MM2015: Proyecto RSA

Para entregar el Jueves, Nov 11, 2009

H. Villafuerte

Carlos E. López Camey

Fase de experimentación/Conocimiento básico

(c) Factorizar 18 y 27 en primos, qué podemos decir de gcd(18,27)?

```
sage: factor(18)
2 * 3^2
sage: factor(27)
3^3
```

Vemos que los dos tienen como factor común el 3, de hecho, 3^2. Podemos decir que gcd(18,27) = 9 precisamente, porque ese es el mayor divisor que tienen en común.

```
sage: gcd(18,27)
9
```

(d) Factorizar 11 y 17 en primos, qué podemos decir de gcd(11,17)?

```
sage: factor(11)
11
sage: factor(17)
17
```

Nos damos cuenta que 11 y 17 no tienen descomposición en factores, es decir que no tienen ningún factor en común exceptuando al 1.

```
sage: gcd(11,17)
1
```

(f) Factorizar 32 y 45 en primos, qué podemos decir de gcd(32,45)?

```
sage: factor(32)
2^5
sage: factor(45)
3^2 * 5
```

Nos damos cuenta (de nuevo) que 32 y 45 no tienen ningún factor en común. En efecto,

```
sage: gcd(32,45)
1
```

Generamos nuestra llave pública

Encontremos dos primos p_1 y p_2

```
#find primes
p1set = False;  p2set = False
reversedPrimes = list(reversed(list(primes(80)))) #len(str(2^80)) is large enough
i = 0
while not p2set:
    prime = reversedPrimes[i]
    pnumber = 2**prime -1

    if is_prime(pnumber):
        if p1set:
            p2 = pnumber
            print 'retorno'
            p2set = True
        else:
            p1 = pnumber
            p1set = True
    i+=1
```

Obtenemos $p_1 = 2305843009213693951$ y $p_2 = 2147483647 \implies n = 4951760154835678088235319297$

Encontremos un $e \in \mathbb{N}$ tal que $gcd(e, \phi(n)) = 1$

Aquí, $\phi(n)$ representa al número de co-primos de n menores que el mismo también llamada función phi de Euler. En sage, esta está implementada como

```
euler_phi(n)
```

Recordemos que $gcd(p, q) = k$ se refiere al número k mayor posible que es divisor de p y de q al mismo tiempo, es decir, al máximo común divisor.

```
def get_e(phi):
    e = 2
    while gcd(e,phi)!=1:
        e +=1
    return e
```

Que, obteniendo un (cualquiera) e que cumpla para $phi(n) = 4951760152529835076874141700$, tenemos $e = 17$

Llave pública

Nuestra llave pública es entonces, la pareja ordenada $(n, e) = (4951760154835678088235319297, 17)$

Encontrando nuestra llave privada

Encontrar $d \in \mathbb{Z}$, tal que $d \cdot e = 1(\mod \phi(n))$

```
def get_d(e,phi):
    return inverse_mod(e,phi)
```

Obtenemos entonces, $d = 40779201256128053574257637\,53$

Llave privada

Nuestra llave privada es, la tripleta $(p_1, p_2, d) = (2305843009213693951, 2147483647, 4077920125612805357425763753)$

Como encriptamos y desencriptamos

Para un entero $m < n$, encriptar usando $c = m^e(\mod n)$, i.e.

```
def get_rsa_number_encryption(m,publicKey):
    e = publicKey[1]
    n = publicKey[0]
    return power_mod(m,e,n)
```

Desencriptamos c usando $m = c^d(\mod n))$, i.e.

```
def get_rsa_number_decryption(c,privateKey):
    n = privateKey[0]*privateKey[1]
    d = privateKey[2]
    return power_mod(c,d,n)
```

Encriptando una palabra

La palabra que se encriptará es ”PUCHICA”

Codificación

La convención que se uso es que para codificar la palabra a un número para que pueda ser encriptado, usamos ASCIIs. En efecto, la palabra ”ABC”, sería codificada por el número 656667.

Para codificar, usamos

```
def message_to_ascii_representation(message):
    ascii_rep = ""
    for character_index in range(0,len(message)):
        character = message[character_index]
        ascii_rep += str(ord(character))

    return int(ascii_rep)
```

en donde codificando ”PUCHICA”, obtenemos el número $m = 80856772736765$.

Decodificación

Análogamente a la codificación, el proceso inverso estaría implementado de la siguiente forma

```
def ascii_representation_to_message(ascii_rep):
    ascii_rep = str(ascii_rep)
    mssg = ""
    for hasta in range(0,len(ascii_rep),2):
        ord = int(ascii_rep[hasta:hasta+2])
        mssg += chr(ord)
    return mssg
```

Encriptando

Para encriptar $m = 80856772736765$, necesitaremos nuestra llave pública $(49517601548356780882353l9297, 17)$, que encriptando como describimos anteriormente, tenemos

$$c = 4175976697833683195181896730$$

Desencriptando

Para desencriptar $c = 4175976697833683195181896730$ con nuestra llave privada, tenemos

```
sage: private_key
(2305843009213693951, 2147483647, 4077920125612805357425763753)
sage: desencriptado = get_rsa_number_decryption(4175976697833683195181896730,private_key)
sage: desencriptado
80856772736765
sage: ascii_representation_to_message(desencriptado)
'PUCHICA'
```

Que es el mensaje que habíamos encriptado inicialmente.