RSA: Generar claves

Grupo 13: Ander Alonso, Adrian San Segundo, Daniel Ruskov

20 de diciembre de 2017

Introducción

El método de encriptado de datos conocido como algoritmo RSA, por los nombres de sus inventores (Rivest, Shamir y Adleman),es uno de los más usados para la transmisión segura de datos. La aritmética modular es la base matemática para al algoritmo de encriptado. Veamos cómo se pueden usar los resultados de la aritmética modular para la construcción de un sistema que permita la comunicación segura entre una persona emisora, que llamaremos E, y un una persona receptora, R. El objetivo es que cuando E envie un mensaje cifrado a R, este tenga la seguridad de que nadie ha podido descifrarlo. Este método se caracteriza por el empleo de dos claves: Una para cifrar, clave pública, y otra para descifrar, clave privada. La clave de cifrado no debe poder ser obtenida a partir de la clave de descifrado.

Desarrollo del algoritmo RSA

```
> library(numbers)
```

1. Elegir un mensaje en claro y codificarlo

```
>   mensaje <- "Este es el mensaje"
>   codificar<-function(mensaje){
+   return(strtoi(charToRaw(mensaje), 16L))
+ }
>   pcodificada<-codificar(mensaje)
>   print(pcodificada)
```

```
 [1]  69 115 116 101  32 101 115  32 101 108  32 109 101 110 115  97 106 101
```

2. Generar una clave pública y una clave privada.

   (a) Calcular n = pq y phi(n) = (p - 1)(q - 1) = n + 1 - (p + q).

```
> Generar_primos <- function(){
+     return(sample(Primes(400,5000),2, replace = FALSE))
+   }
>  Primos<-Generar_primos()
>  p<-Primos[1]
>  print(p)
```

```
[1] 1229

> q<-Primos[2]
> print(q)

[1] 3943

> n<-p*q
> print(n)

[1] 4845947

> phiN<-(p-1)*(q-1)
> print(phiN)

[1] 4840776
```

(b) Seleccionar un entero impar aleatorio e tal que max(p, q) < e < phi(n) mcd(e, phi(n) = 1).

```
> e<-sample(max(p,q):phiN,1)
> print(e)

[1] 3175731

> if (e%%2 == 0){
+   e<-e+1
+ }
> while(!coprime(e,phiN)){
+   e<-e+2
+ }
> print(e)

[1] 3175733
```

(c) d = e-1 mod phi(n).

```
> d <- modinv(e,phiN)
> print(d)

[1] 96845
```

(d) Clave pública (n, e). Clave privada (p, q, d).

```
> clave_Publica <- c(n,e)
> clave_Privada <- c(n,d)
> print(clave_Privada)

[1] 4845947   96845

> print(clave_Publica)

[1] 4845947 3175733
```

3. Cifrar el mensaje codificado en el apartado 1.

```
> cifrar<- function(mensaje,clave_Publica){
+  return(modpower(mensaje, clave_Publica[2],clave_Publica[1]))
+
+ }
> mensaje_cifrado<-cifrar(pcodificada,clave_Publica)
> print(mensaje_cifrado)

 [1] 3935359 1655987 3928257 2662047  673902 2662047 1655987  673902 2662047
[10] 2915682  673902 1875536 2662047 3842224 1655987 4452451 1561914 2662047
```

4. Descifrar el mensaje cifrado en el apartado 3.

```
> descifrar<- function(mensaje,clave_Privada){
+  return(modpower(mensaje, clave_Privada[2],clave_Privada[1]))
+
+ }
> mensaje_descifrado<-cifrar(mensaje_cifrado,clave_Privada )
> print(mensaje_descifrado)

 [1]  69 115 116 101  32 101 115  32 101 108  32 109 101 110 115  97 106 101
```

5. Decodificar el mensaje descifrado en el apartado 4.

```
> decodificar <- function(mensaje){
+   return(rawToChar(as.raw(mensaje)))
+ }
> mensaje_decodificado<-decodificar(mensaje_descifrado)
> print(mensaje_decodificado)

[1] "Este es el mensaje"
```

Conclusión

La seguridad de este algoritmo radica en el problema de factorizar números enteros. Al ser una practica, los numeros elegido para la realización del RSA son relativamente cortos, entre 400 y 5000. En la práctica, los números primos elegidos son de tal longitud, del orden de 200 cifras, que a un ordenador convencional le resulta muy dificil poder descifrarlo, de ahí que el RSA se tan seguro. Se prevé que el número de cifras aumente debido al aumento de la capacidad de cálculo de los ordenadores convencinales.