INITIALISATION

In [22]:

```
import random
def get_random_permutation(n):
    out = [i+1 for i in range(n)]
    random.shuffle(out)
    return out
```

In [23]:

```
IP=get_random_permutation(8)
IP_INV=[1]*8

for i in range(len(IP)):
    IP_INV[IP[i]-1]=i+1

print("IP",IP)
print("IP_INV",IP_INV)
```

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

In [24]:

```
EP=get_random_permutation(4)*2
print(EP)
```

```
[2, 4, 1, 3, 2, 4, 1, 3]
```

In [27]:

```
P10=get_random_permutation(10)
print(P8)
```

```
[1, 2, 6, 3, 7, 4, 10, 9]
```

In [28]:

```
P8=random.sample(get_random_permutation(10),8)
print(P8)
```

```
[7, 5, 2, 10, 8, 9, 1, 6]
```

Helper functions

In [29]:

```
def bin_to_dec(x):
    return int(x,2)

def dec_to_bin(x):
    return bin(x).replace("0b","")
```

In [30]:

```


def permutate(key, perm):
  ret = ""
  for k in perm:
    ret += key[k-1]
  return ret
```

In [31]:

```
def xor(a,b):
    ret=""
    for i in range(len(a)):
        if a[i]==b[i]:
            ret+="0"
        else:
            ret+="1"
    return ret
```

In [32]:

```
def left_circular_shift(x,shift=1):
    shifts=shifts%len(x)
    return x[shifts:]+x[:shifts]
```

In [33]:

```
def split_str(key):
    half=len(key)//2
    key1=key[:half]
    key2=key[half:]
    return key1,key2
```

In [34]:

```
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]
]
```

Functions

In [35]:

```
def getkeys(key):
    n_key=permutate(key,P10)
    left_key,right_key=split_str(n_key)

    left_key=left_circular_shift(left_key,1)
    right_key=left_circular_shift(right_key,1)

    k1=permutate(left_key+right_key,P8)
    left_key=left_circular_shift(left_key,2)
    right_key=left_circular_shift(right_key,2)

    k2=permutate(left_key+right_key,P8)

    return k1,k2
```

In [36]:

```
def s_box(text,s):
    r=text[0]+text[3]
    c=text[1]+text[2]

    r=bin_to_dec(r)
    c=bin_to_dec(c)

    out=s[r][c]
    out=dec_to_bin(out)

    if len(out) < 2:
        out="0"+out
    return out
```

In [37]:

```
def function(left,right,subkey):
    text=right
    text=permutate(text,EP)
    text=xor(text,subkey)
    text_left,text_right=split_str(text)
    text=s_box(text_left,s0)+s_box(text_right,s1)
    text=xor(text,left)
    return text,right
```

In [38]:

```
def encryption(plaintext,key):
    k1,k2=getkeys(key)
    ciphertext=permutate(plaintext,IP)
    left,right=split_str(ciphertext)

    left_right=function(left,right,k1)
    left,right=right,left
    left_right=function(left,right,k2)

    ciphertext=permutate(left+right,IP_INV)
    return ciphertext


def decryption(ciphertext,key):
    k1,k2=getkeys(key)
    plaintext=permutate(ciphertext,IP)
    left,right=split_str(plaintext)

    left_right=function(left,right,k2)
    left,right=right,left
    left_right=function(left,right,k1)

    plaintext=permutate(left+right,IP_INV)
    return plaintext
```

In [ ]:

```
key = "1010001011"
plaintext = "10001010"
```

In [ ]:

```
c = encryption(plaintext, key)
print(c)
p = decryption(c, key)
print(p)
```

In [ ]:

```
##RSA
```

In [ ]:

```
import random

def generate_key():
    p=random.randint(2,999)
    q=random.randint(2,999)

    flag1=isprime(q)
    flag2=isprime(p)

    if flag1==0 and flag2==0:
        RSA(p,q)
    else:
        generate_key()


def isprime(num):
    flag=0
    r=int(num/2)
    for i in range(2,r):
        if(num%i==0):
            flag=1
            break


    return flag

def RSA(a,b):
    print("prime number {} and {}".format(a,b))

    N=a*b
    product=(a-1)*(b-1)

    #finding E

    for i in range(1,99999):
        if product%i==0:
            E=i
            break


    #finding D
    for i in range(0,product-1):
        if ((i*E)%product==1):
            D=i
            break


    PT=[]
    pt=[]
    ct=[]
    CT=[]

    PT=input("Enter the plain text:")
    for i in PT:
        pt.append(ord(i))
    print("plain text:", PT)
    for i in pt:
        ct.append((i**E)%N)
```

```
    print("cipher text:",ct)

    for i in ct:
        CT.append(chr(((i**D)%N)))


    print("Decrypted key:",end="")
    for i in CT:
        print(i,end="")


N=0
E=0
D=0
generate_key()
```

In [ ]:

```
##ECC
```

In [ ]:

```
import math
import random

def generate_points(a,b):
    if(4*(a**3) + 27*(b**2) !=0):
        x=1
        print("generating")
        while True:
            rhs= (x**3)+(a*x)+b
            y=int(math.sqrt(rhs))
            lhs= (y**2)
            if lhs==rhs:

                return [x,y]

            else:
                x+=1


    else:
        print("enter other coffiecient")


a=int(input("enter a"))
b=int(input("enter b"))

Private_a=13
Private_b=15

generators=generate_points(a,b)

msg=int(input("Enter the text:"))

public_a=[generators[0]*Private_a, generators[1]*Private_a]
public_b=[generators[0]*Private_b, generators[1]*Private_b]


print("Public Key A:", public_a)
print("Public Key B:", public_b)

k=random.randint(0,10)

c1= k * (generators[0]+generators[1])
c2= msg + (k*public_b[0]+k*public_b[1])

ciphertext=[c1,c2]

print("Ciphertext:",ciphertext)


r=Private_b*c1
plaintext=c2-r

print("Plaintext:",plaintext)
```

In [ ]:

```
```

In [ ]:

```
import random

def DHA():
    P=int(input("Enter P:Prime number"))
    flag=isprime(P)
    if flag==1:
        DHA()
    else:
        G=int(input("Enter primitive root for P:G"))
        print("value of P:",P)
        print("value of G:",G)

        a=int(input("Enter private key for a"))
        b=int(input("Enter private key for b"))
        print("Private A :",a)
        print("Private B :",b)



        x=int(pow(G,a,P))
        y=int(pow(G,b,P))


        print("Public A:",x)
        print("Public B:",y)

        ka=int(pow(y,a,P))
        kb=int(pow(x,b,P))

        print("secret key of a:",ka)
        print("secret key of b:",kb)



DHA()
```

In [ ]:

```
import random

def generate_key():
    p=random.randint(2,999)
    q=random.randint(2,999)

    flag1=isprime(q)
    flag2=isprime(p)

    if flag1==0 and flag2==0:
        RSA(p,q)
    else:
        generate_key()


def isprime(num):
    flag=0
    r=int(num/2)
    for i in range(2,r):
        if(num%i==0):
            flag=1
            break


    return flag

def RSA(a,b):
    print("prime number {} and {}".format(a,b))

    N=a*b
    product=(a-1)*(b-1)

    #finding E

    for i in range(1,99999):
        if product%i==0:
            E=i
            break


    #finding D
    for i in range(0,product-1):
        if ((i*E)%product==1):
            D=i
            break


    PT=[]
    pt=[]
    ct=[]
    CT=[]

    PT=input("Enter the plain text:")
    for i in PT:
        pt.append(ord(i))
    print("plain text:", PT)
    for i in pt:
        ct.append((i**E)%N)
```

```
    print("cipher text:",ct)

    for i in ct:
        CT.append(chr(((i**D)%N)))


    print("Decrypted key:",end="")
    for i in CT:
        print(i,end="")


N=0
E=0
D=0
generate_key()
```

In [ ]:

```
## AES
```

In [21]:

```
import os
from Crypto.Util.Padding import pad,unpad
from Crypto.Cipher import AES


def getkey(keysize):
    key=os.urandom(keysize)
    return key

def getIV(blocksize):
    iv=os.urandom(blocksize)
    return iv


KEYSIZE=16
BLOCKSIZE=16

plaintext="Hello, Your Security Buddy!"
key=getkey(KEYSIZE)
iv=getIV(BLOCKSIZE)

cipher1 =AES.new(key,AES.MODE_CBC,iv)
ciphertext=cipher1.encrypt(pad(plaintext.encode(),BLOCKSIZE))

print(ciphertext)

cipher2=AES.new(key,AES.MODE_CBC,iv)
plaintext2=unpad(cipher2.decrypt(ciphertext),BLOCKSIZE)

print(plaintext2.decode())
```

```
b'`\x8b\xbaC-\xac:\x85\xd9\xad\xe6\x14\x13BD\xb2z,\x94\xa0\xf4FBli?2XrnT\xc
a'
Hello, Your Security Buddy!
```

In [ ]:

```
```