Exercício 2 - DSA

Neste exercício temos de implementar um algoritmo de DSA.

Como tal existem vários algoritmos disponíveis. Após uma breve conversa entre os dois elementos do grupo achamos por bem implementar uma definição do algoritmos de assinatura de **El Gamal** visto que já foi bastantes vezes referida e falada tanto na unidade curricular anterior (Tecnologias Criptográficas) como nesta mesma unidade curricular. Esta definição encontra-se nos anexos deste relatório.

Parâmetro p:

- **p** - Um número primo de tamanho 2^(tamanho-1)

In [1]:
```
def generateP(tam):
    p = random_prime(2^(tam-1))
    print("p: "+ str(p))
    return p
```

Parâmetro g:

- **g** - Inteiro compreendido em: 0 < g < p , tal que este será um gerador

In [2]:
```
def generateG(p):
    g = randint(1,p-1)
    print("g: "+ str(g))
    return g
```

Gerador de parâmetros:

`generateParameters()` é a função é responsável por gerar os parâmetros:

- **p**
- **g**

In [3]:
```
def generateParameters(tam):
    p = generateP(tam)
    g = generateG(p)
    return p,g
```

Parâmetro x :

- **x** - Inteiro compreendido entre os valores: 1 < x < p-1

Este parâmetro é o segredo do utilizador, que será usado na chave privada e na geração da chave pública.

In [4]:
```
def generateX(p):
    x = randint(2,p-2)
    print('x: ' + str(x))
    return x
```

Chave privada

Esta é a função que recebe os parâmetros e compõe a chave privada.

In [5]:
```
def createPrivateKey(p,g,x):
    privateKey= (p,g,x)
    print('privateKey: ' + str(privateKey))
    return privateKey
```

Chave pública

Esta função é a que recebe os paramatros e compõe a chave pública

In [6]:
```
def createPublicKey(p,g,y):
    publicKey = (p,g,y)
    print('publicKey: ' + str(publicKey))
    return publicKey
```

Gerador de Chave

`generateKey(p,g)` é uma função que cria a chave pública e a chave privada. Começa-se por gerar a parte do segredo da chave privada, sendo invocada a função `generateX(p)` .

Depois a partir desse valor chega-se à parte pública da chave com a operação: `(g^x) mod p` .

Tendo esses dois valores as chaves privada e pública são compostas com as seguintes funções, respetivamente, `createPrivateKey(p,g,x)` e `createPublicKey(p,g,y)`

In [7]:
```
def generateKey(p,g):
    x = generateX(p)

    y =pow(g,x,p)
    print('y: ' + str(y))

    privateKey = createPrivateKey(p,g,x)
    publicKey = createPublicKey(p,g,y)

    return publicKey,privateKey
```

Parâmetro k:

- **k** - Inteiro compreendido entre os valores: 0<k<p-1 e gcd(k,p-1) = 1

In [8]:
```
def generateK(p):
    while(true):
        aux = randint(1,p-2)
        if(gcd(aux,p-1)== 1):
            print('found')
            k = aux
            break
        print('next')
    print("k: "+ str(k))
    return k
```

Assinatura

Esta função é a responsável pela assinatura de uma mensagem, recebendo como argumentos a chave privada e a mensagem.

Esta começa por decompor a chave privada pelos 3 parâmetros que a compõe: `p,g,x` .

Após isso gera-se um inteiro `k` com a função `generateK(p)` .

A partir deste `k` podemos calcular `r` e `l` :

- **r** : `r = g^k mod p`
- **l** : `l = k^-1 mod(p-1)`

Após isso calculamos `re` :

- **re**: `re = l*(hash(m)-r*x)`

Assim podemos calcular `s` :

- **s**: `s = re mod(p-1)`

Após verificarmos se este é positivo, criamos a assinatura com os parâmetros s e r: `signature=(r,s)`

In [9]:
```
def signMessage(privateKey,m):
    (p,g,x) = privateKey
    k = generateK(p)
    #g^k mod p
    r = power_mod(g,k,p)
    #print('r: ' + str(r))
    l = power_mod(k,-1,p-1)
    #print('l: ' + str(l))
    re = l*(hash(m)-r*x)
    #print('re: ' + str(re))
    s= power_mod(re,1,p-1)
    #print('s: ' + str(s))
    if (s < 0):
        print('s < 0')
        return ''
    signature = (r,s)
    print('signature: ' + str(signature))
    return signature
```

Verificação da assinatura

Esta função recebe como argumentos a chave pública, a mensagem e assinatura e é responsável se a assinatura é válida com aquela chave e com aquela mensagem.

O primeiro passo está em decompor a chave pública pelos três parâmetros que a compõe: `(p,g,y)`. Faremos o mesmo com a assinatura, obtendo agora: `(r,s)`.

A primeira fase da verificação da assinatura passa por verificar o tamanho dos parâmetros que a compõe:

- **r**: 0<r<p
- **s**: 0<s<p-1

Depois temos que fazer o calculo de dois valores e fazer a sua comparação: `val1` e `val2`:

- **val1** - Precisa de 4 operações distintas.
  - **cal1** - `y^r`
  - **cal2** - `r^s`
  - **cal** - `cal1*cal2`

    `val1 = cal mod p <=> val1 = y^r * r^s mod`
- **val2** - `g^(hash(m)) mod`

Depois deste cálculo a vericidade da assinatura depende do resultado da comparação destes dois valores:

- **Verdadeiro** se `val1 == val2`
- **Falso** se `val1 != val2`

In [10]:
```
def verifySignature(publicKey, m, signature):
    (p,g,y) = publicKey
    (r,s) = signature

    if(r<=0 or r >= p):
        print('Erro no tamanho de r')
        return False
    if(s<=0 or s >= p-1):
        print('Erro no tamanho de s')
        return False

    cal1 = y^r
    #print('tenho cal1')
    cal2 = r^s
    #print('tenho cal2')
    cal = cal1*cal2
    #print('tenho cal')
    #print('cal: ' +str(cal))

    val1= power_mod(cal,1,p)
    print('val1: ' + str(val1))
    val2 = power_mod(g,hash(m),p)
    print('val2: ' + str(val2))

    if(val1==val2):
        return True
    else:
        return False
```

Main

Esta função é a responsável por testar todas as funcionalidades implementadas.

Esta começa por gerar os parâmetros iniciais necessários para todo o processo.

Depois gera as chaves (privada e pública).

Assina uma mensagem com a chave privada e depois vai verificar essa mesma assinatura dessa mensagem.

A `main1` verifica com a mesma mensagem.

a `main2` verifica com uma mensagem diferente, provocando o erro propositado.

In [11]:
```
def main1():
    p,g = generateParameters(32)

    publicKey, privateKey = generateKey(p,g)

    m=b"Vamos assinar esta mensagem"
    signature = signMessage(privateKey,m)
    if(signature == ''):
        print('Erro na assinatura da mensagem')

    value = verifySignature(publicKey,m,signature)
    print('value: ' + str(value))
```

In [12]:
```
main1()
```

```
p: 1691291033
g: 1236029196
x: 494772165
y: 125931399
privateKey: (1691291033, 1236029196, 494772165)
publicKey: (1691291033, 1236029196, 125931399)
next
found
k: 554805665
signature: (1656598076, 86408788)
val1: 716857687
val2: 716857687
value: True
```

In [ ]:
```
def main2():
    p,g = generateParameters(32)

    publicKey, privateKey = generateKey(p,g)

    m=123
    signature = signMessage(privateKey,m)
    if(signature == ''):
        print('Erro na assinatura da mensagem')

    m2 =456
    value = verifySignature(publicKey,m2,signature)
    print('value: ' + str(value))
```

In [ ]:
```
main2()
```