Ex. No. 4	Implement Symmetric Cipher (Simplified DES)
Date of Exercise	24.08.2023

Aim

To Implement a Symmetric Cipher – Simplified DES

Description:

The Simplified Data Encryption Standard, commonly known as S-DES, is a symmetric-key block cipher that is a simplified version of the original Data Encryption Standard (DES). S-DES operates on fixed-size blocks of data and employs a symmetric key for both encryption and decryption. It was primarily designed for educational purposes and understanding the fundamental concepts of block ciphers.

S-DES uses a 10-bit key to perform data encryption. The encryption process involves several steps, including initial and final permutations, key generation, and multiple rounds of substitution and permutation operations. It is based on the Feistel network structure, where the plaintext is divided into two halves, and each half undergoes multiple rounds of transformation. The S-DES cipher employs permutation tables and S-boxes to create confusion and diffusion, enhancing the security of the encrypted data.

While S-DES is not suitable for modern cryptographic applications due to its limited key length and vulnerability to brute-force attacks, it serves as a valuable educational tool for introducing students and enthusiasts to fundamental encryption techniques. Understanding S-DES can pave the way for a deeper comprehension of more advanced encryption algorithms and their security considerations. Despite its simplicity, S-DES demonstrates key principles used in block ciphers, making it an essential building block in the field of cryptography education.

Algorithm:

Step 1: Input: Prompt the user to enter a plaintext word and a key word as words.

Step 2: Define a function text_to_binary(text) that converts a given text string into binary representation.

Initialize an empty string called binary_string.

Iterate through each character, char, in the input text.

For each character, convert it to its binary representation using bin(ord(char))[2:].zfill(8].

Append the binary representation of the character to the binary_string.

Return the binary_string.

Step 3: Convert the plaintext word and key word to binary representations using the text_to_binary(text) function. Store the binary representations in the variables plaintext and key.

Step 4: Define a function permute(original, permutation) that takes an original string and a permutation order and returns a new string with the characters permuted according to the given order.

Step 5: Define a function generate_round_keys(key) that generates two round keys (round_key1 and round_key2) for the S-DES encryption.

Step 6: Define a function initial_permutation(plaintext) that performs the initial permutation of the plaintext using a given permutation order (ip).

Step 7: Define a function inverse_initial_permutation(ciphertext) that performs the inverse initial permutation of the ciphertext using a given permutation order (ip_inv).

Use the permute function to apply the inverse initial permutation to the ciphertext.

Step 8: Define a function f_function(right_half, round_key) that performs the F-function of S-DES encryption.

Step 9: Define a function round_function(left_half, right_half, round_key) that performs one round of S-DES encryption.

Step 10: Define a function sdes_encrypt(plaintext, key) that performs S-DES encryption

Program:

```
def permute(original, permutation):
    return ''.join(original[i - 1] for i in permutation)


def generate_round_keys(key):
    p10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6] #permutation table
    p8 = [6, 3, 7, 4, 8, 5, 10, 9]

    key = permute(key, p10)#key = 10 bit

    left_half = key[:5]
    right_half = key[5:]

    # Left circular shift
```

```
    left_half = left_half[1:] + left_half[0]
    right_half = right_half[1:] + right_half[0]

    round_key1 = permute(left_half + right_half, p8)

    # Left circular shift again
    left_half = left_half[2:] + left_half[:2]
    right_half = right_half[2:] + right_half[:2]

    round_key2 = permute(left_half + right_half, p8)

    return round_key1, round_key2

def initial_permutation(plaintext):
    ip = [2, 6, 3, 1, 4, 8, 5, 7]
    return permute(plaintext, ip)

def inverse_initial_permutation(ciphertext):
    ip_inv = [4, 1, 3, 5, 7, 2, 8, 6]
    return permute(ciphertext, ip_inv)
```

```
def f_function(right_half, round_key):
    expansion = [4, 1, 2, 3, 2, 3, 4, 1]
    p4 = [2, 4, 3, 1]

    expanded = permute(right_half, expansion)
    xor_result = ''.join(str(int(a) ^ int(b)) for a, b in zip(expanded, round_key))

    sboxes = [
        [['00', '01', '10', '11'], ['10', '00', '01', '11'], ['11', '00', '01', '00'], ['10', '01', '00', '11']],
        [['00', '01', '10', '11'], ['10', '00', '01', '11'], ['11', '00', '01', '00'], ['10', '01', '00', '11']]
    ]

    sbox_output = ''
    for i in range(2):
        row = int(xor_result[i * 4] + xor_result[i * 4 + 3], 2)
        col = int(xor_result[i * 4 + 1:i * 4 + 3], 2)
        sbox_output += sboxes[i][row][col]

    return permute(sbox_output, p4)
```

```
def round_function(left_half, right_half, round_key):
    new_right = ''.join(str(int(a) ^ int(b)) for a, b in zip(right_half, f_function(right_half,
round_key)))
    new_left = right_half
    return new_left, new_right


def sdes_encrypt(plaintext, key):
    round_key1, round_key2 = generate_round_keys(key)
    plaintext = initial_permutation(plaintext)

    left_half = plaintext[:4]
    right_half = plaintext[4:]

    left_half, right_half = round_function(left_half, right_half, round_key1)
    left_half, right_half = right_half, left_half
    left_half, right_half = round_function(left_half, right_half, round_key2)

    ciphertext = left_half + right_half
    ciphertext = inverse_initial_permutation(ciphertext)
```

```
    return ciphertext


def sdes_decrypt(ciphertext, key):
    round_key1, round_key2 = generate_round_keys(key)
    ciphertext = initial_permutation(ciphertext)

    left_half = ciphertext[:4]
    right_half = ciphertext[4:]

    left_half, right_half = round_function(left_half, right_half, round_key2)
    left_half, right_half = right_half, left_half
    left_half, right_half = round_function(left_half, right_half, round_key1)

    plaintext = left_half + right_half
    plaintext = inverse_initial_permutation(plaintext)

    return plaintext


def text_to_binary(text):
```

```
    binary_string = ""
    for char in text:
        binary_char = bin(ord(char))[2:].zfill(8)
        binary_string += binary_char
    return binary_string

# Take user input for plaintext and key as words
plaintext_word = input("Enter the plaintext word: ")
key_word = input("Enter the key word: ")

# Convert words to binary
plaintext = text_to_binary(plaintext_word)
key = text_to_binary(key_word)

encrypted = sdes_encrypt(plaintext, key)
decrypted = sdes_decrypt(encrypted, key)

print("Plaintext (binary):", plaintext)
print("Key (binary):", key)
print("Encrypted:", encrypted)
```

```
print("Decrypted:", decrypted)
```

Output Screenshot:

```
Enter the plaintext word:  rubanginosingh
Enter the key word:  ruban
Plaintext (binary): 01110010011101010110001001100001011011100110011101101001011
01110011011110111001101101001011011100110011101101000
Key (binary): 0111001001110101011000100110000101101110
Encrypted: 11001001
Decrypted: 00111111
```

Result

Thus, the experiment to Implement Symmetric Cipher (Simplified Data Encryption Standards) is carried out successfully and obtained the required output.