryption_of_plaintext(plaintext_binary):
    tmp = permute(plaintext_binary, Initial_Permutation_IP)
    print("Encryption - After Initial Permutation:", ".join(map(str, tmp)))

    arr1 = function_(tmp, key1_8bits)
    print("Encryption - After Function with Key1:", ".join(map(str, arr1)))
    after_swap = swap_bits(arr1, len(arr1) // 2)
    print("Encryption - After Swap:", ".join(map(str, after_swap)))

    arr2 = function_(after_swap, key2_8bits)
    print("Encryption - After Function with Key2:", ".join(map(str, arr2)))

    return permute(arr2, Inverse_of_Inital_Permutation_IP_inv)

def decimal_to_binary(decimal):
    return [int(x) for x in format(decimal, '08b')]
```

```
def shift(binary, n):
    return binary[n:] + binary[:n]

def key_generation(key_10bit):
    key_ = permute(key_10bit, PermutationP10)
    left_side = key_[:5]
    right_side = key_[5:]

    left_shift1 = shift(left_side, 1)
    right_shift1 = shift(right_side, 1)
    key_1 = left_shift1 + right_shift1
    global key1_8bits
    key1_8bits = permute(key_1, PermutationP8)

    left_shift2 = shift(left_side, 2)
    right_shift2 = shift(right_side, 2)
    key_2 = left_shift2 + right_shift2
    global key2_8bits
    key2_8bits = permute(key_2, PermutationP8)

    print(f"Generated 8-Bit Key1 (K1): {''.join(map(str, key1_8bits))}")
    print(f"Generated 8-Bit Key2 (K2): {''.join(map(str, key2_8bits))}")

def generate_random_key(size_of_key):
    return [random.randint(0, 1) for _ in range(size_of_key)]
```

```
def solve():
    key_10bit = generate_random_key(10)

    print(f"Generated 10-Bit Key: {''.join(map(str, key_10bit))}")

    key_generation(key_10bit)

    plaintext = input("Enter Message: ")
    print(f"Plaintext: {''.join(map(str, decimal_to_binary(ord(plaintext[0]))))}")

    cipher_text = []
    for char in plaintext:
        binary_of_plaintext = decimal_to_binary(ord(char))
        print(f"Original Binary of '{char}': {''.join(map(str, binary_of_plaintext))}")
        cipher_text_8bits = encryption_of_plaintext(binary_of_plaintext)
        print(f"Encrypted Binary of '{char}': {''.join(map(str, cipher_text_8bits))}")
        ct = binary_to_decimal(cipher_text_8bits)
        cipher_text.append(ct)

    print(f"Ciphertext: {''.join(map(str, cipher_text_8bits))}")

    decrypted_text = ''
    for ct in cipher_text:
        cipher_text_8bits = decimal_to_binary(ct)
        decrypted_8bits = decryption_of_ciphertext(cipher_text_8bits)
        decrypted_text += chr(binary_to_decimal(decrypted_8bits))
```

```
    print(f"Decrypted Plaintext: {decrypted_text}")


if __name__ == "__main__":
    solve()
```

b.) S-DES on Image

```
import hashlib
import numpy as np
from PIL import Image


def permute(bits, perm):
    return [bits[i] for i in perm]


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


def s_box_lookup(s_box, row, col):
    return s_box[row][col]


def xor(bits1, bits2):
    return [b1 ^ b2 for b1, b2 in zip(bits1, bits2)]


def to_bits(string):
    return [int(bit) for char in string for bit in f"{ord(char):08b}"]


def from_bits(bits):
    return "".join(
        chr(int("".join(map(str, bits[i : i + 8])), 2)) for i in range(0, len(bits), 8)
    )


# Permutation and S-Box Tables
```

```
IP = [1, 5, 2, 0, 3, 7, 4, 6]
IP1 = [3, 0, 2, 4, 6, 1, 7, 5]
EP = [3, 0, 1, 2, 1, 2, 3, 0]

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]]

P4 = [1, 3, 2, 0]


def f_function(right, key):
    expanded_right = permute(right, EP)
    xor_result = xor(expanded_right, key)

    left_sbox = xor_result[:4]
    right_sbox = xor_result[4:]

    row_s0 = (left_sbox[0] << 1) | left_sbox[3]
    col_s0 = (left_sbox[1] << 1) | left_sbox[2]
    s0_result = s_box_lookup(S0, row_s0, col_s0)

    row_s1 = (right_sbox[0] << 1) | right_sbox[3]
    col_s1 = (right_sbox[1] << 1) | right_sbox[2]
    s1_result = s_box_lookup(S1, row_s1, col_s1)

    sbox_output = [
        (s0_result >> 1) & 1,
        s0_result & 1,
        (s1_result >> 1) & 1,
        s1_result & 1,
    ]
    permuted_output = permute(sbox_output, P4)

    return permuted_output


def encrypt_block(block, k1, k2):
    block = [int(b) for b in block]
    block = permute(block, IP)

    left, right = block[:4], block[4:]
```

```
    f_result = f_function(right, k1)
    left = xor(left, f_result)

    left, right = right, left

    f_result = f_function(right, k2)
    left = xor(left, f_result)

    combined = left + right
    ciphertext = permute(combined, IP1)

    return "".join(map(str, ciphertext))


def sdes_key_generation(key_10bit):
    P10 = [2, 4, 1, 6, 3, 9, 0, 8, 7, 5]
    P8 = [5, 2, 6, 3, 7, 4, 9, 8]

    permuted_key = permute([int(b) for b in key_10bit], P10)
    left_half, right_half = permuted_key[:5], permuted_key[5:]

    left_half = left_shift(left_half, 1)
    right_half = left_shift(right_half, 1)
    key1 = permute(left_half + right_half, P8)

    left_half = left_shift(left_half, 2)
    right_half = left_shift(right_half, 2)
    key2 = permute(left_half + right_half, P8)

    return key1, key2


def encrypt_text(text, key_10bit):
    key1, key2 = sdes_key_generation(key_10bit)

    blocks = [text[i : i + 8] for i in range(0, len(text), 8)]
    encrypted_blocks = [encrypt_block(block, key1, key2) for block in blocks]

    return from_bits([int(b) for block in encrypted_blocks for b in block])
```

```
def decrypt_block(block, k1, k2):
    # Convert block to list of bits
    block = [int(b) for b in block]
    # Apply Initial Permutation (IP)
    block = permute(block, IP)

    # Split into left and right halves
    left, right = block[:4], block[4:]

    # Apply Feistel function with key k2
    f_result = f_function(right, k2)
    left = xor(left, f_result)

    # Swap left and right
    left, right = right, left

    # Apply Feistel function with key k1
    f_result = f_function(right, k1)
    left = xor(left, f_result)

    # Combine halves
    combined = left + right
    # Apply Inverse Permutation (IP1)
    plaintext_bits = permute(combined, IP1)

    return "".join(map(str, plaintext_bits))


def decrypt_text(encrypted_text, key_10bit):
    key1, key2 = sdes_key_generation(key_10bit)

    # Ensure encrypted text is in bits
    encrypted_bits = to_bits(encrypted_text)

    # Split into 8-bit blocks
    blocks = [encrypted_bits[i : i + 8] for i in range(0, len(encrypted_bits), 8)]
    decrypted_blocks = [decrypt_block(block, key1, key2) for block in blocks]

    return from_bits([int(b) for block in decrypted_blocks for b in block])
```

```
def image_to_text(image_path):
    # Open the image and convert to grayscale
    image = Image.open(image_path).convert("L")  # 'L' mode converts to grayscale

    # Convert image to numpy array
    image_array = np.array(image)

    # Flatten the array into a 1D array of pixel values
    flattened_array = image_array.flatten()

    # Convert the array into a text string
    text = "".join(chr(pixel) for pixel in flattened_array)

    return text, image_array.shape


def text_to_image(text, shape, output_path):
    # Convert text back to pixel values
    pixel_values = [ord(char) for char in text]

    # Ensure the pixel values match the shape of the original image
    if len(pixel_values) != np.prod(shape):
        raise ValueError("Text length does not match the size of the original image.")

    # Reshape the pixel values to the original shape
    image_array = np.array(pixel_values).reshape(shape)

    # Create an image from the pixel values
    image = Image.fromarray(image_array.astype(np.uint8))

    # Save or show the image
    image.save(output_path)
    print(f"Image saved to {output_path}")

# Example usage
key_10bit = "1010000010"  # Example 10-bit key
# text = hashlib.sha256(input("Enter Input: ").encode()).hexdigest()

img = image_to_text("car.jpg")
```

```
binary_text = to_bits(img[0])
encrypted_text = encrypt_text(binary_text, key_10bit)
text_to_image(encrypted_text,img[1],"SDES_encrypted.jpg")

decrypted_text = decrypt_text(encrypted_text, key_10bit)
text_to_image(decrypted_text,img[1],"SDES_decrypted.jpg")
```

OUTPUT:

a.) S-DES on Plain Text

```
Generated 10-Bit Key: 1011100100
Generated 8-Bit Key1 (K1): 10011100
Generated 8-Bit Key2 (K2): 01110110
Enter Message: send me your password
Plaintext: 01110011
Original Binary of 's': 01110011
Encryption - After Initial Permutation: 10101101
Encryption - After Function with Key1: 11001101
Encryption - After Swap: 11011100
Encryption - After Function with Key2: 00101100
Encrypted Binary of 's': 00110001
Original Binary of 'e': 01100101
Encryption - After Initial Permutation: 11100100
Encryption - After Function with Key1: 01000100
Encryption - After Swap: 01000100
Encryption - After Function with Key2: 11000100
Encrypted Binary of 'e': 01000101
Original Binary of 'n': 01101110
Encryption - After Initial Permutation: 11100011
Encryption - After Function with Key1: 01100011
Encryption - After Swap: 00110110
Encryption - After Function with Key2: 10100110
Encrypted Binary of 'n': 01101001
Original Binary of 'd': 01100100
Encryption - After Initial Permutation: 11100000
Encryption - After Function with Key1: 00110000
```

```
Encryption - After Swap: 00000011
Encryption - After Function with Key2: 10010011
Encrypted Binary of 'd': 11001010
Original Binary of ' ': 00100000
Encryption - After Initial Permutation: 00100000
Encryption - After Function with Key1: 11110000
Encryption - After Swap: 00001111
Encryption - After Function with Key2: 00101111
Encrypted Binary of ' ': 00111011
Original Binary of 'm': 01101101
Encryption - After Initial Permutation: 11100110
Encryption - After Function with Key1: 11110110
Encryption - After Swap: 01101111
Encryption - After Function with Key2: 01001111
Encrypted Binary of 'm': 00011111
Original Binary of 'e': 01100101
Encryption - After Initial Permutation: 11100100
Encryption - After Function with Key1: 01000100
Encryption - After Swap: 01000100
Encryption - After Function with Key2: 11000100
Encrypted Binary of 'e': 01000101
Original Binary of ' ': 00100000
Encryption - After Initial Permutation: 00100000
Encryption - After Function with Key1: 11110000
Encryption - After Swap: 00001111
Encryption - After Function with Key2: 00101111
Encrypted Binary of ' ': 00111011
Original Binary of 'y': 01111001
Encryption - After Initial Permutation: 10101110
```

```
Encryption - After Function with Key1: 00011110
Encryption - After Swap: 11100001
Encryption - After Function with Key2: 11010001
Encrypted Binary of 'y': 11000110
Original Binary of 'o': 01101111
Encryption - After Initial Permutation: 11100111
Encryption - After Function with Key1: 10100111
Encryption - After Swap: 01111010
Encryption - After Function with Key2: 01111010
Encrypted Binary of 'o': 10111100
Original Binary of 'u': 01110101
Encryption - After Initial Permutation: 11101100
Encryption - After Function with Key1: 10111100
Encryption - After Swap: 11001011
Encryption - After Function with Key2: 11111011
Encrypted Binary of 'u': 11111110
Original Binary of 'r': 01110010
Encryption - After Initial Permutation: 10101001
Encryption - After Function with Key1: 01001001
Encryption - After Swap: 10010100
Encryption - After Function with Key2: 00010100
Encrypted Binary of 'r': 10000001
Original Binary of ' ': 00100000
Encryption - After Initial Permutation: 00100000
Encryption - After Function with Key1: 11110000
Encryption - After Swap: 00001111
Encryption - After Function with Key2: 00101111
Encrypted Binary of ' ': 00111011
Original Binary of 'p': 01110000
```

```
Encryption - After Initial Permutation: 10101000
Encryption - After Function with Key1: 00111000
Encryption - After Swap: 10000011
Encryption - After Function with Key2: 00010011
Encrypted Binary of 'p': 10001010
Original Binary of 'a': 01100001
Encryption - After Initial Permutation: 10100100
Encryption - After Function with Key1: 00000100
Encryption - After Swap: 01000000
Encryption - After Function with Key2: 00100000
Encrypted Binary of 'a': 00100000
Original Binary of 's': 01110011
Encryption - After Initial Permutation: 10101101
Encryption - After Function with Key1: 11001101
Encryption - After Swap: 11011100
Encryption - After Function with Key2: 00101100
Encrypted Binary of 's': 00110001
Original Binary of 's': 01110011
Encryption - After Initial Permutation: 10101101
Encryption - After Function with Key1: 11001101
Encryption - After Swap: 11011100
Encryption - After Function with Key2: 00101100
Encrypted Binary of 's': 00110001
Original Binary of 'w': 01110111
Encryption - After Initial Permutation: 11101101
Encryption - After Function with Key1: 10001101
Encryption - After Swap: 11011000
Encryption - After Function with Key2: 10101000
Encrypted Binary of 'w': 01110000
```

```
Original Binary of 'o': 01101111
Encryption - After Initial Permutation: 11100111
Encryption - After Function with Key1: 10100111
Encryption - After Swap: 01111010
Encryption - After Function with Key2: 01111010
Encrypted Binary of 'o': 10111100
Original Binary of 'r': 01110010
Encryption - After Initial Permutation: 10101001
Encryption - After Function with Key1: 01001001
Encryption - After Swap: 10010100
Encryption - After Function with Key2: 00010100
Encrypted Binary of 'r': 10000001
Original Binary of 'd': 01100100
Encryption - After Initial Permutation: 11100000
Encryption - After Function with Key1: 00110000
Encryption - After Swap: 00000011
Encryption - After Function with Key2: 10010011
Encrypted Binary of 'd': 11001010
Ciphertext: 11001010
Decryption - After Initial Permutation: 00101100
Decryption - After Function with Key2: 11011100
Decryption - After Swap: 11001101
Decryption - After Function with Key1: 10101101
Decryption - After Initial Permutation: 11000100
Decryption - After Function with Key2: 01000100
Decryption - After Swap: 01000100
Decryption - After Function with Key1: 11100100
Decryption - After Initial Permutation: 10100110
Decryption - After Function with Key2: 00110110
```

```
Decryption - After Swap: 01100011
Decryption - After Function with Key1: 11100011
Decryption - After Initial Permutation: 10010011
Decryption - After Function with Key2: 00000011
Decryption - After Swap: 00110000
Decryption - After Function with Key1: 11100000
Decryption - After Initial Permutation: 00101111
Decryption - After Function with Key2: 00001111
Decryption - After Swap: 11110000
Decryption - After Function with Key1: 00100000
Decryption - After Initial Permutation: 01001111
Decryption - After Function with Key2: 01101111
Decryption - After Swap: 11110110
Decryption - After Function with Key1: 11100110
Decryption - After Initial Permutation: 11000100
Decryption - After Function with Key2: 01000100
Decryption - After Swap: 01000100
Decryption - After Function with Key1: 11100100
Decryption - After Initial Permutation: 00101111
Decryption - After Function with Key2: 00001111
Decryption - After Swap: 11110000
Decryption - After Function with Key1: 00100000
Decryption - After Initial Permutation: 11010001
Decryption - After Function with Key2: 11100001
Decryption - After Swap: 00011110
Decryption - After Function with Key1: 10101110
Decryption - After Initial Permutation: 01111010
Decryption - After Function with Key2: 01111010
Decryption - After Swap: 10100111
```

```
Decryption - After Function with Key1: 11100111
Decryption - After Initial Permutation: 11111011
Decryption - After Function with Key2: 11001011
Decryption - After Swap: 10111100
Decryption - After Function with Key1: 11101100
Decryption - After Initial Permutation: 00010100
Decryption - After Function with Key2: 10010100
Decryption - After Swap: 01001001
Decryption - After Function with Key1: 10101001
Decryption - After Initial Permutation: 00101111
Decryption - After Function with Key2: 00001111
Decryption - After Swap: 11110000
Decryption - After Function with Key1: 00100000
Decryption - After Initial Permutation: 00010011
Decryption - After Function with Key2: 10000011
Decryption - After Swap: 00111000
Decryption - After Function with Key1: 10101000
Decryption - After Initial Permutation: 00100000
Decryption - After Function with Key2: 01000000
Decryption - After Swap: 00000100
Decryption - After Function with Key1: 10100100
Decryption - After Initial Permutation: 00101100
Decryption - After Initial Permutation: 00101100
Decryption - After Function with Key2: 11011100
Decryption - After Swap: 11001101
Decryption - After Function with Key1: 10101101
Decryption - After Initial Permutation: 00101100
Decryption - After Function with Key2: 11011100
Decryption - After Swap: 11001101
```

```
Decryption - After Function with Key1: 10100100
Decryption - After Initial Permutation: 00101100
Decryption - After Initial Permutation: 00101100
Decryption - After Function with Key2: 11011100
Decryption - After Swap: 11001101
Decryption - After Function with Key1: 10101101
Decryption - After Initial Permutation: 00101100
Decryption - After Function with Key2: 11011100
Decryption - After Swap: 11001101
Decryption - After Function with Key1: 10101101
Decryption - After Initial Permutation: 10101000
Decryption - After Function with Key2: 11011000
Decryption - After Swap: 10001101
Decryption - After Function with Key1: 11101101
Decryption - After Initial Permutation: 01111010
Decryption - After Function with Key2: 01111010
Decryption - After Swap: 10100111
Decryption - After Function with Key1: 11100111
Decryption - After Initial Permutation: 00010100
Decryption - After Function with Key2: 10010100
Decryption - After Swap: 01001001
Decryption - After Function with Key1: 10101001
Decryption - After Initial Permutation: 10010011
Decryption - After Function with Key2: 00000011
Decryption - After Swap: 00110000
Decryption - After Function with Key1: 11100000
Decrypted Plaintext: send me your password
```

b.) S-DES on Image

Original image:

S-DES encrypted image:

S-DES decrypted image:

CONCLUSION: In this experiment, we have successfully implemented and analysed S-DES on a plain text and on image.

REFERENCES:

1. Bruce Schneier, “Applied Cryptography, Protocols Algorithms and Source Code in C”, 2nd Edition, Wiley, 2006.

2. Mark Stamp, “Information Security Principles and Practice”, 2nd Edition, Wiley, 2011.

3. Behrouz A. Ferouzan, “Cryptography & Network Security”, 3rd Edition, Tata McGraw Hill, Nov 2015.

4. William Stallings, “Cryptography and Network Security, Principles and Practice”, 7th Edition, Pearson Education, 2017.