In [127]:

```
#Segunda Entrega
__autor__ = "Jhon Steeven Cabanilla Alvarado"
__email__ = "jhonsteeven.cabanilla@alumnos.uva.es"
__credits__ = """Jhon Steeven Cabanilla Alvarado, Codigos y Criptografía, Grado en Inge
nieria Informatica,
Universidad de Valladolid, Segunda entrega de ejercicios"""

import random
import math

"""********************************************************************************
                                        EJERCICIO #1
********************************************************************************"""

"Funcion que cifra con el cifrado de Vignere"
def cifrado_vignere(Alfabeto, clave, mensaje):
    cifrado = []
    i = 0
    j = 0

    while len(cifrado) != len(mensaje):

        #Comprobamos que el caracter del mansaje no se trate de un espacio en blanco
        if mensaje[j] == " ":
            cifrado.append(" ")
            j += 1

        #Obtenemos la posicion que ocupa el caracter i de nuestra clave. Hacemos lo mis
mo con el caracter j de nuestro mensaje
        pos = Alfabeto.index(clave[i])
        pos_clave = Alfabeto.index(mensaje[j])

        #Obtenemos el caracter nuevo
        nueva_letra = (pos + pos_clave)%27

        #Procedemos a añadirlo a nuestra lista cifrado
        cifrado.append(Alfabeto[nueva_letra])

        i += 1
        j += 1

        #Comprobamos que la i no supere la len de la clave
        if i >= len(clave):
            i = 0

    return cifrado


"Funcion que descifra según el cifrado de Vignere"
def descifrado_vignere(Alfabeto, clave, cifrado):
    descifrado = []

    i = 0
    j = 0

    while len(descifrado) != len(cifrado):

        #Comprobamos que el caracter del mansaje no se trate de un espacio en blanco
        if cifrado[j] == " ":
```

```
            descifrado.append(" ")
            j += 1

        #Obtenemos la posicion que ocupa el caracter i de nuestra clave. Hacemos lo mis
mo con el caracter j de nuestro mensaje
        pos = Alfabeto.index(clave[i])
        pos_clave = Alfabeto.index(cifrado[j])

        nueva_letra = (pos_clave - pos)%27

        #Obtenemos el caracter nuevo
        descifrado.append(Alfabeto[nueva_letra])

        i += 1
        j += 1

        #Comprobamos que la i no supere la len de la clave
        if i >= len(clave):
            i = 0

    return descifrado

"""*****************************************************************************
                                    EJERCICIO #2
*****************************************************************************"""

"a)"
"Funcion que calcula la clave publica y la clave privada del criptosistema RSA dados do
s primos p y q"
def RSA(p,q):
    #Calculamos n y phi
    n = p*q
    phi = (p-1)*(q-1)

    #Eligimos e tal que 0 < e < phi y mcd(e,phi) = 1
    e = random.randint(1,phi)
    #Esto se realizara hasta que obtengamos un e adecuado
    while gcd(e,phi) != 1:
        e = random.randint(1,phi)

    #Procedemos a calcular d
    d = inverse_mod(e,phi)

    #Clave publica
    clave_publica = (n,e)

    #Clave privada
    clave_privada = (d,p,q,phi)

    return clave_publica, clave_privada

"b)"
"Funcion que dada la clave publica (n,e) del criptosistema RSA y un mensaje Zn, cifra d
icho mensaje"
def cifrado_RSA(clave_publica, mensaje):

    n = clave_publica[0]
    e = clave_publica[1]

    cifrado = power_mod(mensaje,e,n)
```

```
    return cifrado

"c)"
"Funcion que dada la clave privada del criptosistema RSA y un mensaje cifrado, descifra
dicho mensaje"
def descifrado_RSA(clave_privada, cifrado):
    d = clave_privada[0]
    n = clave_privada[1] * clave_privada[2]

    mensaje = power_mod(cifrado,d,n)

    return mensaje



"""*********************************************************************************
                                    EJERCICIO #3
*********************************************************************************"""
"a)"
"Funcion que cifra un texto M dada la clave publica del criptosistema RSA"
def cifrado_bloque_RSA(M,k):

    #Comenzamos buscando p y q tales que cumplan: N^k <= n < N^k+1
    N = 256

    limite_inf = pow(N,k)
    limite_sup = pow(N,k+1)
    raiz = int(math.sqrt(limite_inf))

    p, q = 0,0
    ite = 100
    while(p==q or limite_inf>(p*q) or limite_sup<=(p*q)):
        p = next_prime(raiz-250*ite)
        q = next_prime(raiz+1000*ite)
        ite += 1

    #Llamamos a nuestra funcion RSA para obtener las claves a partir de los primos que
 hemos obtenido
    publica, privada = RSA(p,q)

    #Procedemos a dividir nuestro mensaje en bloques de longitud k
    bloques = dividir_mensaje(M,k)
    cifrado = ""

    #Procedemos a cifrar cada palabra dentro de los bloques
    for i in bloques:
        palabra = cifrado_bloque(publica,i,k,N)
        for j in palabra:
            cifrado += chr(j)

    return cifrado, privada, N, publica[0]


"Funcion que nos divide el mensaje enviado en bloques de longitud k"
def dividir_mensaje(mensaje,k):

    lista = list(mensaje)
    bloques = []
    ite = 0

    while(len(lista)-ite >= k):
```

```
        palabra = ""
        for j in range(k):
            palabra += lista[j+ite]

        ite += k
        bloques.append(palabra)


    if ite != len(lista):
        palabra = ""

        for j in range(ite,len(lista)):
            palabra += lista[j]

        for z in range(len(palabra),k):
            palabra += " "

        bloques.append(palabra)

    return bloques


"Funcion que cifra en codigo UNICODE"
def cifrado_bloque(publica,M,k,N):

    n = publica[0]
    e = publica[1]
    m = 0
    cont = 1

    #Procedemos a calcular m
    #Utilizamos la funcion .ord(), la cual recibe un caracter y devuelve su representac
ion en codigo unicode
    for i in M:
        m += ord(i) * pow(N,k-cont)
        cont += 1

    #Ciframos
    c = power_mod(m,e,n)

    #Calculamos c en BASE N
    aux = c
    lista = []
    while(aux>=N):
        resto = int(aux%N)
        #Insertamos al principio de la lista cada resto
        lista.insert(0,resto)
        #Actualizamos el valor de nuestra variable auxiliar
        aux = int(aux/N)

        if(aux<N):
            lista.insert(0,aux)

    return lista


"b)"
"Funcion que descifra un texto cifrado dada la clave privada del criptosistema RSA"
def descifrado_bloque_RSA(privada,cifrado,n,k,N):
    lista = dividir_mensaje(cifrado,k)
```

```
    descifrado = ""

    for i in lista:
        palabra = descifrado_bloque(privada,i,k,N,n)
        for j in palabra:
            descifrado += chr(j)

    return descifrado

def descifrado_bloque(privada,cifrado,k,N,n):
    ite = 0
    c = 0
    d = privada[0]

    for i in range(len(cifrado)):
        c += ord(cifrado[-i-1])*pow(N,0+ite)
        ite += 1

    #Calculamos b_i
    b = power_mod(c,d,n)
    aux = b
    lista = []

    #Escribimos b_i en BASE N con longitud K
    while(aux>=N):

        resto = int(aux%N)
        #Insertamos al principio de la lista cada resto
        lista.insert(0,resto)
        #Actualizamos el valor de nuestra variable auxiliar
        aux = int(aux/N)

        if(aux<N):
            lista.insert(0,aux)

    return lista



"""*****************************************************************************
                                    EJERCICIO #4
*****************************************************************************"""
"a)"
"Funcion que por fuerza bruta factoriza n, la cual es dada"
def factoriza(n):
    #Para factorizar n utilizo divisiones sucesivas y además empleo el paradigma de pro
gramacion modular, el cual permite
    #disminuir la complejidad del algoritmo.

    primos = []
    factor_primo = 2

    #Condicion de iteracion
    while n>1:
        if n % factor_primo == 0:
            #Actualizamos el valor de n. Para encontrar los siguientes factores primos,
utilizamos el cociente entero de la
            #division entre n y el factor primo encontrado
            n = n // factor_primo
            primos.append(factor_primo)
```

```
        else:
            #En caso de no ser un factor primo de n, utilizo next_prime para obtener el
siguiente primo
            factor_primo = next_prime(factor_primo)

    return primos[0], primos[1]


"b)"
"Funcion que calcula por fuerza bruta phi(n) dada la clave publica (n,e)"
def calcula_phi(clave_publica):
    #Utilizamos la funcion creada en el apartado a
    n = clave_publica[0]
    p,q = factoriza(n)

    phi = (p-1)*(q-1)

    return phi


"c)"
"Funcion que calcula por fuerza bruta la clave privada d dada la clave publica (n,e)"
def calcula_d(clave_publica):
    #Utilizamos la funcion creada en el apartado a
    n = clave_publica[0]
    e = clave_publica[1]
    p,q = factoriza(n)

    phi = (p-1)*(q-1)

    #Procedemos a calcular d
    d = inverse_mod(e,phi)

    return d

"d)"
"""
¿Hasta que tamano del input las funciones anteriores son capaces de terminar?
Son capaces de terminar cualquier tamaño siempre y cuando se cumpla que n sea producto
 de 2 numeros primos.
Obviamente, cuando mayor sean estos numeros, mayor sera el coste a la hora de factoriza
r n

¿Hemos atacado con exito el criptosistema RSA?
Sí, hemos sido capaces de atacar exitosamente este criptosistema

"""


"""*********************************************************************************
                                    EJERCICIO #5
*********************************************************************************"""
"a)"
"Funcion que calcula un primo p y un generador del grupo ciclico (Zp)*, dado un tamaño
 minimo para un primo"
def generador_multiplicativo(q):

    #Comprobamos si P: 2q + 1 es primo mediante el metodo de Sage .is_prime()
    p = 2*q + 1

    while is_prime(p) != True:
```

```
        #En el caso de que no sea primo, obtenemos el siguiente primo grande
        q = next_prime(q)
        p = 2*q + 1

    #De esta forma tenemos que p-1 = 2(primo) * q(primo), por lo tanto ya tenemos facto
rizado p-1

    #Por ultimo, buscamos al azar g tal que, 1<g<p-1 y que ademas, cumpla con lo siguie
nte: g^2 != 1 mod p y que g^g != 1 mod p

    g = random.randint(1,p-1)
    sol = False

    while not sol:
        if power_mod(g,2,p) != 1 and power_mod(g,g,p) != 1:
            sol = True

        else:
            g = random.randint(1,p-1)

    return p,g

"b)"
"Funcion que crea la clave publica y la clave privada del criptosistema ElGamal"
def elgamal(p,g):

    #A partir de p y g que los hemos obtenido mediante la anterior funcion...
    #Procedemos a generar las claves

    #La clave privada será un numero aleatorio (a) comprendido entre 2 y p-2
    a = random.randint(2,p-2)
    clave_privada = (a)

    #La clave publica entará formada por (p,g,A)
    #Por lo tanto, procedemos a calcular A
    A = power_mod(g,a,p)
    clave_publica = (p,g,A)

    return clave_publica, clave_privada

"c)"
"Funcion que cifra un mensaje dada la clave publica del criptosistema ElGamal"
def cifrado_elgamal(clave_publica, M):

    p = clave_publica[0]
    g = clave_publica[1]
    A = clave_publica[2]

    #El mensaje de cifrado es el par (B,C)

    #Primero de todo, tenemos que elegir un numero aleatorio (b) para cifrar dicho par
 que debe estar comprendido entre 2 y p-2
    b = random.randint(2,p-2)
    #Calculamos B y C de la siguiente manera:
    B = power_mod(g,b,p)

    C = power_mod(A,b,p) * M % p

    return B,C
```

```
"d)"
"Funcion que descifra un mensaje cifrado dada la clave privada del criptosistema ElGama
l"
def descifrado_elgamal(a, B, C, p):

    #Para descifrar comenzamos calculando K:
    K = power_mod(B,a,p)

    #Y recuperamos el mensaje
    M = power_mod(B,p-1-a,p)*C % p

    return M


"""*****************************************************************************
                                    EJERCICIO #6
*****************************************************************************"""
"a)"
"Funcion que firma digitalmente un mensaje utilizando la firma digital con RSA y una fu
ncion hash"
def firma_RSA(d,n,M):

    #Comenzamos utilizando una funcion hash:
    hM = hash(M)

    #Calculamos la firma de la siguiente manera:
    S = power_mod(hM,d,n)

    return S

"b)"
"Funcion que comprueba si la firma calculada en la anterior funcion es valida o no, dad
o un mensaje, dicha firma  y la clave e"
def comprueba_firma_RSA(M,S,e,n):

    #Comenzamos generando nuevamente h(M)
    hM = hash(M)

    #Calculamos S^e mod n y comprobamos que este valor sea igual al de la funcion hash
    firma = power_mod(S,e,n)

    if firma == hM:
        print("Firma valida")
    else:
        print("Firma no valida")



"""*****************************************************************************
                                    EJERCICIO #7
*****************************************************************************"""
"a)"
"""
Funcion que firma digitalmente un mensaje utilizando la firma digital con RSA y una fun
cion hash, utilizando la funcion a)
del ejercicio #6 y ademas, cifra el mensaje utilizando RSA a partir de la funcion b) de
l ejercicio #2
"""
def firma_cifrado_RSA(privada_Alice, publica_Bob, M):

    d = privada_Alice[0]
```

```
    n = publica_Bob[0]

    #Alice firma el mensaje utilizando su clave privada
    S = firma_RSA(d, n, M)

    #Bob por su parte, Bob cifra dicho mensaje utilizando su clave publica
    cifrado = cifrado_RSA(publica_Bob, M)

    return cifrado, S

"b)"
"""Funcion que descifra un mensaje cifrado utilizando la funcion C) del ejercicio #2 y
 que ademas, comprueba si la firma es
valida o no de acuerdo a la funcion b) del ejercicio #6"""
def comprueba_firma_descifrado_RSA(publica_Alice, privada_Bob, cifrado, firma):

    n = publica_Alice[0]
    e = publica_Alice[1]

    #Desciframos el mensaje
    descifrado = descifrado_RSA(privada_Bob, cifrado)

    #Comprobamos si la firma es valida
    comprueba_firma_RSA(descifrado,firma,e,n)

    return descifrado


"""*********************************************************************************
                                    MAIN
*********************************************************************************"""

def main():

    """
    ------------------------------------README-----------------------------
    A continuacion se muestra el codigo para probar los ejercicios implementados.
    En la cabecera de cada uno se indica el numero del ejercicio y el codigo de este, s
e encuentra comentado.
    Para probar cada uno, hay que descomentarlo, quitando las 3 comillas debajo del tit
ulo y las 3 finales debajo de
    la ultima linea del codigo que se refiere a cada ejercicio.
    """

    # ---EJERCICIO #1
    """
    #En cuanto al alfebeto, utilizamos el alfabeto que incluye la ñ, por lo tanto tendr
emos 27 caracteres
    Alfabeto = "ABCDEFGHIJKLMNÑOPQRSTUVWXYZ"
    #TEXTO A CIFRAR
    mensaje = "HOLA MUNDO"
    clave = "PYTHON"
    modo = int(input("Modo: (1)CIFRAR (2)DESCIFRAR: "))

    if modo == 1:
        cifrado = cifrado_vignere(Alfabeto, clave, mensaje)
        print(cifrado)
    else:
        cifrado = "WNEH AHCBI"
        descifrado = descifrado_vignere(Alfabeto, clave, cifrado)
        print(descifrado)
```

```
    """

    # ---EJERCICIO #2
    """
    p, q = 11, 23
    publica, privada = RSA(p,q)
    print("Clave publica:",publica," Clave privada:",privada)
    c = cifrado_RSA(publica, 200)
    print("Crifrado:",c)

    m = descifrado_RSA(privada, c)
    print("Mensaje original:",m)
    """

    # ---EJERCICIO #3
    """
    #Definimos k
    k = 5
    #Mensaje original
    M = "Los ordenadores cuanticos acabaran con el blockchain"
    print("El mensaje original es:",M)

    cifrado,privada,N,n = cifrado_bloque_RSA(M,k)
    print("El mensaje cifrado es:",cifrado)

    descifrado = descifrado_bloque_RSA(privada,cifrado,n,k,N)
    print("El mensaje descifrado es:",descifrado)
    """

    #---EJERCICIO #4
    """
    n = 829348951
    clave_publica = (n,197)
    p, q = factoriza(n)
    print("p=",p,"q=",q)
    phi = calcula_phi(clave_publica)
    print("Phi=",phi)
    d = calcula_d(clave_publica)
    print("d=",d)
    """

    #---EJERCICIO #5
    """
    q = next_prime(2^100) #Tamaño minimo para un primo
    p,g = generador_multiplicativo(q)
    print("Generador:",g,",Primo p:",p)

    publica, privada = elgamal(p,g)
    print("Clave publica:",publica,"\n","Clave privada:",privada)

    #El mensaje para cifrar debe cumplir que 0<= M < p-1
    M = random.randint(0,p-1)
    print()
    print("Mensaje a cifrar:",M)
    B,C = cifrado_elgamal(publica, M)
    print("Mensaje cifrado:",C)
    #Le paso tambien p ya que este es publico y ademas lo necesito para hacer los modul
os
    p = publica[0]
    descifrado = descifrado_elgamal(privada,B,C,p)
    print("Mensaje descifrado:",M)
```

```
    """

    #---EJERCICIO #6
    """
    #Utilizamos los mismos valores para las claves del ejercicio 2, tal y como indica e
l enunciado
    p, q = 11, 23
    publica, privada = RSA(p,q)

    n = p*q
    d = privada[0]
    print("d:",d)
    #El mensaje M tiene que cumplir: 0<= M < n
    M = random.randint(0,n)
    print("Mensaje a firmar:",M)

    S = firma_RSA(d,n,M)
    print("Firma:",S)

    e = inverse_mod(d,n)
    print("e:",e)
    comprueba_firma_RSA(M,S,e,n)
    """

    #---EJERCICIO #7
    """
    #Alice y Bob escogen los siguientes primos:
    p = next_prime(12345)
    q = next_prime(56789)

    #Obtenemos las claves de Alice
    publica_Alice, privada_Alice = RSA(p,q)
    print("Publica de Alice:",publica_Alice,"Privada de Alice:",privada_Alice)
    #Obtenemos las claves de Bob
    publica_Bob, privada_Bob = RSA(p,q)
    print("Publica de Bob:",publica_Bob,"Privada de Bob:",privada_Bob)

    #Mensaje
    M = 123456789
    print("Mensaje:",M)

    cifrado, firma = firma_cifrado_RSA(privada_Alice, publica_Bob, M)
    print("Mensaje cifrado:",cifrado,"Y la firma:",firma)

    M = comprueba_firma_descifrado_RSA(publica_Alice, privada_Bob, cifrado, firma)
    print("Mensaje descifrado:",M)
    """


if __name__ == '__main__':
    main()
```