Códigos y Criptografía

Práctica 2

Alonso Sayalero Blázquez

In [35]:
```
#*****************************************************************
#                           Ejercicio 1
#*****************************************************************
# Función para ordenar el arbol
def ordenar(alfabeto, probabilidad):
    for i in range(len(alfabeto) - 1):
        if len(alfabeto[i]) > len(alfabeto[i + 1]):
            auxiliar = alfabeto[i]
            alfabeto[i] = alfabeto[i + 1]
            alfabeto[i + 1] = auxiliar
            auxiliar = probabilidad[i]
            probabilidad[i] = probabilidad[i + 1]
            probabilidad[i + 1] = auxiliar

# Funcion Huffman, el parametro alfabeto contiene el alfabeto y se transforma en
# el parametro probabilidad contiene las probabilidades
def creacion_huffman(alfabeto, probabilidad):
    while len(probabilidad) > 1:
        minimo1 = 0
        minimo2 = 1

        for i in range(2, len(probabilidad)):
            if probabilidad[minimo1] > probabilidad[i]:
                minimo1 = i
            elif probabilidad[minimo2] > probabilidad[i]:
                minimo2 = i

        if minimo2 < minimo1:
            auxiliar = minimo1
            minimo1 = minimo2
            minimo2 = auxiliar

        probabilidad[minimo1] = probabilidad[minimo1] + probabilidad[minimo2]
        probabilidad.pop(minimo2)
        alfabeto[minimo1] = [alfabeto[minimo1], alfabeto[minimo2]]
        alfabeto.pop(minimo2)
        ordenar(alfabeto, probabilidad)
```

In [107…
```
#*****************************************************************
#                           Ejercicio 2
#*****************************************************************
# a) Funcion que calcula la clave privada y la clave publica de RSA
import random

def calculo_claves_RSA(p, q):
    # Calculo de n y phi
    n = p * q
    phi = (p - 1) * (q - 1)
```

```
    # Eleccion de e tal que 0 < e < phi y mcd(e, phi) = 1
    e = random.randint(1, phi)
    while(gcd(e, phi) != 1):
        e = random.randint(1, phi)

    # Calculo de d
    d = inverse_mod(e, phi)

    # Devuelve la clave publica ([n, e]) y la clave privada ([d, p, q, phi])
    return [n, e], [d, p, q, phi]

# b) Funcion que cifra un mensaje con RSA
def cifrado_RSA(clave_publica, mensaje):
    n = clave_publica[0]
    e = clave_publica[1]

    mensaje_cifrado = power_mod(mensaje, e, n)

    return mensaje_cifrado

# c) Funcion que descifra un mensaje con RSA
def descifrado_RSA(clave_privada, mensaje_cifrado):
    n = clave_privada[1] * clave_privada[2]
    d = clave_privada[0]

    mensaje = power_mod(mensaje_cifrado, d, n)

    return  mensaje
```

In [106…

```
#*****************************************************************
#                           Ejercicio 3
#*****************************************************************
# a) Funcion que cifra un mensaje por cifrado en bloque de RSA (N = 256)
def cifrar(publica, bloque, k, N):
    n = publica[0]
    e = publica[1]

    # Calcular m
    exponente = 1
    m = 0
    for i in bloque:
        m += ord(i) * N^(k - exponente)
        exponente += 1

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

    # Calcular c en base N (C)
    C = []
    auxiliar = c
    while auxiliar > N:
        resto = int(auxiliar % N)
        C.insert(0, resto)
        auxiliar = int(auxiliar / N)

        if auxiliar < N:
            C.insert(0, auxiliar)
    return C
```

```
def cifrado_en_bloque_RSA(clave_publica, mensaje):
    N = 256

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

    # Calcular k
    k = 10
    while true:
        if N^k <= n and n < N^(k+1):
            break
        elif N^k > n:
            k -= 1
        elif N^(k + 1) < n:
            k += 1

    # Separacion en bloques
    bloques = [mensaje[i:i+k] for i in range(0, len(mensaje), k)]

    if len(bloques[len(bloques) - 1]) < k:
        while len(bloques[len(bloques) - 1]) < k:
            bloques[len(bloques) - 1] = bloques[len(bloques) - 1] + (" ")

    cifrado = ""
    for i in bloques:
        palabra = cifrar(clave_publica, i, k, N)
        for j in palabra:
            cifrado += chr(j)

    return cifrado

# Funcion que descifra el mensaje cifrado en la anterior funcion
def descifrar(privada, bloque_cifrado, k, N, n):
    d = privada[0]

    exponente = 0
    c = 0

    for i in range(len(bloque_cifrado)):
        c += ord(bloque_cifrado[-i-1]) * N^(0 + exponente)
        exponente += 1

    # Calcular b
    b = power_mod(c, d, n)

    C = []
    auxiliar = b
    while auxiliar > N:
        resto = int(auxiliar % N)
        C.insert(0, resto)
        auxiliar = int(auxiliar / N)

        if auxiliar < N:
            C.insert(0, auxiliar)

    return C

def descifrado_en_bloque_RSA(clave_publica, clave_privada, mensaje_cifrado):
    N = 256
    n = clave_publica[0]
```

```
    # Calcular k
    k = 10
    while true:
        if N^k <= n and n < N^(k+1):
            break
        elif N^k > n:
            k -= 1
        elif N^(k + 1) < n:
            k += 1

    # Separacion en bloques
    bloques = [mensaje_cifrado[i:i+k+1] for i in range(0, len(mensaje_cifrado),

    mensaje = ""
    for i in bloques:
        palabra = descifrar(clave_privada, i, k, N, n)
        for j in palabra:
            mensaje += chr(j)

    return mensaje
```

In [108…

```
#*****************************************************************
#                           Ejercicio 4
#*****************************************************************
# a) Fuerza bruta para sacar n
def factorizar_n(n):
    primos = []
    factor = 2

    while n > 1:
        if n % factor == 0:
            n = n // factor
            primos.append(factor)
        else:
            factor = next_prime(factor)

    return primos[0], primos[1]

# b) Fuerza bruta para sacar phi a partir de la clave publica
def factorizar_phi(n, e):
    p, q = factorizar_n(n)

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

    return phi

# c) Fuerza bruta para calcular la clave privada a partir de la clave publica
def calcular_d(n, e):
    phi = factorizar_phi(n, e)

    d = inverse_mod(e, phi)

    return d
```

In [105…

```
#*****************************************************************
#                           Ejercicio 5
#*****************************************************************
# Funcion que firma un documento mediante el Gamal
```

```
def firma_el_gamal(a, p, g, mensaje):
    valor_hash = abs(hash(mensaje))
    while valor_hash > p - 2:
        valor_hash = int(valor_hash / 10)

    # Calcular A
    A = power_mod(g, a, p)

    # Generar un k aleatorio tal que mcd(k, p - 1) = 1
    k = random.randint(1, p - 2)
    while(gcd(k, p - 1) != 1):
        k = random.randint(1, p -2)

    # Calcular r y s
    r = power_mod(g, k, p)
    s = power_mod(k^(-1) * (abs(valor_hash - a * r)), 1, p - 1)

    return r,s

# Funcion que comprueba que la firma sea valida, firma ([r, s]),
# clave_publica ([p, g, A])
def comprobar_firma(mensaje, firma, clave_publica):
    r = firma[0]
    p = clave_publica[0]
    if r < 1 or r > p - 1:
        return false

    A = clave_publica[2]
    valor_hash = abs(hash(mensaje))
    while valor_hash > p - 2:
        valor_hash = int(valor_hash / 10)
    s = firma[1]
    g = clave_publica[1]

    if power_mod((A^r * r^s), 1, p) == power_mod(g, valor_hash, p):
        return true
    else:
        return false
```

In [1]:

```
#*****************************************************************
#                           Ejercicio 6
#*****************************************************************
# Funcion que firma un mensaje y luego lo cifra
def cifrado_firma(clave_privada_gamal, clave_publica_RSA, mensaje):
    a = clave_privada_gamal
    p = 7
    g = 13

    r, s = firma_el_gamal(a, p, g, mensaje)

    mensaje_cifrado = cifrado_RSA(clave_publica_RSA, mensaje)

    return r, s, mensaje_cifrado

# Funcion que descifra un mensaje y luego comprueba que la firma es correcta
def descifrado_comprobar_firma(clave_publica_gamal, clave_privada_RSA, mensaje_c
    mensaje = descifrado_RSA(clave_privada_RSA, mensaje_cifrado)

    firma_correcta = comprobar_firma(mensaje, firma, clave_publica_gamal)
```

```
    return mensaje, firma_correcta
```