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.

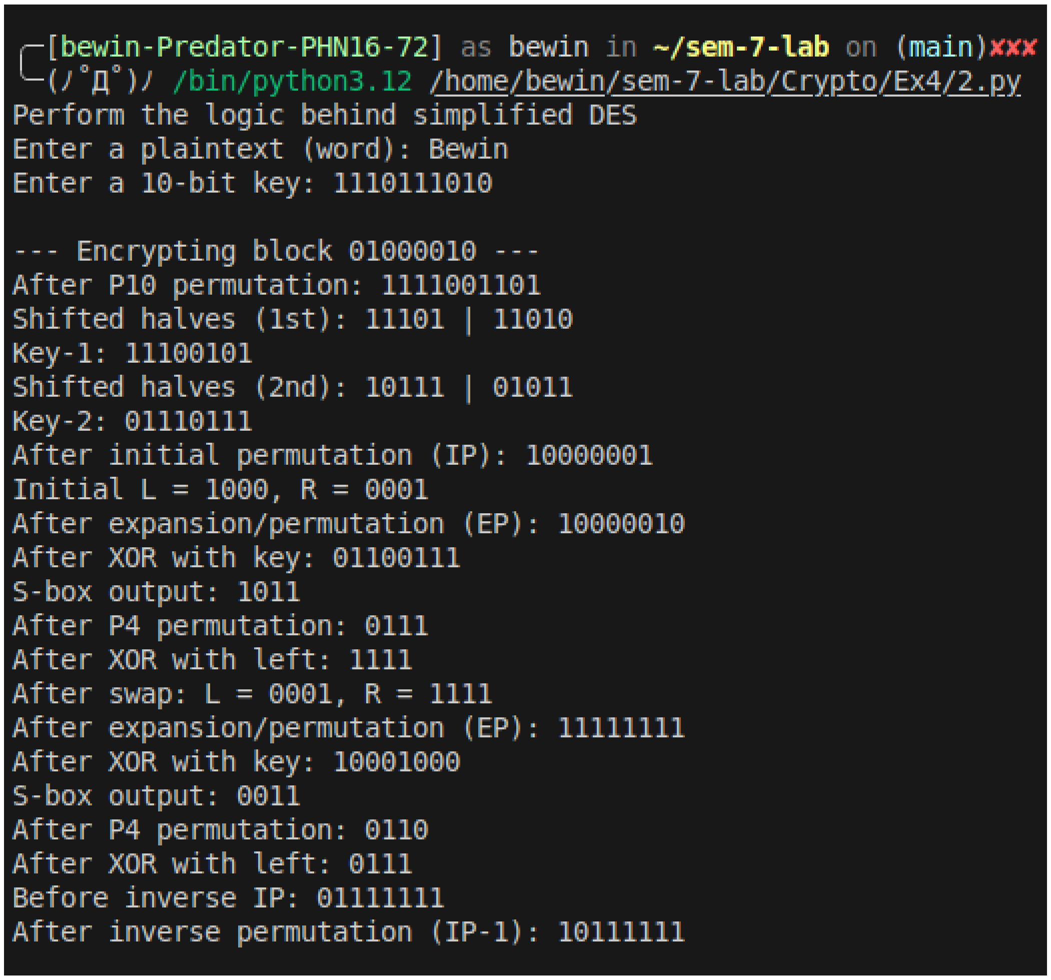
Program

```
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)
```

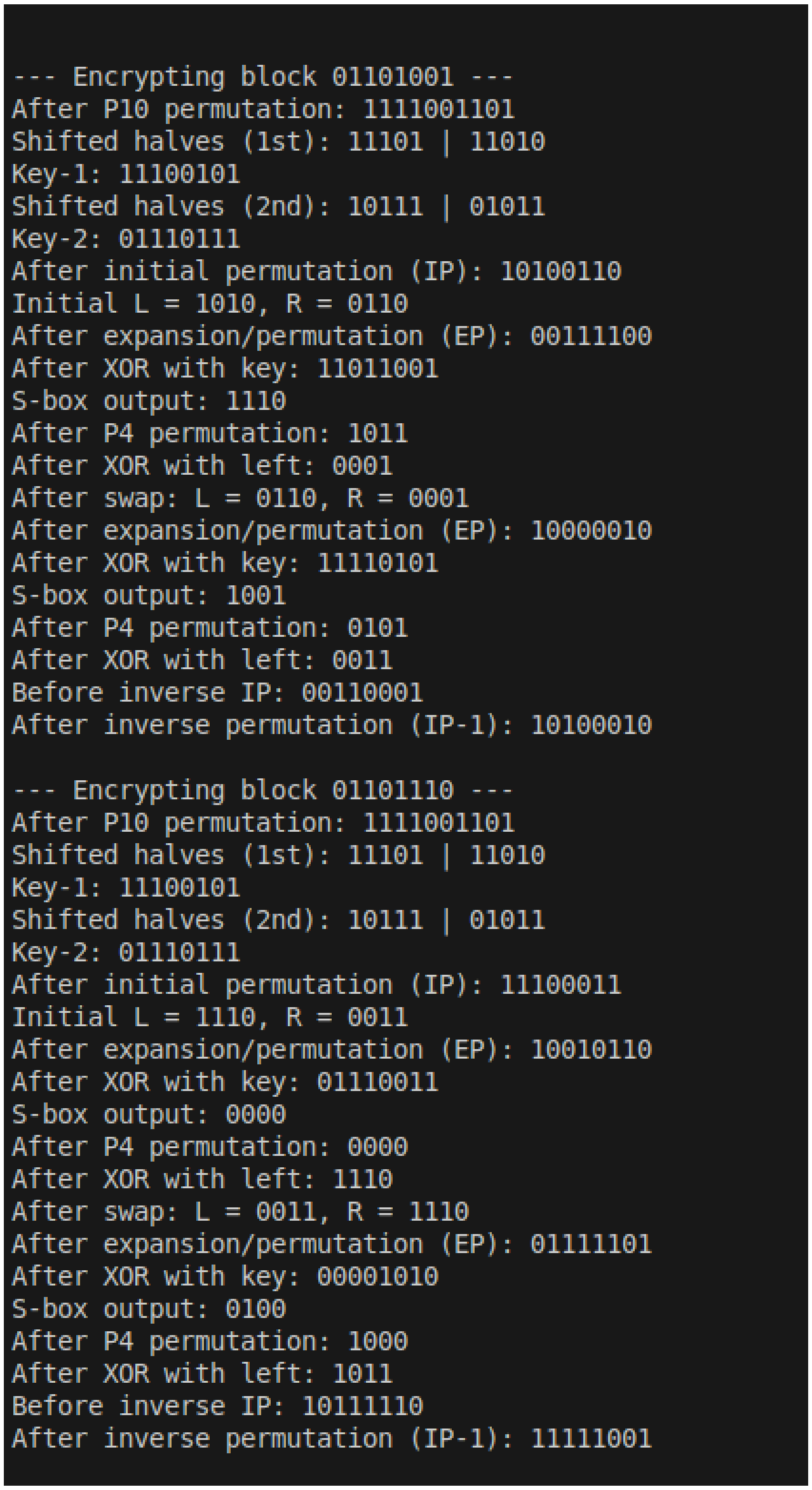
```
right_half = shift_left(right_half, 2)
shifted_key_2 = left_half + right_half
print(f"Shifted halves (2nd): {''.join(left_half)} | {''.join(right_half)}")
key_2 = permute(shifted_key_2, p8, 8)
print(f"Key-2: {''.join(key_2)}")
k1, k2 = (key_1, key_2) if encrypt else (key_2, key_1)
pt_ip = permute(plain_text, 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}")
```

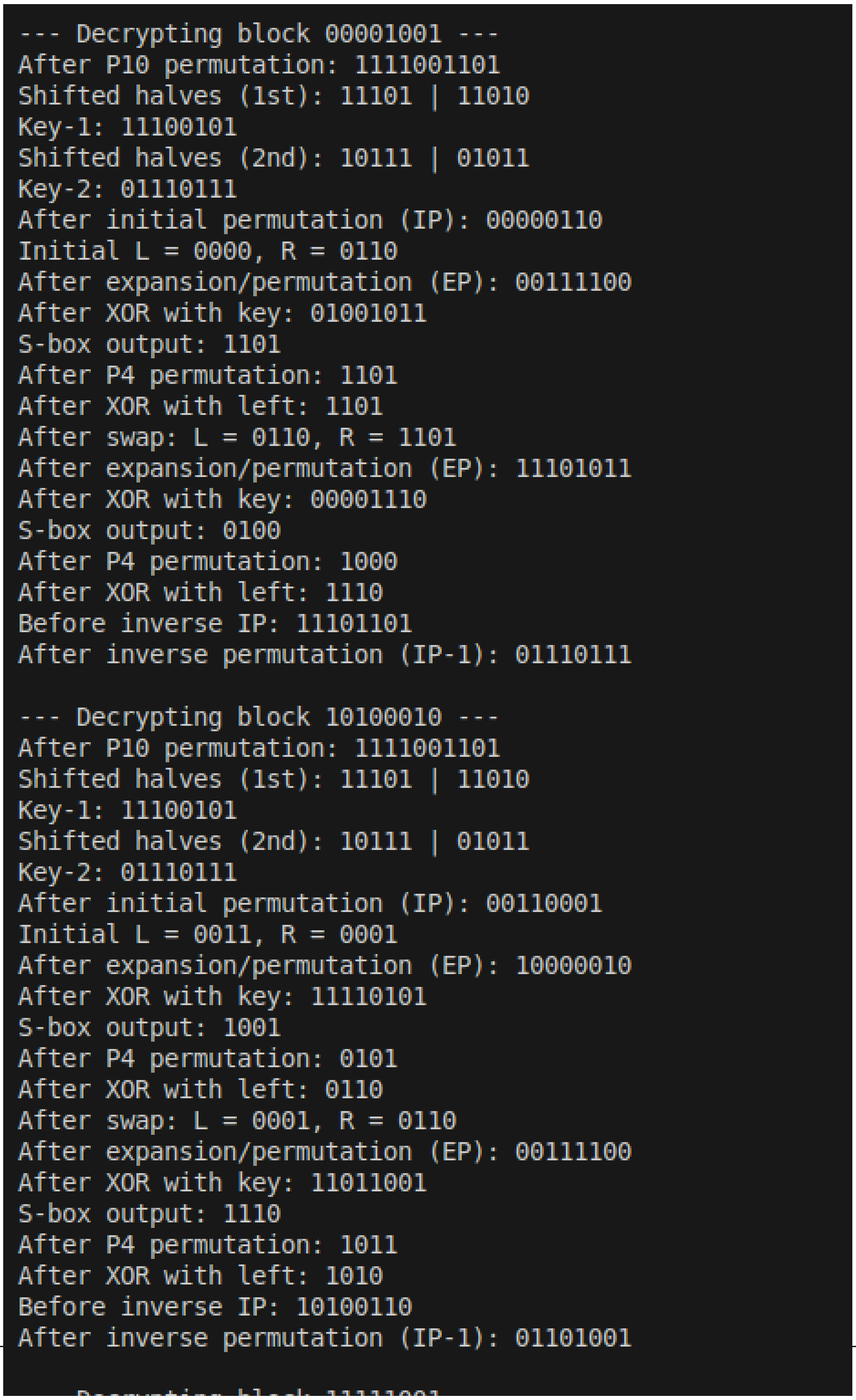
Output Screenshot



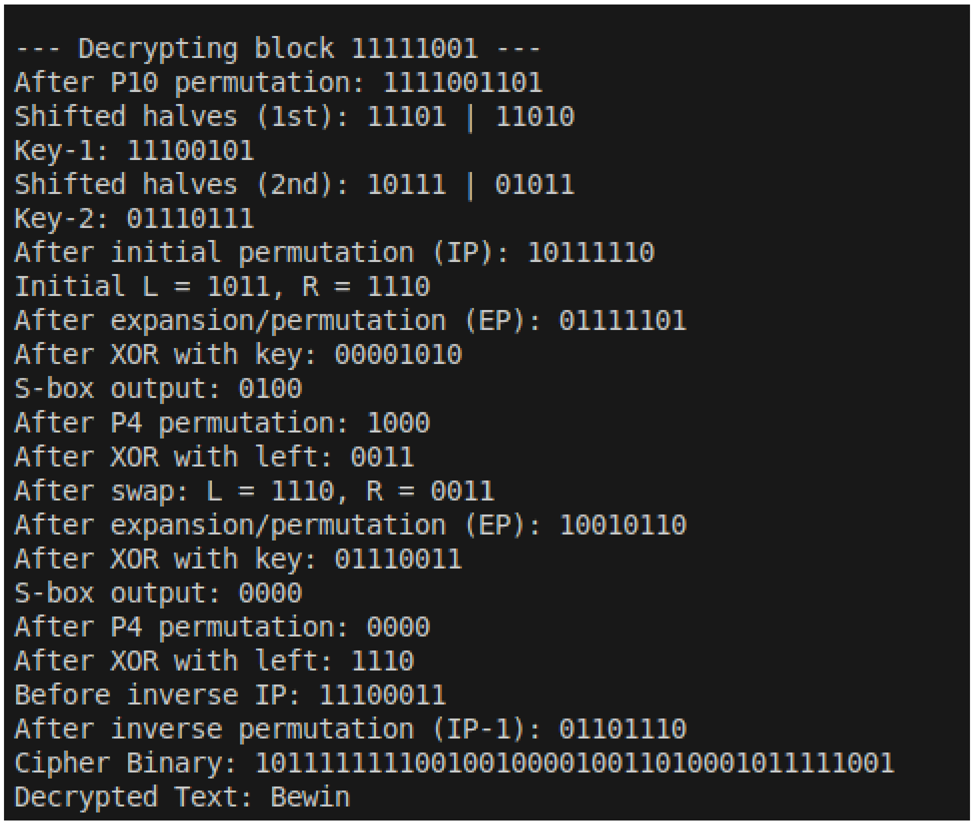
```
┌[bewin-Predator-PHN16-72] as bewin in ~/sem-7-lab on (main)xxx
└(ﾉ°Д°)ﾉ /bin/python3.12 /home/bewin/sem-7-lab/Crypto/Ex4/2.py
Perform the logic behind simplified DES
Enter a plaintext (word): Bewin
Enter a 10-bit key: 1110111010

--- Encrypting block 01000010 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 10000001
Initial L = 1000, R = 0001
After expansion/permutation (EP): 10000010
After XOR with key: 01100111
S-box output: 1011
After P4 permutation: 0111
After XOR with left: 1111
After swap: L = 0001, R = 1111
After expansion/permutation (EP): 11111111
After XOR with key: 10001000
S-box output: 0011
After P4 permutation: 0110
After XOR with left: 0111
Before inverse IP: 01111111
After inverse permutation (IP-1): 10111111
```



```
--- Encrypting block 01101001 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 10100110
Initial L = 1010, R = 0110
After expansion/permutation (EP): 00111100
After XOR with key: 11011001
S-box output: 1110
After P4 permutation: 1011
After XOR with left: 0001
After swap: L = 0110, R = 0001
After expansion/permutation (EP): 10000010
After XOR with key: 11110101
S-box output: 1001
After P4 permutation: 0101
After XOR with left: 0011
Before inverse IP: 00110001
After inverse permutation (IP-1): 10100010

--- Encrypting block 01101110 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 11100011
Initial L = 1110, R = 0011
After expansion/permutation (EP): 10010110
After XOR with key: 01110011
S-box output: 0000
After P4 permutation: 0000
After XOR with left: 1110
After swap: L = 0011, R = 1110
After expansion/permutation (EP): 01111101
After XOR with key: 00001010
S-box output: 0100
After P4 permutation: 1000
After XOR with left: 1011
Before inverse IP: 10111110
After inverse permutation (IP-1): 11111001
```



```
--- Decrypting block 00001001 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 00000110
Initial L = 0000, R = 0110
After expansion/permutation (EP): 00111100
After XOR with key: 01001011
S-box output: 1101
After P4 permutation: 1101
After XOR with left: 1101
After swap: L = 0110, R = 1101
After expansion/permutation (EP): 11101011
After XOR with key: 00001110
S-box output: 0100
After P4 permutation: 1000
After XOR with left: 1110
Before inverse IP: 11101101
After inverse permutation (IP-1): 01110111

--- Decrypting block 10100010 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 00110001
Initial L = 0011, R = 0001
After expansion/permutation (EP): 10000010
After XOR with key: 11110101
S-box output: 1001
After P4 permutation: 0101
After XOR with left: 0110
After swap: L = 0001, R = 0110
After expansion/permutation (EP): 00111100
After XOR with key: 11011001
S-box output: 1110
After P4 permutation: 1011
After XOR with left: 1010
Before inverse IP: 10100110
After inverse permutation (IP-1): 01101001
```



```
--- Decrypting block 11111001 ---
After P10 permutation: 1111001101
Shifted halves (1st): 11101 | 11010
Key-1: 11100101
Shifted halves (2nd): 10111 | 01011
Key-2: 01110111
After initial permutation (IP): 10111110
Initial L = 1011, R = 1110
After expansion/permutation (EP): 01111101
After XOR with key: 00001010
S-box output: 0100
After P4 permutation: 1000
After XOR with left: 0011
After swap: L = 1110, R = 0011
After expansion/permutation (EP): 10010110
After XOR with key: 01110011
S-box output: 0000
After P4 permutation: 0000
After XOR with left: 1110
Before inverse IP: 11100011
After inverse permutation (IP-1): 01101110
Cipher Binary: 1011111111001001000010011010001011111001
Decrypted Text: Bewin
```

Result

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