|  |  |
| --- | --- |
| **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)

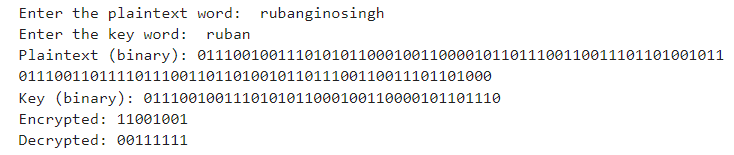
print("Plaintext (binary):", plaintext)

print("Key (binary):", key)

print("Encrypted:", encrypted)

print("Decrypted:", decrypted)

**Output Screenshot:**

****

**Result**

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