|  |  |
| --- | --- |
| **Ex. No. 4** | **Implement Symmetric cipher** |
| **Date of Exercise** | **02.09.2024** |

**Aim**

To write a Python program to implement a simplified Data Encryption Standard (DES) using Python Language.

**Description**

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are used in the algorithm. The algorithm involves combinations, substitutions and permutations between the text to be encrypted and the key, while ensuring the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k1 to k16. Given that "only" 2^56 bits are used for encrypting, there can be 256 different keys.

The main parts of the algorithm are as follows:

● Fractioning of the text into 64-bit blocks

●Initial permutation of blocks

●Breakdown of the blocks into two parts: left and right, named L and R

Permutation and substitution steps are repeated 16 times

● Re-joining of the left and right parts then inverse initial permutation

1. **Perform the logic behind simplified DES**

**Algorithm:**

STEP-1: Read the 64-bit plain text.

STEP-2: Split it into two 32-bit blocks and store it in two different arrays.

STEP-3: Perform XOR operation between these two arrays.

STEP-4: The output obtained is stored as the second 32-bit sequence and

the original second 32-bit sequence forms the first part.

STEP-5: Thus the encrypted 64-bit cipher text is obtained in this way.

Repeat the same process for the remaining plain text characters.

**Program**

def string\_to\_binary(text):

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

def binary\_to\_string(binary):

chars = [binary[i:i+8] for i in range(0, len(binary), 8)]

return ''.join(chr(int(char, 2)) for char in chars)

def permute(k, arr, n):

return [k[i - 1] for i in arr]

def shift\_left(k, shifts):

return k[shifts:] + k[:shifts]

def xor(a, b):

return [str(int(x) ^ int(y)) for x, y in zip(a, b)]

def sbox\_output(bits, sbox):

row = int(bits[0] + bits[3], 2)

col = int(bits[1] + bits[2], 2)

return format(sbox[row][col], '02b')

def encrypt\_decrypt(plain\_text, key, encrypt=True):

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

ip\_inv = [4, 1, 3, 5, 7, 2, 8, 6]

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

p4 = [2, 4, 3, 1]

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]]

p10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

p8 = [6, 3, 7, 4, 8, 5, 10, 9]

key\_p10 = permute(key, p10, 10)

print(f"After P10 permutation: {''.join(key\_p10)}")

left\_half, right\_half = key\_p10[:5], key\_p10[5:]

left\_half = shift\_left(left\_half, 1)

right\_half = shift\_left(right\_half, 1)

shifted\_key\_1 = left\_half + right\_half

print(f"Shifted halves (1st): {''.join(left\_half)} | {''.join(right\_half)}")

key\_1 = permute(shifted\_key\_1, p8, 8)

print(f"Key-1: {''.join(key\_1)}")

left\_half = shift\_left(left\_half, 2)

right\_half = shift\_left(right\_half, 2)

shifted\_key\_2 = left\_half + right\_half

print(f"Shifted halves (2nd): {''.join(left\_half)} | {''.join(right\_half)}")

key\_2 = permute(shifted\_key\_2, p8, 8)

print(f"Key-2: {''.join(key\_2)}")

k1, k2 = (key\_1, key\_2) if encrypt else (key\_2, key\_1)

pt\_ip = permute(plain\_text, ip, 8)

print(f"After initial permutation (IP): {''.join(pt\_ip)}")

left\_half, right\_half = pt\_ip[:4], pt\_ip[4:]

print(f"Initial L = {''.join(left\_half)}, R = {''.join(right\_half)}")

def fk(l, r, key):

r\_ep = permute(r, ep, 8)

print(f"After expansion/permutation (EP): {''.join(r\_ep)}")

xor\_result = xor(r\_ep, key)

print(f"After XOR with key: {''.join(xor\_result)}")

left\_sbox\_in, right\_sbox\_in = xor\_result[:4], xor\_result[4:]

sbox\_out = sbox\_output(left\_sbox\_in, s0) + sbox\_output(right\_sbox\_in, s1)

print(f"S-box output: {sbox\_out}")

p4\_result = permute(sbox\_out, p4, 4)

print(f"After P4 permutation: {''.join(p4\_result)}")

l\_xor\_p4 = xor(l, p4\_result)

print(f"After XOR with left: {''.join(l\_xor\_p4)}")

return l\_xor\_p4, r

left\_half, right\_half = fk(left\_half, right\_half, k1)

left\_half, right\_half = right\_half, left\_half

print(f"After swap: L = {''.join(left\_half)}, R = {''.join(right\_half)}")

left\_half, right\_half = fk(left\_half, right\_half, k2)

combined = left\_half + right\_half

print(f"Before inverse IP: {''.join(combined)}")

cipher\_text = permute(combined, ip\_inv, 8)

print(f"After inverse permutation (IP-1): {''.join(cipher\_text)}")

return ''.join(cipher\_text)

print('Perform the logic behind simplified DES')

plaintext = input("Enter a plaintext (word): ")

key = input("Enter a 10-bit key: ")

plaintext\_binary = string\_to\_binary(plaintext)

cipher\_binary = ''

for i in range(0, len(plaintext\_binary), 8):

block = plaintext\_binary[i:i+8]

print(f"\n--- Encrypting block {block} ---")

cipher\_binary += encrypt\_decrypt(block, key, encrypt=True)

decrypted\_binary = ''

for i in range(0, len(cipher\_binary), 8):

block = cipher\_binary[i:i+8]

print(f"\n--- Decrypting block {block} ---")

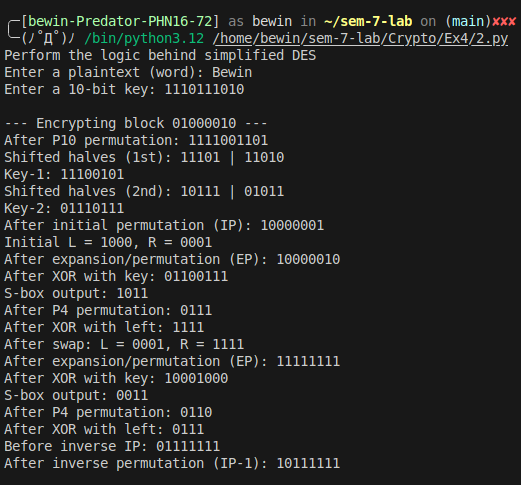
decrypted\_binary += encrypt\_decrypt(block, key, encrypt=False)

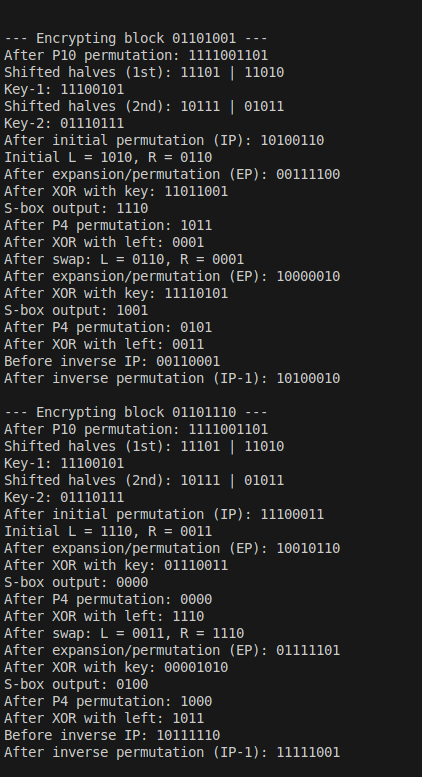
decrypted\_text = binary\_to\_string(decrypted\_binary)

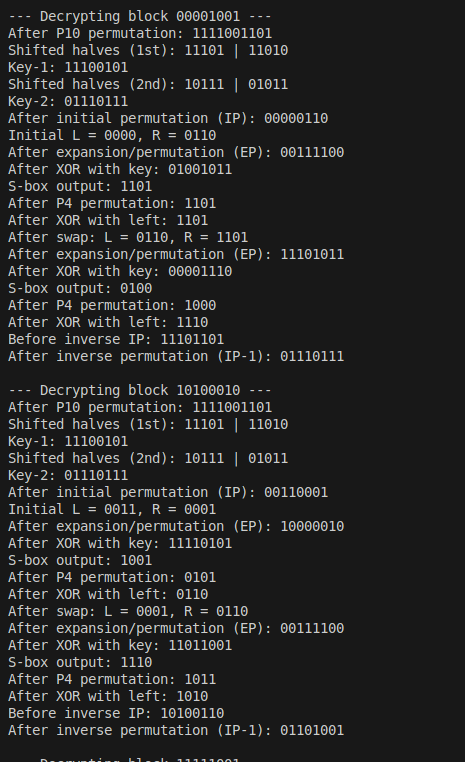
print(f"Cipher Binary: {cipher\_binary}")

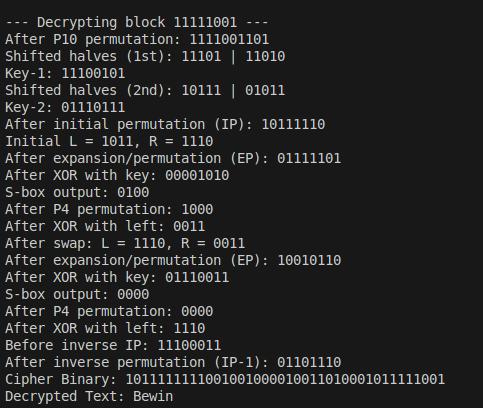
print(f"Decrypted Text: {decrypted\_text}")

**Output Screenshot**









**Result**

The program has executed successfully and the output is displayed in the console.