VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CRYPTOGRAPHY AND NETWORK SECURITY

LAB 4: WRITE A CODE FOR IMPLEMENTING SDES IN SUITABLE PROGRAMMING LANGUAGE.

NAME : OM SUBRATO DEY

REG NO.: 21BAI1876

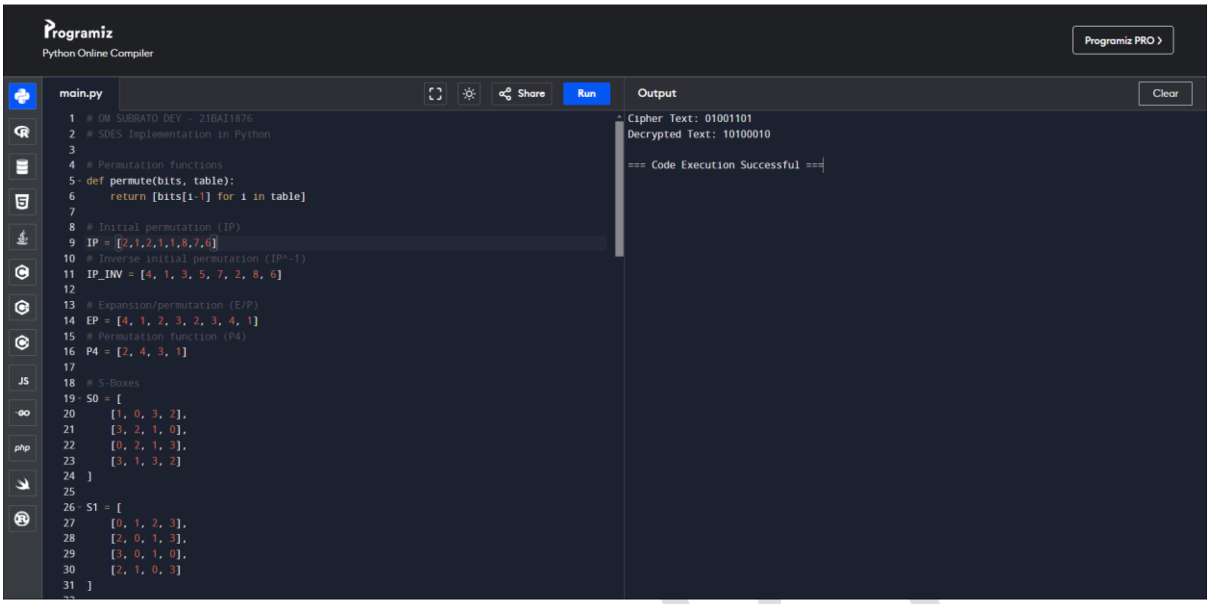
CODE:

```
# OM SUBRATO DEY - 21BAI1876

# SDES Implementation in Python


# Permutation functions

def permute(bits, table):

    return [bits[i-1] for i in table]


# Initial permutation (IP)

IP = [2,1,2,1,1,8,7,6]

# Inverse initial permutation (IP^-1)

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


# Expansion/permutation (E/P)

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

# Permutation function (P4)

P4 = [2, 4, 3, 1]


# S-Boxes

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]

]
```

```
def sbox_lookup(sbox, row, col):
    return sbox[row][col]

def fk(bits, key):
    left, right = bits[:4], bits[4:]
    expanded_permuted = permute(right, EP)
    xor_result = [a ^ b for a, b in zip(expanded_permuted, key)]

    left_half = xor_result[:4]
    right_half = xor_result[4:]

    row0 = (left_half[0] << 1) + left_half[3]
    col0 = (left_half[1] << 1) + left_half[2]
    s0_output = sbox_lookup(S0, row0, col0)

    row1 = (right_half[0] << 1) + right_half[3]
    col1 = (right_half[1] << 1) + right_half[2]
    s1_output = sbox_lookup(S1, row1, col1)

    sbox_output = [
        (s0_output & 0b10) >> 1, s0_output & 0b01,
        (s1_output & 0b10) >> 1, s1_output & 0b01
    ]

    permuted_sbox_output = permute(sbox_output, P4)
    left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]

    return left_result + right

def switch(bits):
    return bits[4:] + bits[:4]

# Key generation
def generate_keys(key):
    P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    P8 = [6, 3, 7, 4, 8, 5, 10, 9]

    permuted_key = permute(key, P10)
    left, right = permuted_key[:5], permuted_key[5:]

    left = left[1:] + left[:1]
    right = right[1:] + right[:1]
    k1 = permute(left + right, P8)

    left = left[2:] + left[:2]
    right = right[2:] + right[:2]
    k2 = permute(left + right, P8)

    return k1, k2

# Encryption and decryption
def sdes_encrypt(plain_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(plain_text, IP)
    round1_result = fk(initial_permuted, k1)
    switched = switch(round1_result)
    round2_result = fk(switched, k2)
    cipher_text = permute(round2_result, IP_INV)
    return cipher_text

def sdes_decrypt(cipher_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(cipher_text, IP)
    round1_result = fk(initial_permuted, k2)
    switched = switch(round1_result)
    round2_result = fk(switched, k1)
    plain_text = permute(round2_result, IP_INV)
    return plain_text
```

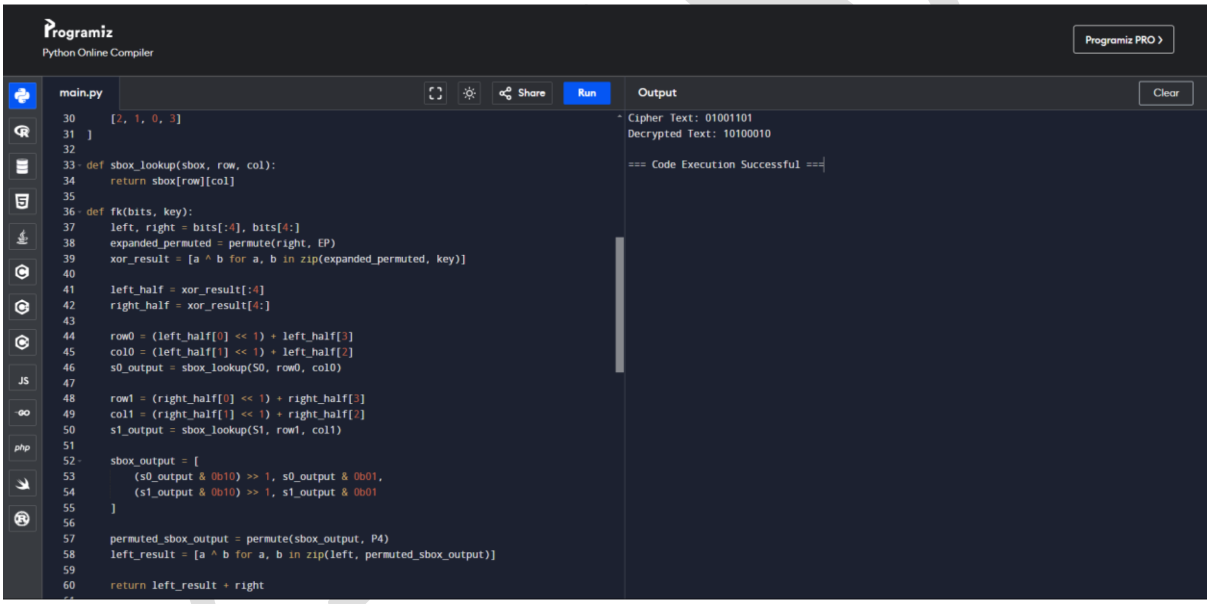
```
# Helper functions to convert between binary strings and lists
def string_to_bits(s):
    return [int(b) for b in s]

def bits_to_string(bits):
    return ''.join(str(b) for b in bits)

# Example usage
plain_text = "10101010"
key = "1010000010"

plain_bits = string_to_bits(plain_text)
key_bits = string_to_bits(key)

cipher_bits = sdes_encrypt(plain_bits, key_bits)
cipher_text = bits_to_string(cipher_bits)

print(f"Cipher Text: {cipher_text}")

decrypted_bits = sdes_decrypt(cipher_bits, key_bits)
decrypted_text = bits_to_string(decrypted_bits)

print(f"Decrypted Text: {decrypted_text}")
```

OUTPUT:



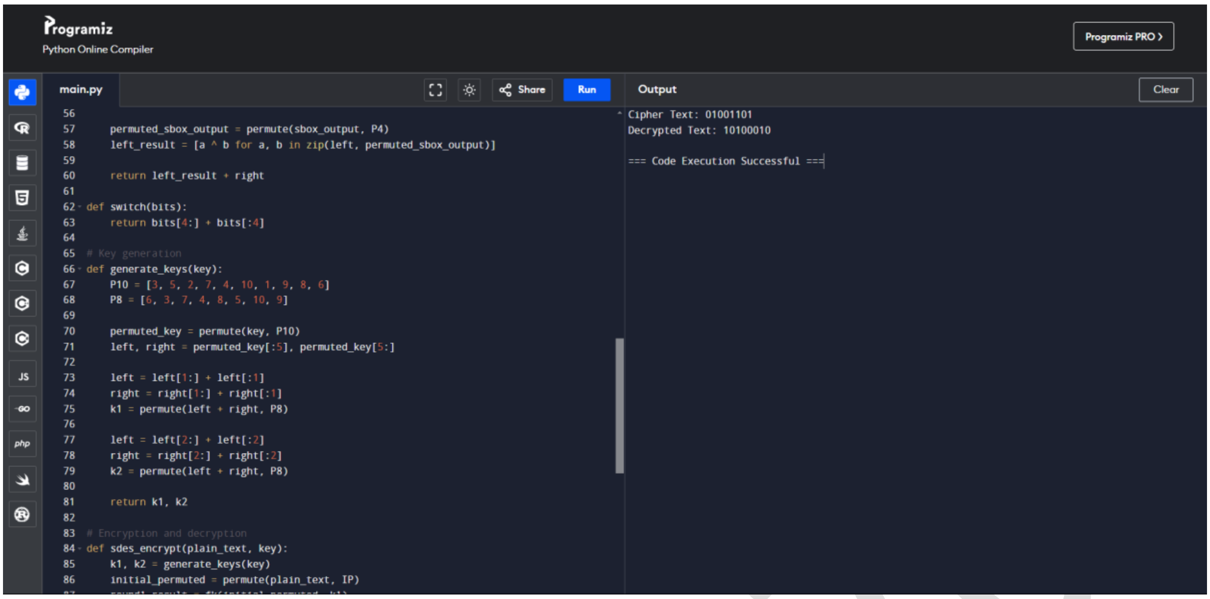
```python
# OM SUBRATO DEY - 21BAI1876
# SDES Implementation in Python

# Permutation functions
def permute(bits, table):
    return [bits[i-1] for i in table]

# Initial permutation (IP)
IP = [2,1,2,1,1,8,7,6]
# Inverse initial permutation (IP^-1)
IP_INV = [4, 1, 3, 5, 7, 2, 8, 6]

# Expansion/permutation (E/P)
EP = [4, 1, 2, 3, 2, 3, 4, 1]
# Permutation function (P4)
P4 = [2, 4, 3, 1]

# S-Boxes
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]
]
```

```
Cipher Text: 01001101
Decrypted Text: 10100010

=== Code Execution Successful ===
```



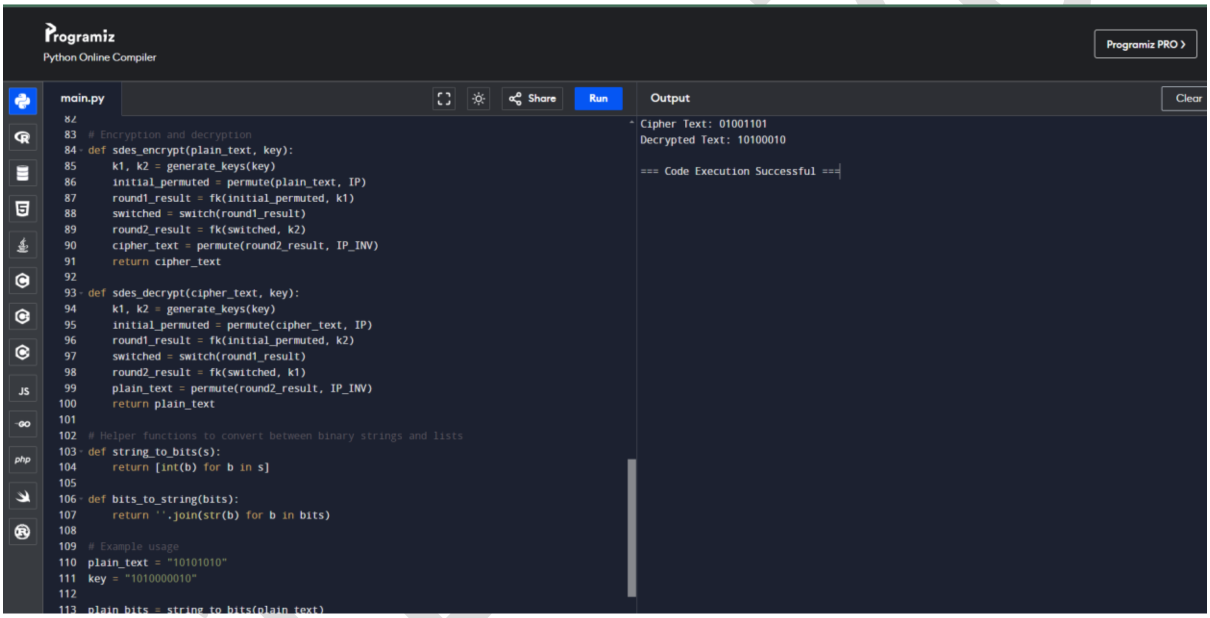
```python
    [2, 1, 0, 3]
]

def sbox_lookup(sbox, row, col):
    return sbox[row][col]

def fk(bits, key):
    left, right = bits[:4], bits[4:]
    expanded_permuted = permute(right, EP)
    xor_result = [a ^ b for a, b in zip(expanded_permuted, key)]

    left_half = xor_result[:4]
    right_half = xor_result[4:]

    row0 = (left_half[0] << 1) + left_half[3]
    col0 = (left_half[1] << 1) + left_half[2]
    s0_output = sbox_lookup(S0, row0, col0)

    row1 = (right_half[0] << 1) + right_half[3]
    col1 = (right_half[1] << 1) + right_half[2]
    s1_output = sbox_lookup(S1, row1, col1)

    sbox_output = [
        (s0_output & 0b10) >> 1, s0_output & 0b01,
        (s1_output & 0b10) >> 1, s1_output & 0b01
    ]

    permuted_sbox_output = permute(sbox_output, P4)
    left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]

    return left_result + right
```

```
Cipher Text: 01001101
Decrypted Text: 10100010

=== Code Execution Successful ===
```



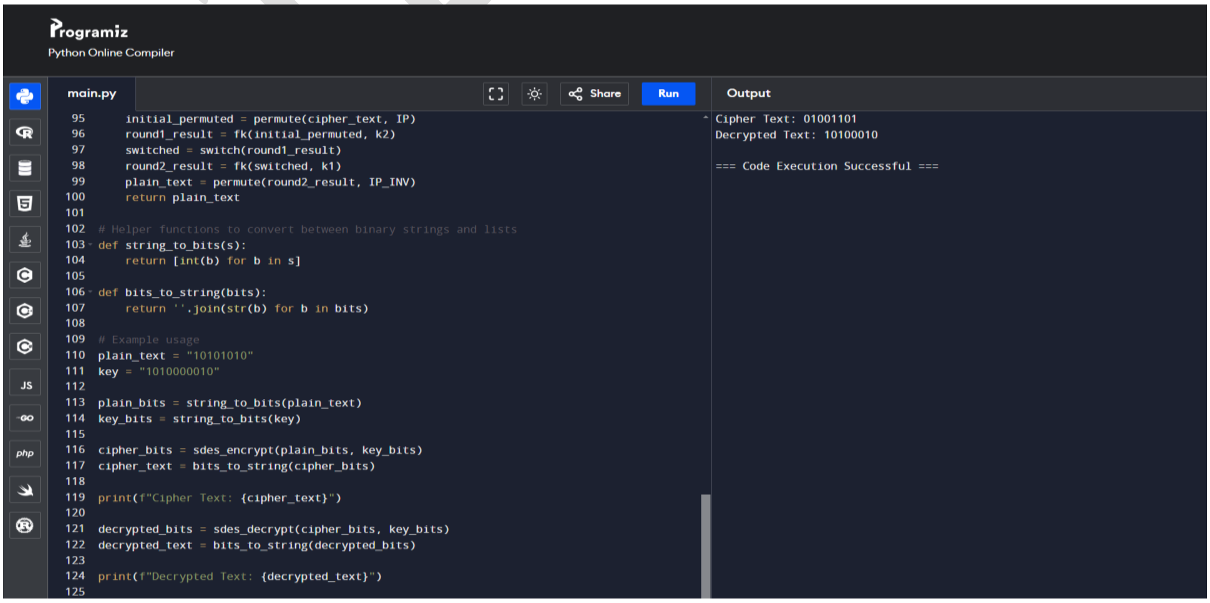
```python

    permuted_sbox_output = permute(sbox_output, P4)
    left_result = [a ^ b for a, b in zip(left, permuted_sbox_output)]

    return left_result + right

def switch(bits):
    return bits[4:] + bits[:4]

# Key generation
def generate_keys(key):
    P10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]
    P8 = [6, 3, 7, 4, 8, 5, 10, 9]

    permuted_key = permute(key, P10)
    left, right = permuted_key[:5], permuted_key[5:]

    left = left[1:] + left[:1]
    right = right[1:] + right[:1]
    k1 = permute(left + right, P8)

    left = left[2:] + left[:2]
    right = right[2:] + right[:2]
    k2 = permute(left + right, P8)

    return k1, k2

# Encryption and decryption
def sdes_encrypt(plain_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(plain_text, IP)
    round1_result = fk(initial_permuted, k1)
```

```
Cipher Text: 01001101
Decrypted Text: 10100010

=== Code Execution Successful ===
```



```python

# Encryption and decryption
def sdes_encrypt(plain_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(plain_text, IP)
    round1_result = fk(initial_permuted, k1)
    switched = switch(round1_result)
    round2_result = fk(switched, k2)
    cipher_text = permute(round2_result, IP_INV)
    return cipher_text

def sdes_decrypt(cipher_text, key):
    k1, k2 = generate_keys(key)
    initial_permuted = permute(cipher_text, IP)
    round1_result = fk(initial_permuted, k2)
    switched = switch(round1_result)
    round2_result = fk(switched, k1)
    plain_text = permute(round2_result, IP_INV)
    return plain_text

# Helper functions to convert between binary strings and lists
def string_to_bits(s):
    return [int(b) for b in s]

def bits_to_string(bits):
    return ''.join(str(b) for b in bits)

# Example usage
plain_text = "10101010"
key = "1010000010"

plain_bits = string_to_bits(plain_text)
```

```
Cipher Text: 01001101
Decrypted Text: 10100010

=== Code Execution Successful ===
```



```python
    initial_permuted = permute(cipher_text, IP)
    round1_result = fk(initial_permuted, k2)
    switched = switch(round1_result)
    round2_result = fk(switched, k1)
    plain_text = permute(round2_result, IP_INV)
    return plain_text

# Helper functions to convert between binary strings and lists
def string_to_bits(s):
    return [int(b) for b in s]

def bits_to_string(bits):
    return ''.join(str(b) for b in bits)

# Example usage
plain_text = "10101010"
key = "1010000010"

plain_bits = string_to_bits(plain_text)
key_bits = string_to_bits(key)

cipher_bits = sdes_encrypt(plain_bits, key_bits)
cipher_text = bits_to_string(cipher_bits)

print(f"Cipher Text: {cipher_text}")

decrypted_bits = sdes_decrypt(cipher_bits, key_bits)
decrypted_text = bits_to_string(decrypted_bits)

print(f"Decrypted Text: {decrypted_text}")

```

```
Cipher Text: 01001101
Decrypted Text: 10100010

=== Code Execution Successful ===
```