```
P10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)     #p10 table
P8 = (6, 3, 7, 4, 8, 5, 10, 9)            #p8 table
P4 = (2, 4, 3, 1)                         #p4 table

IP = (2, 6, 3, 1, 4, 8, 5, 7)             #initial Permutation
IPi = (4, 1, 3, 5, 7, 2, 8, 6)            #inverse Permutation

E = (4, 1, 2, 3, 2, 3, 4, 1)              #from 4 to 8 bit conversion

S0 = [                                    #s0 table
       [1, 0, 3, 2],
       [3, 2, 1, 0],
       [0, 2, 1, 3],
       [3, 1, 3, 2]
    ]

S1 = [                                    #s1 table
       [0, 1, 2, 3],
       [2, 0, 1, 3],
       [3, 0, 1, 0],
       [2, 1, 0, 3]
    ]

''' all above data needs to be learned/ needed in advance '''

def permutation(perm, key):                       #generic permutation function
    permutated_key = ""
    for i in perm:
        permutated_key += key[i-1]
    return permutated_key                         #returns a string

def generate_first_key(left_key, right_key): #only for first string
    left_key_rot = left_key[1:] + left_key[:1]
    right_key_rot = right_key[1:] + right_key[:1]
    print("Key rotation: ",left_key_rot,"|",right_key_rot)
    key_rot = left_key_rot + right_key_rot
    return permutation(P8, key_rot)

def generate_second_key(left_key, right_key):     #only for second string
    left_key_rot = left_key[3:] + left_key[:3]
    right_key_rot = right_key[3:] + right_key[:3]
    key_rot = left_key_rot + right_key_rot
    return permutation(P8, key_rot)

def F(right, subkey):                             #used for xor
    expanded_cipher = permutation(E, right)
    xor_cipher = bin( int(expanded_cipher, 2) ^ int(subkey, 2) )[2:].zfill(8)
    left_xor_cipher = xor_cipher[:4]
    right_xor_cipher = xor_cipher[4:]
    left_sbox_cipher = Sbox(left_xor_cipher, S0)
    right_sbox_cipher = Sbox(right_xor_cipher, S1)
    return permutation(P4, left_sbox_cipher + right_sbox_cipher)
```

```
def Sbox(myinput, sbox):      #searching in 2D array
    row = int(myinput[0] + myinput[3], 2)
    column = int(myinput[1] + myinput[2], 2)
    return bin(sbox[row][column])[2:].zfill(4)

def f(first_half, second_half, key):
     left = int(first_half, 2) ^ int(F(second_half, key), 2)
     print ("Fk: " , bin(left)[2:].zfill(4) ,"", second_half)
     return bin(left)[2:].zfill(4), second_half


key = "0111111101"              #KEY for decryption
print("Key: ",key)

p10key = permutation(P10, key)
left = p10key[:int(len(p10key)/2)]
right = p10key[int(len(p10key)/2):]
print("Permuted key: ",left,"|",right)

first_key = generate_first_key(left, right)
second_key = generate_second_key(left, right)
print ("[*] First key:(after p8) " + first_key)
print ("[*] Second key:(after p8) " + second_key)

cipher = "10100010"             #text to be decrypted
print("\nCipher: ",cipher)

permutated_cipher = permutation(IP, cipher)
print ("IP: " + permutated_cipher)
first_half_cipher = permutated_cipher[:int(len(permutated_cipher)/2)]
second_half_cipher = permutated_cipher[int(len(permutated_cipher)/2):]

left, right = f(first_half_cipher, second_half_cipher, second_key)
print ("SW: " + right + left)
left, right = f(right, left, first_key)     # switch left and right!
print ("IP^-1: " + permutation(IPi, left + right))
'''
```

OUTPUT:

```
Key:  0111111101
Permuted key:  11111 | 10011
Key rotation:  11111 | 00111
[*] First key:(after p8) 01011111
[*] Second key:(after p8) 11111100

Cipher:  10100010
IP: 00110001
Fk:  0011  0001
SW: 00010011
Fk:  0101  0011
IP^-1: 10001110
 '''
```