PART B

1. Feistel Cipher

```
s=input("Enter String: ")
k=input("Enter key: ")

res=""
key=""

for ch in s:
    res+=format(ord(ch),'08b')

for ch in k:
    key+=format(ord(ch),'08b')
print("Key: "+key)

mid=len(s)//2
right=res[mid:]
left=res[:mid]

s=bin(int(right,2)+int(key,2))
ans=bin(int(s[2:],2)^int(left,2))
newr=ans[2:]
newl=right
newl,newr=newr,newl


s=bin(int(newr,2)+int(key,2))
ans=bin(int(s[2:],2)^int(newl,2))
l=newr
r=ans[2:]
l,r=r,l

cip=l+r

if len(cip) != len(res):
    while len(cip) != len(res):
        cip = "0" + cip
print("CipherText is: "+cip)

msg=""
for i in range(0,len(cip),8):
    temp=cip[i:i+8]
    d=int(temp,2)
    msg+=chr(d)
print("Decypted Text is: "+msg)
```

2. Hill Cipher

```
import numpy as np

def hill(keyMatrix, plaintext):
    cipherText = ""
    for i in range(0, len(plaintext), 2):
        c1 = plaintext[i]
        c2 = plaintext[i + 1]
        L = [ord(c1) % 65, ord(c2) % 65]
        res = np.dot(L,keyMatrix) # or np.dot(keyMatrix,L)
        res[0] = res[0] % 26
        res[1] = res[1] % 26
        cipherText += chr(res[0] + 65)
        cipherText += chr(res[1] + 65)
    return cipherText

def createKeyMatrix(key):
    keyMatrix[0][0] = ord(key[0]) % 65
    keyMatrix[0][1] = ord(key[1]) % 65
    keyMatrix[1][0] = ord(key[2]) % 65
    keyMatrix[1][1] = ord(key[3]) % 65
    return keyMatrix

key = input("Enter Key: ")
keyMatrix=[[0,0],[0,0]]
keyMatrix = createKeyMatrix(key)

plaintext = input("Enter plaintext of size even: ")
enMsg = hill(keyMatrix, plaintext)
print("Encryption: " + enMsg)

# Creating the modular multiplicative inverse matrix
det = keyMatrix[0][0] * keyMatrix[1][1] - keyMatrix[0][1] * keyMatrix[1][0]
try:
    det_inv = pow(int(det), -1, 26)  # Calculate the modular multiplicative inverse
    IMatrix = np.array([[keyMatrix[1][1], -keyMatrix[0][1]], [-keyMatrix[1][0],
keyMatrix[0][0]]]) * det_inv % 26
    print("Decryption: " + hill(IMatrix, enMsg))
except:
    print("Wrong key!!")
#determinant of the matrix must be coprime with 26. Eg: HELP , HELO can be used as key
```

3. Playfair

```
key = input("Enter key: ")
key=key.upper()

keysAlready=[]
mapper={}

matrix=[[0,0,0,0,0],
        [0,0,0,0,0],
        [0,0,0,0,0],
        [0,0,0,0,0],
        [0,0,0,0,0]]
i=0
j=0
#key matrix creation

for ch in key:
    if(ch not in keysAlready):
        matrix[i][j]=ch
        mapper[ch]=(i,j)
        keysAlready.append(ch)
        j=(j+1)%5
        if(j==0):
            i+=1
for ascii in range(65,91):
    ch=chr(ascii)
    if(ch not in keysAlready and ch!='J'):
        matrix[i][j]=ch
        mapper[ch]=(i,j)
        keysAlready.append(ch)
        j=(j+1)%5
        if(j==0):
            i+=1
print(matrix)

#plain text modification

plaintext=input("Enter plain text: ")
plaintext=plaintext.upper()

for i in range(0,len(plaintext)-1,2):
    if(plaintext[i]==plaintext[i+1]):
       plaintext=plaintext[:i+1]+"X"+plaintext[i+1:]
if(len(plaintext)%2!=0):
       plaintext+="X"
print(plaintext)

#encrption
cipherText=""
for i in range(0,len(plaintext)-1,2):
    char1=plaintext[i]
```

```
    char2=plaintext[i+1]

    (row1,col1)=mapper[char1]
    (row2,col2)=mapper[char2]

    if(row1==row2):
        row=row1=row2
        cipherText+=matrix[row][(col1+1)%5]
        cipherText+=matrix[row][(col2+1)%5]
    elif(col1==col2):
        col=col1=col2
        cipherText+=matrix[(row1+1)%5][col]
        cipherText+=matrix[(row2+1)%5][col]
    else:
        cipherText+=matrix[row1][col2]
        cipherText+=matrix[row2][col1]
print(cipherText)

#decyption
Text=""
for i in range(0,len(cipherText)-1,2):
    char1=cipherText[i]
    char2=cipherText[i+1]

    (row1,col1)=mapper[char1]
    (row2,col2)=mapper[char2]

    if(row1==row2):
        row=row1=row2
        Text+=matrix[row][(col1-1)%5]
        Text+=matrix[row][(col2-1)%5]
    elif(col1==col2):
        col=col1=col2
        Text+=matrix[(row1-1)%5][col]
        Text+=matrix[(row2-1)%5][col]
    else:
        Text+=matrix[row1][col2]
        Text+=matrix[row2][col1]
print(Text)
```

4. RSA

```
import math
def gcd(a,b):
    if(b==0):
        return a
    return gcd(b,a%b)

p=3
q=7
n=p*q
o=(p-1)*(q-1)
e=2

while(e<o):
    if(gcd(o,e)==1):
        break
    else:
        e=e+1

k=2
d=(1+k*o)/e
m=12.0
print("Message: "+str(d))
print("Value of e: "+str(e))

c=pow(m,e)
c=math.fmod(c,n)
print("Encypted Data: "+str(c))

b=pow(c,d)
b=math.fmod(b,n)
print("Decypted Data: "+str(b))
```

All the Best 😊
-Nelson D Cunha