RSA

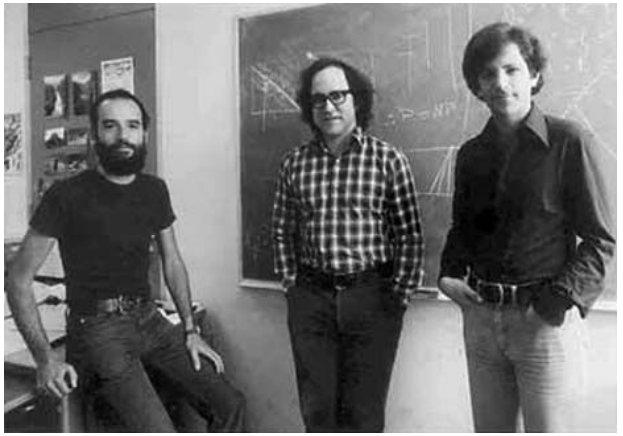
Canek García (kaan.ek@ciencias.unam.mx)

Agenda

- Breve historia
- Algoritmo
- Implementación

Breve historia

- RSA=Rivest, Shamir y Adleman
- Publicado en 1977, en el MIT.
- La seguridad de este algoritmo radica en el problema de la factorización de números enteros.
- Los mensajes enviados se representan mediante números y el funcionamiento se basa en el producto de dos números primos grandes elegidos al azar y mantenidos en secreto.
- Sistema de clave pública.



Algoritmo

Partes del algoritmo

El algoritmo consta de tres pasos:

1. **Generación de claves**
2. **Cifrado**
3. **Descifrado**

Generación de claves

1. Cada usuario elige dos números primos distintos **p** y **q** aleatorios.
2. Se calcula:

   $$n = p \cdot q$$

   donde **n**, se usará como módulo para la clave pública y privada.
3. Se calcula la función $\varphi(n)$ de Euler:

   $$\varphi(n) = (p-1) \cdot (q-1)$$

4. Se escoge un entero positivo **e** menor que $\varphi(n)$, que sea coprimo con $\varphi(n)$ **e** se da a conocer como exponente de la **clave pública**.

5. Se determina un **d**, que satisfaga la congruencia:

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

i.e., d congruente con e modulo inverso de $\varphi(n)$

d se guarda como el exponente de la clave privada.

6. La clave pública es: (n, e)

La clave privada es: (n, d)

Cifrado

Se calcula el texto cifrado **C**, mediante la operación:

$$c \equiv m^e \pmod{n}$$

Descifrado

Se recupera el mensaje original calculando:

$$m \equiv c^d \pmod{n}$$

Implementación

Java

Generación de llaves:

```java
public RSA() {
    // 1. Dos números primos aleatorios.
    r = new Random();
    p = BigInteger.probablePrime(bitlength, r);
    q = BigInteger.probablePrime(bitlength, r);

    // 2. Se calcula n=p*q
    N = p.multiply(q);

    // 3. Se calcula la función de Euler
    // Phi(n) = (p-1)(q-1)
    phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

    // 4. Entero positivo 'e' menor que phi y que sea coprimo
    e = BigInteger.probablePrime(bitlength/2, r);

    while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0 ) {
        e.add(BigInteger.ONE);
    }

    // 5. Se determina 'd' que satisfaga la congruencia e*d=1(mod Phi(n))
    d = e.modInverse(phi);
}
```

Cifrado:

```
/**
 * c = c.pow(d) mod n
 * @param message
 * @return
 */
public byte[] encrypt(byte[] message) {
    return (new BigInteger(message)).modPow(e, N).toByteArray();
}
```

Descifrado:

```
/**
 * m = c.pow(d) mod n
 * @param message
 * @return
 */
public byte[] decrypt(byte[] message) {
    return (new BigInteger(message)).modPow(d, N).toByteArray();
}
```

Código fuente:

https://github.com/kanekko/cryptography-and-security/blob/master/Lab04_RSA/RSA.java