

# El cifrado RSA:

Ulkei Szabolcs

D210871

## Descripción

El cifrado RSA es un cifrado assymetrico en sentido en que tiene dos claves separados: una publica y una secreta. Como sugieren los nomres la clave publica esta accessible por todo mundo y la clave secreta esta guardada en privat.

La idea datras del algorithmo RSA esta que esta mut dificil de hacer un factorizacion del un numero entero grande. Esto es important por que una del claves publicas esta el producto del dos primos mayores. Entonces si alguien puede factorizar el clave publica puede calcular facil la clave secreta tambien.

Mencionando estos esta bastante claro que la principal factor del seguridad en RSA esta bazando en el tamano del numeros primos. Si mayoremos los numeros la fuerta del cifrado crece exponencial.

El algoritmo tiene 3 pasos principales:

### **1. Generar la clave publica**

### >> Generating Public Key :

- Select two prime no's. Suppose  $P = 53$  and  $Q = 59$ .  
Now First part of the Public key :  $n = P*Q = 3127$ .
- We also need a small exponent say  $e$  :  
But  $e$  Must be
  - An integer.
  - Not be a factor of  $n$ .
  - $1 < e < \phi(n)$  [ $\phi(n)$  is discussed below],  
Let us now consider it to be equal to 3.
- Our Public Key is made of  $n$  and  $e$

```
int KeyGeneration(){
    int p, q;
    int phi_n;

    do{
        do
            p = rand();
        while (p % 2 == 0);
    }
    while (!PrimarityTest(2, p));

    do{
        do
            q = rand();
        while (q % 2 == 0);
    }
    while (!PrimarityTest(2, q));

    n = p * q;
    phi_n = (p - 1) * (q - 1);

    do
        e = rand() % (phi_n - 2) + 2; // 1 < e < phi_n
    while (gcd(e, phi_n) != 1);

    d = inverse(phi_n, e);
}
```

## 2. Generar la clave privada/ secreta

### >> Generating Private Key :

- We need to calculate  $\Phi(n)$  :  
Such that  $\Phi(n) = (P-1)(Q-1)$   
so,  $\Phi(n) = 3016$
- Now calculate Private Key,  $d$  :  
 $d = (k \cdot \Phi(n) + 1) / e$  for some integer  $k$   
For  $k = 2$ , value of  $d$  is 2011.

Junto con el punto 1.

```
int inverse(int a, int b){
    int inv;
    int q, r, r1 = a, r2 = b, t, t1 = 0, t2 = 1;

    while (r2 > 0){
        q = r1 / r2;
        r = r1 - q * r2;
        r1 = r2;
        r2 = r;

        t = t1 - q * t2;
        t1 = t2;
        t2 = t;
    }

    if (r1 == 1){
        inv = t1;
    }

    if (inv < 0){
        inv = inv + a;
    }

    return inv;
}
```

## 3. Cifrar/ Descifrar

- Convert letters to numbers : H = 8 and I = 9

- Thus Encrypted Data  $c = 89^e \bmod n$ .

Thus our Encrypted Data comes out to be 1394

Now we will decrypt 1394 :

- Decrypted Data  $= c^d \bmod n$ .

Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

```
int Encryption(int value, FILE *out){
    int cipher;
    cipher = FindT(value, e, n);
    fprintf(out, "%d ", cipher);
}

int Decryption(int value, FILE *out){
    int decipher;
    decipher = FindT(value, d, n);
    fprintf(out, "%c", decipher);
}
```

```
int FastExponentiation(int bit, int n, int *y, int *a){
    if (bit == 1){
        *y = (*y * (*a)) % n;
    }

    *a = (*a) * (*a) % n;
}

int FindT(int a, int m, int n)
{
    int r;
    int y = 1;

    while (m > 0)
    {
        r = m % 2;
        FastExponentiation(r, n, &y, &a);
        m = m / 2;
    }
    return y;
}
```

## Principales ataques

Desde cuando tenemos el cifrado RSA muchas gentes han intentado encontrar una manera de atacar. Son 4 categorías principales:

1. Ataques elementarios basados en uso incorrecto del Sistema
  - a. Factorización:

- Si alguien resuelva la factorización del clave pública, el atacante puede calcular fácilmente  $\phi(N)$  y  $d$  también.
- Pero si RSA está usando una manera apropiada de este tipo de ataque no hace un riesgo grande.
- El algoritmo más rápido ahora es el General Number Field Sieve
- $N(o) = ((c + o(1))n^{1/3} \log^{2/3} n)$

#### b. Módulo común

- En este caso el ataque es interno. Si un usuario tiene las claves  $(N, e)$  y  $(N, d)$  puede factorizar más fácilmente  $N$  haciendo uso de su exponente.
- Esta observación de Simmons nos muestra que un módulo es más seguro si está usando solo una entidad.

#### c. Blinding

- En este caso un usuario tiene que cifrar un mensaje  $M \in \mathbb{Z}_N^*$  que lo sabe el atacante. Si hace esto el usuario, el atacante puede obtener la clave secreta del usuario:

$$S^c = (S')^c / Y^c = (M')^{cd} / Y^c \equiv M' / Y^c = M \pmod{N}$$

### 2. Exponente menor privada

- Esto es el más serio ataque por el sistema RSA. Si tenemos un  $d$  menor esto puede romper la seguridad del sistema totalmente.
- Let  $N = pq$  with  $q < p < 2q$ . Let  $d < 1/3 N^{1/4}$

$$\left| \frac{e}{\phi(N)} - \frac{k}{d} \right| = \frac{1}{d\phi(N)}$$

- Since  $\phi(N) = N - p - q + 1$  and  $p + q - 1 < 3\sqrt{N}$  an attacker can use  $N$  to approximate  $\phi(N)$ .
- Para evitar esto en caso en que  $N$  tiene 1024 bits,  $d$  necesita la menor 256 bits.

### 3. Exponente menor pública

- Para reducir el tiempo de cifrar y descifrar una exponente pública menor está usando. La recomendación por  $e$  es  $e = 2^{16} + 1$ . En este caso necesita solo 17 multiplicaciones para verificar la firma en lugar de 1000+ si  $e$  está aleatorio.
- Pero en este caso los ataques aplicados cuando  $e$  es menor son bastante complejos y no rompen el sistema como hacen los ataques con exponente menor privada.
- Teoremas para atacar en esta manera son:
  1. Coppersmith theorem
  2. Hastad's broadcast attack
  3. Franklin-Reiter Related Message Attack
  4. Partial Key Exposure attack

#### 4. Ataque del implementacion

- Son totalmente dependiente del implementacion del sistema.
- En casos en que RSA esta implementado correctamente son dos metodos principals del ataques del implementacion:

##### 1. Timing attacks

- Let  $d = d_n d_{n-1} \dots d_0$  (binary of  $d$ )
- Set  $z = M$  and  $C = 1$ . For  $i=0 \dots n$  do:
  - (1) if  $d_i = 1$  set  $C = C \cdot z \bmod N$
  - (2)  $z = z^2 \bmod N$
- $C$  at the end has the value  $M^d \bmod N$

##### 2. Random Faults

$$C_p = M^{d_p} \bmod p \quad \text{and} \quad C_q = M^{d_q} \bmod q$$

Where  $d_p = d \bmod (p-1)$  and  $d_q = d \bmod (q-1)$ .

then  $C = T_1 C_p + T_2 C_q \pmod{N}$

where

$$T_1 = \begin{cases} 1 \bmod p \\ 0 \bmod q \end{cases} \quad \text{and} \quad T_2 = \begin{cases} 0 \bmod p \\ 1 \bmod q \end{cases}$$

#### Referencias:

<https://www.utc.edu/sites/default/files/2021-04/course-paper-5600-rsa.pdf>

<https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>

<https://www.educba.com/rsa-algorithm/>

