

LABORATORIO 6: HARD PROBLEMS IN CRYPTOGRAPHY II

Torres Olivera Karla Paola

Escuela Superior de Cómputo
Instituto Politécnico Nacional, México
kpto997@gmail.com

Resumen: El presente trabajo consiste en la descripción e implementación en Python de una manera en que se puede encontrar la llave privada en RSA, d , si se conoce la llave pública (e, n) haciendo uso del programa desarrollado en la práctica anterior encargado de encontrar factores primos de un número compuesto, así como de la librería criptográfica *PyCryptodome* para calcular el inverso modular.

Palabras Clave: número compuesto, inverso multiplicativo.

1 Introducción

Los algoritmos asimétricos o de clave pública utilizan dos parejas de claves, una pública conocida por todo el mundo y otra privada, que debe ser conocida únicamente por su propietario. Con este tipo de algoritmos se puede tanto cifrar información como firmarla [1].

Después de que Whitfield Diffie y Martin Hellman introdujeran la criptografía de clave pública en su histórico documento de 1976, de repente se abrió una nueva rama de la criptografía. Como consecuencia, se comenzaron a buscar métodos con los que se pudiera realizar el cifrado de clave pública. En 1977, Ronald Rivest, Adi Shamir y Leonard Adleman propusieron un esquema que se convirtió en el esquema criptográfico asimétrico más utilizado, RSA [2].

La función unidireccional subyacente de RSA es el problema de factorización de enteros: multiplicar dos números primos grandes es computacionalmente fácil, pero factorizar el producto resultante es muy difícil [2].

Tomando en cuenta lo anterior, en esta práctica se mostrará una forma que permite encontrar la llave privada en RSA, a partir de la llave pública. Para la parte de pruebas, se trabajará con números pequeños, por lo que el programa los resolverá rápido. Adicional a esto, se hará uso de una librería criptográfica y de un programa desarrollado en la práctica anterior.

Contents

1	Introducción	1
2	Conceptos básicos	3
2.1	Generación clave en RSA	3
2.1.1	Ejemplo de la generación de claves en RSA	4
3	Experimentación y Resultados	5
3.1	Algoritmo para encontrar la llave privada en RSA	5
3.1.1	Ejemplo del funcionamiento del pseudocódigo	6
3.2	Implementación del algoritmo	7
3.2.1	Resultados obtenidos para cada uno de los casos	7

2 Conceptos básicos

2.1 Generación clave en RSA

Una característica distintiva de todos los esquemas asimétricos es que hay una fase de configuración durante la cual se calculan las claves pública y privada. Dependiendo del esquema de clave pública, la generación de claves puede ser bastante compleja [2].

Los pasos necesarios para calcular la clave pública y privada para un criptosistema RSA se muestran en el pseudocódigo Algorithm 1, el cual se obtuvo de [2].

Algorithm 1 Generación de claves RSA

Result: Llave pública: $k_{pub} = (n, e)$ y llave privada: $k_{pr} = (d)$

- 1: Seleccionar dos números primos grandes p y q
- 2: Calcular $n = p \cdot q$
- 3: Calcular $\Phi(n) = (p - 1)(q - 1)$
- 4: Seleccionar el exponente público $e \in \{1, 2, \dots, \Phi(n) - 1\}$ tal que

$$\gcd(e, \Phi(n)) = 1$$

- 5: Calcular la llave privada d tal que

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

La condición $\gcd(e, \Phi(n)) = 1$ asegura que exista el inverso de e módulo $\Phi(n)$, de modo que siempre haya una clave privada d [2].

Dos partes de la generación de claves no son triviales: la primera corresponde al paso 1, en el que se eligen los dos números primos grandes y, la segunda abarca los pasos 4 y 5 en los que se calculan las claves pública y privada. El cálculo de las llaves d y e se puede hacer en una sola iteración haciendo uso del algoritmo extendido de Euclides. En la práctica, a menudo se comienza seleccionando primero el parámetro público e en el rango de $0 < e < \Phi(n)$. El valor de e debe satisfacer la condición $\gcd(e, \Phi(n)) = 1$. Se aplica el algoritmo extendido de Euclides con los parámetros de entrada $\Phi(n)$ y e , obteniendo la relación:

$$\gcd(\Phi(n), e) = s \cdot \Phi(n) + t \cdot e$$

Si $\gcd(e, \Phi(n)) = 1$, se puede concluir que e es una clave pública válida. Además, también se sabe que el parámetro t calculado por el algoritmo extendido de Euclides es el inverso de e , y por lo tanto:

$$d = t \pmod{\Phi(n)}$$

En caso de que e y $\Phi(n)$ no sean primos relativos, se selecciona un nuevo valor para esta variable y se repite el proceso [2].

2.1.1 Ejemplo de la generación de claves en RSA

Alice quiere mandar un mensaje cifrado a Bob. Bob primero calcula sus parámetros RSA mostrados en el pseudocódigo Algorithm 1. Luego le envía a Alice su clave pública para que cifre el mensaje con esta. Alice envía el mensaje resultante y Bob lo descifra usando su clave privada [2].

Alice



Bob



mensaje $x = 4$

$$y = x^e \equiv 4^3 \equiv 31 \pmod{33}$$

$$\xleftarrow{k_{pub}=(33,3)}$$

$$\xrightarrow{y=31}$$

1. Selecciona $p = 3$ y $q = 11$
2. $n = p \cdot q = 33$
3. $\Phi(n) = (3 - 1)(11 - 1) = 20$
4. Selecciona $e = 3$
5. $d \equiv e^{-1} \equiv 7 \pmod{20}$

$$y^d = 31^7 \equiv 4 = x \pmod{33}$$

Es importante tener en cuenta que los exponentes privados y públicos cumplen la condición $e \cdot d = 3 \cdot 7 \equiv 1 \pmod{\Phi(n)}$ [2].

3 Experimentación y Resultados

3.1 Algoritmo para encontrar la llave privada en RSA

En la sección 2.1 se explicaron los pasos a seguir para la generación de las claves pública y privada en RSA. Pero, ¿cómo se puede calcular el valor de esta última a partir de la primera?. Antes de presentar el pseudocódigo que se utilizó para la implementación se explicará brevemente el análisis realizado.

En la parte teórica se vio que la clave privada, es igual a: $d = t \bmod \Phi(n)$, donde t es el inverso de e módulo $\Phi(n)$. Partiendo de esto, lo primero que se debe hacer es calcular el valor del módulo, es decir, $\Phi(n)$.

Las variables que nos proporciona la llave pública son: e y n . Del paso 2 de la generación de claves (véase Algorithm 1), tenemos que $n = p \cdot q$, donde p y q son números primos. Adicional a esto, estas variables también se utilizan en el paso 3 para el cálculo de $\Phi(n)$, por lo que es necesario obtener sus valores. Por suerte, en la práctica anterior se implementó un programa que encuentra los factores primos de un número compuesto, cubriendo así este paso.

Como resultado de lo anterior se obtendrán los valores correspondientes a p y q . Con esto, el siguiente paso consiste en calcular $\Phi(n)$. Finalmente, se hará uso del algoritmo extendido de Eculides, para calcular el inverso de e módulo $\Phi(n)$, obteniendo así el valor de la llave privada d .

El pseudocódigo diseñado se muestra en Algorithm 2.

Algorithm 2 Obtención de la llave privada a partir de la pública en RSA

Result: Llave privada: $k_{pr} = (d)$

Require: Llave pública: $k_{pub} = (n, e)$

- 1: Obtener factores primos de n ▷ Valores de p y q
 - 2: Calcular $\Phi(n) = (p - 1)(q - 1)$
 - 3: Calcular el inverso de e módulo $\Phi(n)$ ▷ Algoritmo extendido de Euclides
-

3.1.1 Ejemplo del funcionamiento del pseudocódigo

Con la finalidad de mostrar cómo funciona el pseudocódigo mostrado en Algorithm 2, se retomará el ejemplo mostrado en la sección 2.1.1.

Tenemos que Alice desea mandarle un mensaje cifrado a Bob, por lo que este último realiza el procedimiento de generación de llaves pública y privada de RSA.

Alice



Bob



mensaje $x = 4$

1. Selecciona $p = 3$ y $q = 11$
2. $n = p \cdot q = 33$
3. $\Phi(n) = (3 - 1)(11 - 1) = 20$
4. Selecciona $e = 3$
5. $d \equiv e^{-1} \equiv 7 \pmod{20}$

Siguiendo los pasos del pseudocódigo tenemos:

1. Se obtienen los factores primos de n haciendo uso del programa implementado en la práctica anterior: $n = 33 = 3 \cdot 11 \Rightarrow p = 3$ y $q = 11$
2. Se calcula $\Phi(n) = (p - 1)(q - 1) = (3 - 1)(11 - 1) = 2 \cdot 10 = 20$
3. Se calcula el inverso de e módulo $\Phi(n)$ con ayuda del algoritmo extendido de Euclides o alguna función implementada en una librería criptográfica
 $e^{-1} \pmod{\Phi(n)} = 3^{-1} \pmod{20} = 7$

Comprobando tenemos: $(3 \cdot 7) \pmod{20} = 21 \pmod{20} = 1$

3.2 Implementación del algoritmo

Para encontrar el inverso de e módulo $\Phi(n)$ se hizo uso de la función `inverse` que proporciona la librería criptográfica *PyCryptodome*.

`Crypto.Util.number.inverse(u, v)`

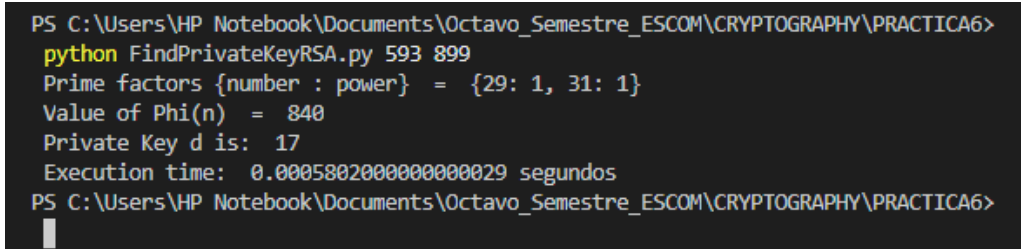
Calcula el inverso de $u \bmod v$

3.2.1 Resultados obtenidos para cada uno de los casos

1. Llave pública de mi compañera Paola Raya $(e, n) = (593, 899)$

Para este caso la respuesta obtenida por el programa fue $d = 17$ con un tiempo de ejecución de 0.00058 segundos. La comprobación la realice de manera manual, primero multiplique el valor de e y d , posteriormente con ayuda de una calculadora online saque el módulo del resultado con $\Phi(n)$ concluyendo que el procedimiento se realizó de forma correcta.

El programa en ejecución se puede observar en la Figura 1.



```
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
python FindPrivateKeyRSA.py 593 899
Prime factors {number : power} = {29: 1, 31: 1}
Value of Phi(n) = 840
Private Key d is: 17
Execution time: 0.000580200000000029 segundos
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
```

Figura 1: Programa en ejecución para el inciso 1

2. Llave pública de mi compañero Luis Varela $(e, n) = (34567, 23264323)$

Para el inciso 2 la respuesta obtenida por el programa fue $d = 8,616,903$ con un tiempo de ejecución de 0.0021 segundos. Como se puede observar este último dato no aumento mucho, sin embargo, esto se debe a que se sigue trabando con valores un tanto pequeños.

Por otro lado, el valor de $\Phi(n)$ fue 23,254,000. La comprobación al igual que en el caso anterior la realice de manera manual, con lo que pude concluir que el programa dio los resultados correctos. El programa en ejecución se puede observar en la Figura 2.

```

PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
python FindPrivateKeyRSA.py 34567 23264323
Prime factors {number : power} = {3323: 1, 7001: 1}
Value of Phi(n) = 23254000
Private Key d is: 8616903
Execution time: 0.00211199999999996 segundos
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>

```

Figura 2: Programa en ejecución para el inciso 2

3. Llave pública de mi compañero Héctor Hernández $(e, n) = (1283, 4258141)$

Para este inciso los resultados obtenidos fueron $d = 3,159,839$, $p = 1,987$ y $q = 2,143$ con un tiempo de ejecución de 0.0030 segundos. A pesar de que el valor de n para este caso es mucho menor a comparación del inciso anterior, el tiempo de ejecución aumentó, esto se puede deber a que estaba utilizando otros programas mientras se realizaba el cálculo.

Por otro lado, se encontró que $\Phi(n) = 4,254,012$. Con ayuda de la comprobación que realice manualmente, pude concluir que los cálculos también se realizaron de forma correcta para este caso. El programa en ejecución se puede observar en la Figura 3.

```

PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
python FindPrivateKeyRSA.py 1283 4258141
Prime factors {number : power} = {1987: 1, 2143: 1}
Value of Phi(n) = 4254012
Private Key d is: 3159839
Execution time: 0.003068599999999977 segundos
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>

```

Figura 3: Programa en ejecución para el inciso 3

4. Llave pública de mi compañero Luis A. $(e, n) = (249617, 555911)$

Para el inciso 4 la respuesta obtenida por el programa fue $d = 498,353$ con un tiempo de ejecución de 0.00076 segundos. La comprobación la realice nuevamente de manera manual, para esto hice uso del valor de $\Phi(n) = 554,400$ y de una calculadora online para el módulo.

Los valores para p y q fueron 631 y 881 respectivamente. El programa en ejecución se puede observar en la Figura 4.


```

PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
python FindPrivateKeyRSA.py 249617 555911
Prime factors {number : power} = {631: 1, 881: 1}
Value of Phi(n) = 554400
Private Key d is: 498353
Execution time: 0.000766599999999992 segundos
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>

```

Figura 4: Programa en ejecución para el inciso 4

5. Llave pública de mi compañero Nicolas Sánchez $(e, n) = (32563, 16453868461)$

Finalmente tenemos el inciso 5, el valor obtenido de la clave privada fue 14,797,786,555 con $\Phi(n) = 16,453,606,624$, $p = 104,729$ y $q = 157,109$. Esta cifra fue la más grande que se probó, a pesar de esto, el tiempo de ejecución no aumentó drásticamente, con 0.0166 segundos.

La comprobación al igual que en todos los casos anteriores la realice de manera manual, concluyendo que el programa funciona de manera correcta. El programa en ejecución se puede observar en la Figura 5.

```

PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>
python FindPrivateKeyRSA.py 32563 16453868461
Prime factors {number : power} = {104729: 1, 157109: 1}
Value of Phi(n) = 16453606624
Private Key d is: 14797786555
Execution time: 0.016633299999999997 segundos
PS C:\Users\HP Notebook\Documents\Octavo_Semestre_ESCOM\CRYPTOGRAPHY\PRACTICA6>

```

Figura 5: Programa en ejecución para el inciso 5

References

- [1] J. L. B. Gómez, *UF1846 - Desarrollo de aplicaciones web distribuidas*. Arganda del Rey, Madrid: Paraninfo, 2016.
- [2] C. Paar and J. Pelzl, *Understanding Cryptography*. London New York: Springer, 2010.