P10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)

P8 = (6, 3, 7, 4, 8, 5, 10, 9)

P4 = (2, 4, 3, 1)

IP = (2, 6, 3, 1, 4, 8, 5, 7)

IPi = (4, 1, 3, 5, 7, 2, 8, 6)

E = (4, 1, 2, 3, 2, 3, 4, 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]

]

def permutation(pattern, key):

permuted = ""

for i in pattern:

permuted += key[i-1]

return permuted

def generate\_first(left, right):

left = left[1:] + left[:1] # 5-bit left & right. Now last 4 bits selected & first bit added at last. result 5 bit only.

right = right[1:] + right[:1]

key = left + right #left & right added. result 10 bit key. But we need 8 bit key. so, permutate using P8.

return permutation(P8, key)

def generate\_second(left, right):

left = left[3:] + left[:3] # 5-bit left & right. Now last 2 bits selected & first 3 bit added at last. result 5 bit only.

right = right[3:] + right[:3]

key = left + right

return permutation(P8, key) #But we need 8 bit key. so, permutate using P8.

def transform(right, key):

extended = permutation(E, right)

xor\_cipher = bin(int(extended, 2) ^ int(key, 2))[2:].zfill(8)

xor\_left = xor\_cipher[:4]

xor\_right = xor\_cipher[4:]

print("Xor left",xor\_left)

print("Xor right",xor\_right)

new\_left = Sbox(xor\_left, S0)

new\_right = Sbox(xor\_right, S1)

print(new\_left[2:],new\_right[2:])

#print("cipher from fk1",permutation(P4, new\_left[2:] + new\_right[2:]))

return permutation(P4, new\_left[2:] + new\_right[2:])

def Sbox(data, box):

row = int(data[0] + data[3], 2) #We take the first and fourth bit as row and the second and third bit as a column for our S boxes.

column = int(data[1] + data[2], 2)

return bin(box[row][column])[2:].zfill(4)

def encrypt(left, right, key): #int(string, Base). Converts into integer. Here binary numbers needed to convert into int. So, base 2

cipher = int(left, 2) ^ int(transform(right, key), 2)

return right, bin(cipher)[2:].zfill(4) #if length not 4 add zeros at start.

def decrypt(left, right, key): #int(string, Base). Converts into integer. Here binary numbers needed to convert into int. So, base 2

plain = int(left, 2) ^ int(transform(right, key), 2)

return right, bin(plain)[2:].zfill(4)

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

if len(key) != 10:

raise Exception("Check the input")

plaintext = input("Enter 8-bit plaintext: ")

if len(plaintext) != 8:

raise Exception("Check the input")

#key generation

p10key = permutation(P10, key) #we need result 10 bit key, so permutate using P10.

print("First Permutation", p10key)

left\_key = p10key[:len(p10key)//2] #why //2 --> to get middle element even in case of odd.

print("Left key",left\_key)

right\_key = p10key[len(p10key)//2:]

print("Right key",right\_key)

first\_key = generate\_first(left\_key, right\_key)

print("\*\*\*\*\*")

print("First key:", first\_key)

second\_key = generate\_second(left\_key, right\_key)

print("Second key", second\_key)

print("\*\*\*\*\*")

#Encryption

initial\_permutation = permutation(IP, plaintext)

print("Initial Permutation",initial\_permutation)

left\_data = initial\_permutation[:len(initial\_permutation)//2]

right\_data = initial\_permutation[len(initial\_permutation)//2:]

print("Left data",left\_data)

left, right = encrypt(left\_data, right\_data, first\_key)

print("After 1st round",left,right)

left, right = encrypt(left, right, second\_key)

print(left,right)

ciphertext=permutation(IPi, right+left)

print("Ciphertext:", ciphertext )

initial\_permutation = permutation(IP, ciphertext)

print("Initial Permutation",initial\_permutation)

left\_data = initial\_permutation[:len(initial\_permutation)//2]

right\_data = initial\_permutation[len(initial\_permutation)//2:]

print("Left data",left\_data)

left, right = decrypt(left\_data, right\_data, second\_key)

print("After 1st round",left,right)

left, right = decrypt(left, right, first\_key)

print(left,right)

newplaintext=permutation(IPi, right+left)

print("After decryption plaintext:", newplaintext )