86-CRIPTO-firma_digital

March 4, 2018

0.1 1 Generamos claves

```
In [1]: p = random_prime(floor(sqrt(16^65)),lbound=16^16)
```

```
In [2]: q= random_prime(floor(sqrt(16^65)),lbound=16^16)
```

```
In [3]: n = p*q;print n
```

```
1961741043601797916919804867861013846023376072489087023241743496368516941724 77
```

```
In [4]: print p-q
```

```
-361288464761693431612266338075140943964
```

```
In [5]: Phi = (p-1)*(q-1)
```

```
In [6]: def invertible(Phi):
            for int in xsrange(16^7,16^10):
                if gcd(int,Phi)==1:
                    return int,xgcd(int,Phi)
                    break
```

```
In [7]: SOL = invertible(Phi);SOL
```

```
Out[7]: (268435457,
          (1,
           -775263846734167445222470293053749966969698587973060442511285610587157969063 67,
           106083474))
```

```
In [8]: SOL[0]*SOL[1][1]+Phi*SOL[1][2]
```

```
Out[8]: 1
```

```
In [9]: clave_pr=SOL[1][1]%Phi;print clave_pr
```

```
1186477196867630471697334574807263879044110741223711371109176899299045302869 13
```

```
In [10]: 16^65 > n > 16^64 #Deben cumplirse
```

```
Out[10]: True
```

Ahora usamos las claves de este usuario (llamémosle B), y las del profesor si B quiere encriptar el mensaje, para enviar al profesor un mensaje firmado. La clave pública de B es (n,SOL[0]) y su clave privada es *clave_pr*.

0.2 2 Preparamos la firma

$C1$ es la cadena indicada en las instrucciones, que contiene los dos enteros que forman la clave pública de B.

```
In [11]: C1 = str(n)+'G'+str(SOL[0])+'G';print C1
```

```
1961741043601797916919804867861013846023376072489087023241743496368516941724 77G268435457G
```

```
In [12]: from hashlib import *
         HH =md5(C1).hexdigest()
         print HH
```

```
151c517a8d1ca528e0ed4c4870e87224
```

Ahora B debe encriptar el hash usando su clave privada, es decir, el d que es el inverso de e módulo *Phi*.

```
In [13]: alfb = '0123456789abcdef';len(alfb)

Out[13]: 16

In [14]: L_alfb = list(alfb)

         def ord2(c):
             return L_alfb.index(c)

         def chr2(n):
             return L_alfb[n]

In [15]: def codifica(text,alfb):
             L = list(text)
             L1 = map(ord2,L)
             m,i,base = 0,0,len(alfb)
             for j in L1:
                 m += j*base^i
                 i += 1
             return m

In [16]: m = codifica(HH,alfb); print m
```

```
87934427997138181416283182592012173649
```

```
In [17]: gcd(m,n) #Debe ser 1 para que valga el teorema de Fermat-Euler

Out[17]: 1

In [18]: m_encript = power_mod(m,clave_pr,n); print m_encript
```

```
15390135153514114561888543326240082067816387637717477392742050075033356979261 2
```

El mensaje que B envía al profesor contiene dos líneas: la primera es $C1$ y la segunda es *m_encript*.

0.3 3 Comprobación de la firma

Cuando recibo el mensaje, en primer lugar debería desencriptarlo con mi clave privada si estuviera encriptado. Una vez hecho ésto, para comprobar la identidad de B debo desencriptar la segunda línea del mensaje usando el e de la clave pública de B como exponente. En esta parte estamos usando que e, que forma parte de la clave pública de B, y d, que es su clave privada, juegan un papel simétrico en el punto 12 de la explicación del método RSA.

```
In [19]: m2 = power_mod(m_encript,SOL[0],n); print m2
```

```
87934427997138181416283182592012173649
```

```
In [20]: from string import join
         def descodifica(m,alfb):
             L = m.digits(base=len(alfb))
             L.reverse
             L1 = map(chr2,L)
             return  join(L1,sep = '')
```

```
In [21]: print descodifica(m2,alfb)== HH ; print descodifica(m2,alfb);print HH
```

```
True
151c517a8d1ca528e0ed4c4870e87224
151c517a8d1ca528e0ed4c4870e87224
```

Si ahora calculo el *hash* md5 de la primera línea del mensaje, es decir de $C1$, obtengo HH, que es igual al resultado de desencriptar la segunda línea. He comprobado entonces que quién me envió el mensaje conoce la clave privada de B y debo suponer que es B. Además, dado que el *hash* de la primera línea del mensaje coincide con el hash que he obtenido al desencriptar la segunda tengo la *casi seguridad* de que el mensaje no ha sido alterado al ser trasmitido por la red.

```
In [ ]:
```