TASK - 1 : Toy Block Cipher

```
#defining the s-box and p-box

sBox = {
    0: 14, 1: 4, 2: 13, 3: 1,
    4: 2, 5: 15, 6: 11, 7: 8,
    8: 3, 9: 10, 10: 6, 11: 12,
    12: 5, 13: 9, 14: 0, 15: 7
}
inverse_sBox = {
    14: 0, 4: 1, 13: 2, 1: 3,
    2: 4, 15: 5, 11: 6, 8: 7,
    3: 8, 10: 9, 6: 10, 12: 11,
    5: 12, 9: 13, 0: 14, 7: 15
}

pBox = [15, 0, 8, 7, 12, 3, 9, 5, 6, 1, 10, 4, 2, 14, 13, 11]
inverse_pBox = [1, 9, 12, 5, 11, 7, 8, 3, 2, 6, 10, 15, 4, 14, 13, 0]
```

```
plainText = format(0b1101001010110110, '016b')

"""converts the plainText to the correspoding
   binary , b means binary and 016 means 16 bit ,
   if they are not filled they will add zero till 16 bit"""

key = format(0b0110110110010100, '016b')  #same as the plainText

def apply_sBox(block, sBox):
    substituted = ""
    for i in range(0, 16, 4):
        index = int(block[i:i+4], 2)  # Convert 4-bit binary string to integer
        substituted += format(sBox[index], '04b')  # Convert back to 4-bit binary
    return substituted

def apply_XOR(block, key):
    result = ""
    for i in range(16):
        result += str(int(block[i]) ^ int(key[i]))  # XOR operation
    return result

def apply_pBox(block, pBox):
    permuted = ""  # Start with an empty string
    for i in pBox:  # Loop through each index in pBox
        permuted += block[i]  # Get the bit at position i and add it to permuted
    return permuted  # Return the rearranged binary string


# Apply transformations
xor_result = apply_XOR(plainText, key) # plaintext XOR Key
substituted_result = apply_sBox(xor_result, sBox) # applying s-box
ciphertext = apply_pBox(substituted_result, pBox) # applying permutations

print("ENCRYPTION")
print("Plaintext: ", plainText)
print("Ciphertext:", ciphertext)

pbox_reversed = apply_pBox(ciphertext, inverse_pBox)  # Reverse P-Box
sbox_reversed = apply_sBox(pbox_reversed, inverse_sBox)  # Reverse S-Box
decrypted_text = apply_XOR(sbox_reversed, key)  # Reverse XOR

print()

print("DECRYPTION")
print("Ciphertext:", ciphertext)
print("Plaintext:  ", decrypted_text)
```

Task 2 : ECB

```
def ecb_encrypt(plaintext, key):
    ciphertext = ""
    for c in plaintext:
        ciphertext += chr(ord(c) + key)  # Shift each character
    return ciphertext
```

```
def ecb_decrypt(ciphertext, key):
    decrypted_text = ""
    for c in ciphertext:
        decrypted_text += chr(ord(c) - key)  # Reverse shift
    return decrypted_text.rstrip()  # Remove padding spaces

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

# Pad the input to be a multiple of 16
while len(plaintext) % 16 != 0:
    plaintext += " "  # Adding spaces as padding

key = 1  # Simple shift key

ciphertext = ecb_encrypt(plaintext, key)
print("Ciphertext:", ciphertext)

decrypted_text = ecb_decrypt(ciphertext, key)
print("Decrypted:", decrypted_text)
```

Task 3 - CBC Mode

```
def get_valid_16bit_input(prompt):
    value = input(prompt)
    while len(value) != 16:
        valid = True  # Assume input is valid
        for c in value:
            if c != '0' and c != '1':  # Check if input contains only 0s and 1s
                valid = False
                break  # Stop checking further

        if not valid:
            value = input("Invalid input! " + prompt)  # Ask again
    return value

plainText = get_valid_16bit_input("Enter a 16-bit binary plaintext: ")
key = get_valid_16bit_input("Enter a 16-bit binary key: ")
IV = get_valid_16bit_input("Enter a 16-bit binary IV: ")

def apply_sBox(block, sBox):
    substituted = ""
    for i in range(0, 16, 4):
        index = int(block[i:i+4], 2)
        substituted += format(sBox[index], '04b')
    return substituted

def apply_XOR(block1, block2):
    result = ""
    for i in range(16):
        bit1 = int(block1[i])  # Convert character '0' or '1' to integer (0 or 1)
        bit2 = int(block2[i])  # Convert character '0' or '1' to integer (0 or 1)
        xor_bit = bit1 ^ bit2   # Perform XOR operation (1 if bits are different, 0 if same)
        result += str(xor_bit)  # Convert XOR result back to string and add to result

    return result  # Return the final XOR result (16-bit binary string)


def apply_pBox(block, pBox):
    permuted = ""
    for i in pBox:
        permuted += block[i]
    return permuted  # Return the final permuted 16-bit binary string

def cbc_encrypt(plaintext, key, IV, sBox, pBox):
    ciphertext_blocks = []
    prev_ciphertext = IV

    for i in range(0, len(plaintext), 16):
        block = plaintext[i:i+16]

        xor_result = apply_XOR(block, prev_ciphertext)
        substituted = apply_sBox(xor_result, sBox)
        ciphertext = apply_pBox(substituted, pBox)

        ciphertext_blocks.append(ciphertext)
        prev_ciphertext = ciphertext

    return "".join(ciphertext_blocks)
```

```
def cbc_decrypt(ciphertext, key, IV, inverse_sBox, inverse_pBox):
    decrypted_blocks = []
    prev_ciphertext = IV

    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i+16]

        pbox_reversed = apply_pBox(block, inverse_pBox)
        sbox_reversed = apply_sBox(pbox_reversed, inverse_sBox)
        decrypted_block = apply_XOR(sbox_reversed, prev_ciphertext)

        decrypted_blocks.append(decrypted_block)
        prev_ciphertext = block

    return "".join(decrypted_blocks)

ciphertext = cbc_encrypt(plainText, key, IV, sBox, pBox)
print("Ciphertext:", ciphertext)

decrypted_text = cbc_decrypt(ciphertext, key, IV, inverse_sBox, inverse_pBox)
print("Decrypted:", decrypted_text)
```

```
Enter a 16-bit binary plaintext: 1111100000101010
Enter a 16-bit binary key: 1111010101000001
Enter a 16-bit binary IV: 1010111010100101
Ciphertext: 1101010011111111
Decrypted: 1111100000101010
```

Task 4 - CFB Mode

```
sBox = {
    0: 14, 1: 4, 2: 13, 3: 1,
    4: 2, 5: 15, 6: 11, 7: 8,
    8: 3, 9: 10, 10: 6, 11: 12,
    12: 5, 13: 9, 14: 0, 15: 7
}
pBox = [15, 0, 8, 7, 12, 3, 9, 5, 6, 1, 10, 4, 2, 14, 13, 11]

def get_valid_16bit_input(prompt):
    value = input(prompt)
    while len(value) != 16 or any(c not in '01' for c in value):
        value = input("Invalid input! " + prompt)
    return value

plainText = get_valid_16bit_input("Enter a 16-bit binary plaintext: ")
key = get_valid_16bit_input("Enter a 16-bit binary key: ")
IV = get_valid_16bit_input("Enter a 16-bit binary IV: ")

def apply_sBox(block, sBox):
    substituted = ""
    for i in range(0, 16, 4):
        index = int(block[i:i+4], 2)
        substituted += format(sBox[index], '04b')
    return substituted

def apply_XOR(block1, block2):
    return "".join(str(int(block1[i]) ^ int(block2[i])) for i in range(16))

def apply_pBox(block, pBox):
    return "".join(block[i] for i in pBox)

def cfb_encrypt(plaintext, key, IV, sBox, pBox):
    ciphertext_blocks = []
    prev_ciphertext = IV

    for i in range(0, len(plaintext), 16):
        block = plaintext[i:i+16]

        encrypted_IV = apply_pBox(apply_sBox(prev_ciphertext, sBox), pBox)
        ciphertext = apply_XOR(block, encrypted_IV)

        ciphertext_blocks.append(ciphertext)
        prev_ciphertext = ciphertext

    return "".join(ciphertext_blocks)

def cfb_decrypt(ciphertext, key, IV, sBox, pBox):
    decrypted_blocks = []
    prev_ciphertext = IV
    for i in range(0, len(ciphertext), 16):
        block = ciphertext[i:i+16]

        encrypted_IV = apply_pBox(apply_sBox(prev_ciphertext, sBox), pBox)
        decrypted_block = apply_XOR(block, encrypted_IV)

        decrypted_blocks.append(decrypted_block)
        prev_ciphertext = block

    return "".join(decrypted_blocks)

ciphertext = cfb_encrypt(plainText, key, IV, sBox, pBox)
print("Ciphertext:", ciphertext)

decrypted_text = cfb_decrypt(ciphertext, key, IV, sBox, pBox)
print("Decrypted:", decrypted_text)
```

```
Enter a 16-bit binary plaintext: 1010110101010110
Enter a 16-bit binary key: 1111000010101010
Enter a 16-bit binary IV: 1111101010101010
Ciphertext: 1010101010111000
Decrypted: 1010110101010110
```