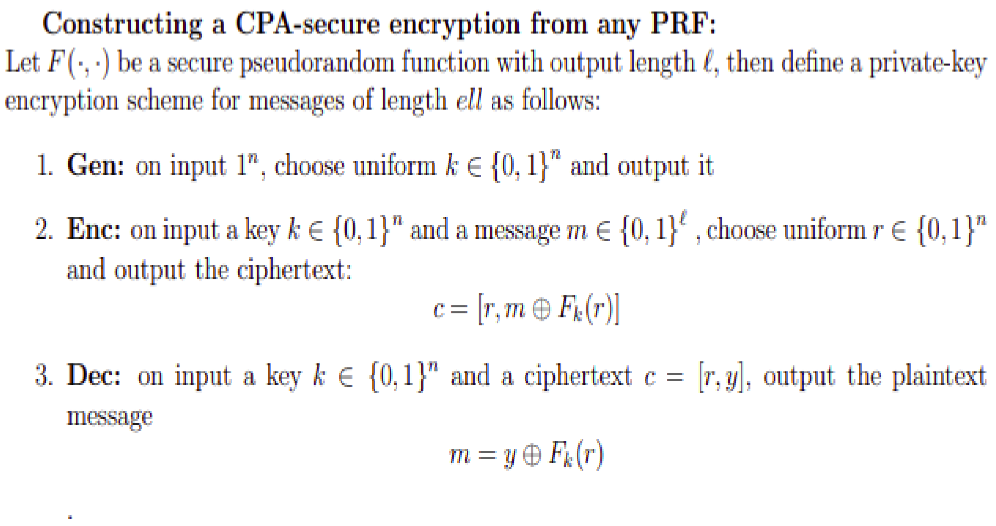
POIS ASSIGNMENT 1

TASK 3

USE THE PRF TO OBTAIN A CPA-SECURE ENCRYPTION SCHEME

CODE EXPLANATION



Constructing a CPA-secure encryption from any PRF:

Let $F(\cdot,\cdot)$ be a secure pseudorandom function with output length ℓ, then define a private-key encryption scheme for messages of length ell as follows:

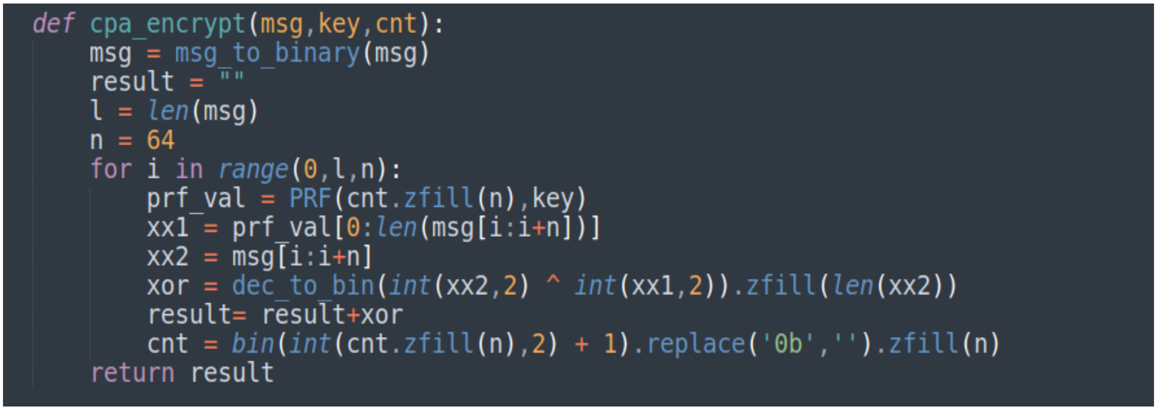
1. **Gen:** on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it
2. **Enc:** on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^\ell$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext:
$$c = [r, m \oplus F_k(r)]$$
3. **Dec:** on input a key $k \in \{0,1\}^n$ and a ciphertext $c = [r,y]$, output the plaintext message
$$m = y \oplus F_k(r)$$

.

COUNTER MODE :

In the randomized counter mode of operation for block ciphers :
We begin by choosing a random IV. Then, we encrypt the message by encrypting each plaintext block i with F(k, IV +i): m[i]⊕F(k, IV +i).

cpa_encrypt FUNCTION :

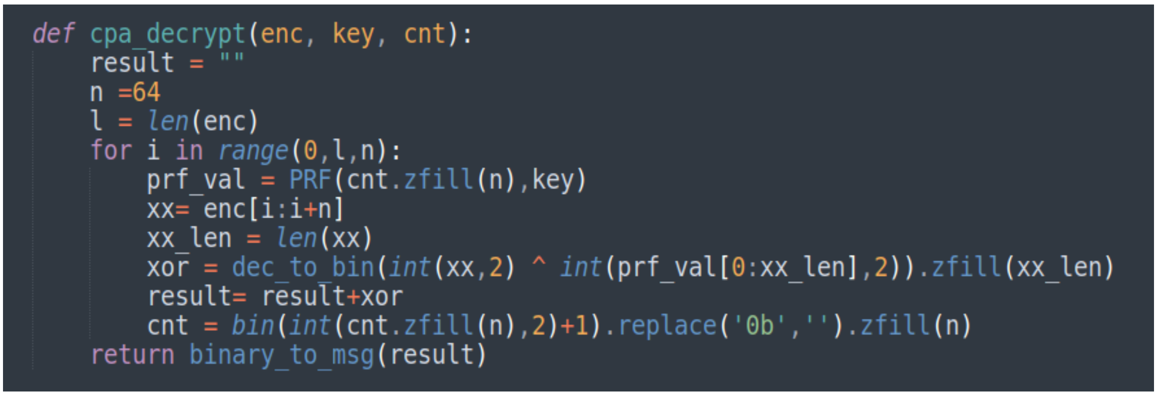


```python
def cpa_encrypt(msg,key,cnt):
    msg = msg_to_binary(msg)
    result = ""
    l = len(msg)
    n = 64
    for i in range(0,l,n):
        prf_val = PRF(cnt.zfill(n),key)
        xx1 = prf_val[0:len(msg[i:i+n])]
        xx2 = msg[i:i+n]
        xor = dec_to_bin(int(xx2,2) ^ int(xx1,2)).zfill(len(xx2))
        result= result+xor
        cnt = bin(int(cnt.zfill(n),2) + 1).replace('0b','').zfill(n)
    return result
```

The message is converted to binary value. To encrypt the message, we have to each message block of size n = 64. PRF value is generated and equal to the length of the msg block, is chosen for further use. As stated in the above counter mode description, random IV has been chosen and formula is applied the (msg-block xor iv value(prf value)). This xor value is

added to the variable result. Count variable (cnt) is being updated again for a new prf function in the next block iteration.
The encrypted msg (result) is being returned in the end.

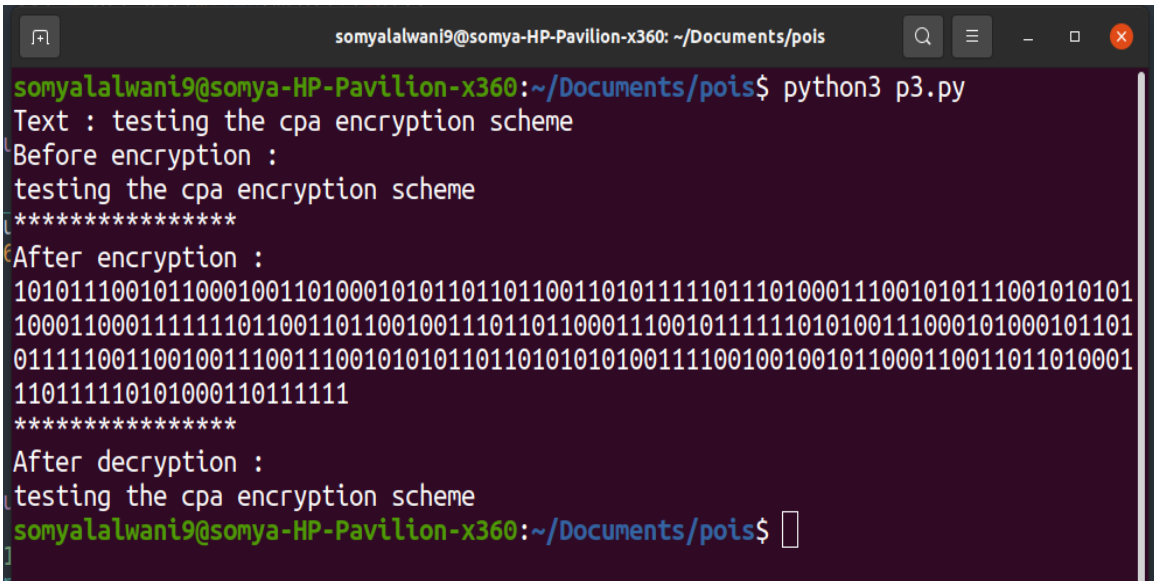
cpa_decrypt FUNCTION :



```python
def cpa_decrypt(enc, key, cnt):
    result = ""
    n =64
    l = len(enc)
    for i in range(0,l,n):
        prf_val = PRF(cnt.zfill(n),key)
        xx= enc[i:i+n]
        xx_len = len(xx)
        xor = dec_to_bin(int(xx,2) ^ int(prf_val[0:xx_len],2)).zfill(xx_len)
        result= result+xor
        cnt = bin(int(cnt.zfill(n),2)+1).replace('0b','').zfill(n)
    return binary_to_msg(result)
```

The same goes for cpa_decrypt function. The only change is instead of xoring the msg block with the prf value, now the prf value will be xor with the encrypted msg block recvd from the cpa_encrypt.

OUTPUT:



```
somyalalwani9@somya-HP-Pavilion-x360:~/Documents/pois$ python3 p3.py
Text : testing the cpa encryption scheme
Before encryption :
testing the cpa encryption scheme
****************
After encryption :
1010111001011000100110100010101101101100110101111101110100011100101011100101010
1000110001111111011001101100100111011011000111001011111101010011100010100010110
1011111001100100111001110010101011011010101010011110010010010110001100110110100
01
1101111101010001101111111
****************
After decryption :
testing the cpa encryption scheme
somyalalwani9@somya-HP-Pavilion-x360:~/Documents/pois$ 
```