RSA

April 5, 2019

1 RSA - *Rivest-Shamir-Adleman*

O **RSA** é um dos primeiros sistemas de criptografia de chave pública e é bastante utilizado para a transmissão segura de dados. Neste sistema, a chave de cifragem é pública e é diferente da chave de decifragem, que é secreta (privada).

1.1 SETUP

Como *input*, o algoritmo recebe um inteiro l que, neste contexto, designa-se por **parâmetro de segurança**. Começa-se por escolher, de forma aleatória, dois números primos grandes **p** e **r**, com $p > 2r > 2^{l/2}$. Tais primos fazem parte da informação classificada como privada.

De seguida, calcula-se: $q = p\ q$, designado por **módulo**, que faz parte da informação classificada como pública.

Determina-se: $\mathbf{m} = \phi(q) = (p-1)(r-1)$, classificado como privado.

Gera-se posteriomente um inteiro **k**, que se designa por **chave pública**, e que verifica: $mdc\ (k, m) = 1$. Calcula-se a inversa de k módulo m, **s**, tal que:

$$s = \frac{1}{k}\ mod\ m.$$

O algoritmo produz assim dois *outputs*: - a chave privada **s**; - a chave pública **k** e o módulo **q**.

1.2 CIFRAGEM

Para cifrar uma mensagem $\mathbf{a} \not\equiv 0\ mod\ (q)$, o algoritmo de cifra constrói um criptograma: $c = a^k\ mod\ q$.

1.3 DECIFRAGEM

Para decigrar o ciprograma controi-se: $\mathbf{a'} \equiv c^s\ mod\ q$. Deste modo, a'$\equiv\ (a^k)^s\ mod\ q$, donde a'$\equiv$ $a\ mod\ q$

1.4 ASSINATURA DIGITAL

Para assinar uma mensagem $0 < \mathbf{m} < q$, o algoritmo de assinatura constrói a assinatura: **sig** $\equiv m^s mod\ q$. E, para verificar a assinatura, sig, usa-se de novo a mensagem m e a chave pública k, e testa-se a igualdade:

$$m = sig^k\ mod\ q.$$

A mensagem é autenticada para a assinatura quando a igualdade se verifica.

```
In [25]: class RSA:
             def __init__(self, l):
                 p = random_prime(2^(floor(l/2)))
                 r = random_prime(2^(floor(l/2)))
                 q = p*r
                 m = (p-1)*(r-1)
                 k = randint(2,m)
                 while gcd(m,k)!=1:
                     k = randint(2,m)
                 s = power_mod(k,-1,m)
                 self.pubkey = (q,k)
                 self.privkey = s

             def cifra(self,a):
                 q,k = self.pubkey
                 c = power_mod(a,k,q)
                 return c

             def decifra(self, c):
                 s = self.privkey
                 q,_=self.pubkey
                 z = power_mod(c,s,q)
                 return z

             def assina(self,m):
                 sig = power_mod(m,self.privkey,self.pubkey[0])
                 return sig

             def verifica(self,m, sig):
                 m_k = power_mod(sig,self.pubkey[1],self.pubkey[0])
                 return (m_k == m)

In [26]: R = RSA(16);
         msg = 1234
         criptograma = R.cifra(msg)

In [27]: R.decifra(criptograma)

Out[27]: 1234

In [28]: sig = R.assina(msg); R.verifica(msg,sig)

Out[28]: True
```