Criptografía

ETSInf - UPV

Práctica RSA

Cifrado y criptoanálisis RSA

---

1. Introducción

El primer objetivo de esta práctica es que el alumno implemente un sistema de cifrado de clave pública considerando todas las fases, desde la generación de los elementos de la clave, a los procesos de cifra y descifrado de los mensajes.

El método de cifrado escogido es el conocido sistema RSA. La sencillez del sistema lo hace perfecto para ser simulado en el laboratorio.

El segundo objetivo de la práctica es la implementación de herramientas básicas para el ataque a sistemas cuya seguridad se basa en el problema de la factorización de enteros (como el RSA).

2. Generación de números primos

Los números primos tienen un papel importante en el diseño de sistemas criptográficos de clave pública. La implementación de herramientas para la elección de números primos *aleatorios* y de cierto tamaño es importante pero costosa, por lo que normalmente se recurre a tests que determinan, con cierto grado de seguridad, si un número es compuesto o no.

Uno de estos test de *pseudo-primalidad* se basa en el *Pequeño Teorema de Fermat* que establece que, si un número p es primo, entonces, para cualquier $1 \leq a \leq p-1$ se cumple que:

$$a^{p-1} \equiv 1 \pmod{p}$$

el reverso del teorema no se cumple. No obstante, el Teorema de Fermat permite implementar un test de pseudo-primalidad.

El test de Fermat permite la implementación de un algoritmo para la búsqueda aleatoria de valores primos

3. Elección de la clave

Para la selección de la clave a utilizar en esta práctica se recomienda seguir el siguiente esquema:

1. Seleccionar aleatoriamente dos números primos p y q grandes (del orden de 100 cifras c.u.)

INPUT: Un número natural $n > 1$ (potencialmente primo)
INPUT: Un número k (que determina la fiabilidad del test)
OUTPUT: *compuesto* si n no es primo, o *posible primo* si n cumple el test.

Método
Para $i = 1$ to k **hacer:**
Escoger a aleatorio tal que $1 < a < n$
Si $a^{n-1} \not\equiv 1 \pmod{n}$ **entonces:**
Devolver *compuesto*
FinSi
FinPara
Devolver *posible primo*
FinMétodo.

Figura 1: Algoritmo de pseudo-primalidad de Fermat.

INPUT: El número de bits k mínimo para el número primo
INPUT: Un número t (que determina la fiabilidad del generador)
OUTPUT: Un número probablemente primo de k bits

Método
Repetir:
Generar n un número aleatorio impar entre 2^k y $2^{k+1} - 1$
Si $Fermat(n, t) ==$ *posible primo* **entonces:**
Devolver n
FinSi
end Repetir:
FinMétodo.

Figura 2: Algoritmo de generación de valores (probablemente) primos.

2. $n = pq$
3. Seleccionar un número impar e tal que $mcd(e, \phi(n)) = 1$
4. Calcular $d = e^{-1}$ mód $\phi(n)$
5. Clave pública: (n, e); Clave privada: (d, p, q)

Nótese que la selección aleatoria de los valores primos que permiten obtener el valor n puede realizarse mediante el algoritmo expuesto anteriormente.

4. Cifrado y descifrado

Nótese que tanto el proceso de cifrado como de descifrado consisten en exponenciaciones modulares. Así, dado n, el dominio es $\mathbb{Z}_n$, y para todo elemento del mensaje ($x \in \mathbb{Z}_n$)

$$e_{k_{pb}}(x) = x^e \text{ mód } n$$
$$d_{k_{pr}}(y) = y^d \text{ mód } n$$

Recordamos que la seguridad de RSA recae en el problema de factorizar un número. Si n es un valor de k bits, es necesario realizar del orden de $2^{k/2}$ divisiones. A modo ilustrativo, a un ritmo de 1 división por microsegundo, en un año pueden realizarse aproximadamente 10^{14} operaciones.

Ejercicio 1:

Implementar dos módulos *Mathematica* para el cifrado y descifrado de elementos de mensaje.

Ejercicio 2:

Diseñar un módulo *Mathematica* que, dado un mensaje y un valor n, codifique el mensaje en una secuencia de valores de $\mathbb{Z}_n$.

Funciones de *Mathematica* útiles para este proceso son:

- **IntegerDigits**, que, dado un entero cualquiera, obtiene los dígitos del entero en cualquier base, pudiendo especificar el mínimo número de dígitos de la representación.

- **Flatten**, que elimina toda estructura de la lista recibida como entrada.
- **Partition**, que divide una lista en segmentos (solapantes o no) de determinada longitud.
- **FromDigits**, que dada una lista con los dígitos de un número, en una base especificada, devuelve el número en base 10.

5. Criptoanálisis

Como se ha comentado, a seguridad de RSA se basa en la complejidad del problema de descomponer un número en factores primos. Algunos de los algoritmos de descomposición actuales se basan en la búsqueda de valores x e y tales que sus respectivos cuadrados son congruentes módulo n, el valor a descomponer. En esta sección se exponen distintos algoritmos de factorización describiendo brevemente sus características.

5.1. Factorización por divisiones sucesivas

La factorización por divisiones sucesivas de un número n necesita a lo sumo $\sqrt{n}$ divisiones ya que en el peor de los casos los factores seran próximos a la raíz del número. La factorización por divisiones sucesivas habitualmente encuentra factores pequeños (en caso de existir) de forma rápida, utilizando una gran cantidad de tiempo en encontrar los factores mayores. Nótese que, para un valor n aleatorio, n contiene el factor 2 en un 50 % de las ocasiones, el factor 3 en un 33 % de las ocasiones, el factor 5 en un 20 % de las ocasiones, y así sucesivamente.

5.2. Factorización de Fermat

El algoritmo de Fermat considera la representación de un entero impar n como diferencia de dos cuadrados, esto es, $n = a^2 - b^2$. Aplicando álgebra básica se tiene que $n = (a + b)(a - b)$, que convierte los valores a y b en factores de n siempre que sean distintos de 1.

El algoritmo considera un valor cercano a la raiz del número a factorizar, calculando su cuadrado módulo n. El algoritmo itera mientras no se obtenga un cuadrado perfecto, lo que proporcionaría los factores de n.

INPUT: Un número entero positivo compuesto n
OUTPUT: Factores a y b de n
Método
$A = \lceil \sqrt{n} \rceil$
$B = A^2 - n$
Mientras B no sea un cuadrado perfecto **hacer:**
$A++$
$B = A^2 - n$
FinMientras
Devolver $\langle A - \sqrt{B}, A + \sqrt{B} \rangle$
FinMétodo.

Figura 3: Algoritmo de factorización de Fermat.

5.3. Algoritmo de factorización rho de Pollard

Este algoritmo se basa en la búsqueda de dos valores a y b congruentes módulo p. En caso que p sea factor del número a factorizar n, entonces este puede obtenerse como $mcd(|a - b|, n)$, ya que p divide tanto a n como a la diferencia de a y b.

El algoritmo considera una función pseudoaleatoria que hace variar los valores de a y b, aplicando la función una vez en el cálculo de uno de los números y dos veces en el cálculo del segundo. En la práctica, en caso que n posea factores pequeños, el algoritmo tiene un muy buen comportamiento.

5.4. Algoritmo de factorización $p-1$ de Pollard

El algoritmo de factorización $p-1$ de Pollard considera el caso particular que el número anterior al factor posee factores pequeños (menores que una determinada cota).

El algoritmo tiene en cuenta el Teorema de Fermat:

$$a^{K(p-1)} \equiv 1 \pmod{p},$$

junto con el hecho de que si se encuentra un número congruente con 1 módulo un factor de n, entonces $mcd(x - 1, n)$ será divisible por ese factor.

La idea es considerar como exponente un múltiplo grande de $p - 1$ considerando un conjunto de primos menores que determinada cota B, en cada

INPUT: Un número entero positivo compuesto n
OUTPUT: Un factor de n
Método
$A = 2$
$B = 2$
Mientras True **hacer:**
$A = A^2 + 1$ mód n
$B = B^2 + 1$ mód n
$B = B^2 + 1$ mód n
$p = mcd(a - b, n)$
Si $1 < p < n$ **entonces: Devolver** p
Si $p = n$ **entonces: Devolver** n
FinMientras
FinMétodo.

Figura 4: Algoritmo de factorización Pollard's rho.

iteración se considera un nuevo valor primo, comprobando si se cumplen las condiciones. En caso de alcanzar la cota, el algoritmo acaba devolviendo fallo.

5.5. Busqueda de cuadrados congruentes

Recordamos que el problema de factorización de un número n puede enunciarse como la búsqueda de factores primos de un valor n. Supongamos que se dispone de valores x e y tales que $x^2 \equiv y^2$ mód n, pero que no son congruentes ($x \not\equiv \pm y$ mód n). Si x^2 e y^2 son congruentes módulo n, entonces $x^2 - y^2 = (x + y)(x - y) = kn$, donde k es un valor entero.

Con otras palabras, n divide a $x^2 - y^2$ pero no divide a $(x+y)$ ni a $(x-y)$, por lo que, con cierta probabilidad, puede encontrarse un factor no trivial de n calculando el $mcd(x - y, n)$.

Algoritmos de descomposición en factores como el denominado algoritmo de *criba cuadrática* o el algoritmo de *criba en campos de números* se basan en la búsqueda guiada de valores x e y que cumplan las condiciones mencionadas (no congruentes módulo n pero cuyos cuadrados sí lo son). En esta sección únicamente ilustraremos como la selección aleatoria de estos valores x e y permite resolver la factorización de enteros. La implementación del algoritmo de criba cuadrática queda como ejercicio propuesto.

INPUT: Un número entero positivo compuesto n
OUTPUT: Un factor de n
Método
Seleccionar una cota B
Considerar *base* un array con los primos de B
Seleccionar A aleatoriamente en $2 \leq A \leq n-1$
$d = mcd(A, n)$
Si $d \geq 2$ **entonces: Devolver** d
Para cada primo p en *base* **hacer:**
$e = \lfloor \frac{\ln n}{\ln p} \rfloor$
$A = A^{p^e}$ mód n
$d = mcd(A-1, n)$
FinPara
Si $d = 1$ **or** $d = n$ **entonces: Devolver** $Fallo$
Devolver d
FinMétodo.

Figura 5: Algoritmo de factorización $p-1$ de Pollard.

INPUT: Un número entero positivo compuesto n
OUTPUT: Un factor no trivial de n
Método
$factor = 1$;
Mientras $factor == 1$ **hacer:**
Generar dos valores positivos a y b aleatorios menores que n
$x = a^2$ mód n
$y = b^2$ mód n
$factor = mcd(x - y, n)$
FinMientras
Devolver $factor$
FinMétodo.

Figura 6: Algoritmo de factorización mediante búsqueda aleatoria de cuadrados congruentes módulo n.

Ejercicio 3:

Implementar los algoritmos descritos en esta sección como módulos *Mathematica*. Estudiar el comportamiento comparado de estos algoritmos considerando la factorización de distintos productos de dos números primos. En esta comparativa es interesante considerar tanto el tamaño, como la proximidad de los valores primos.

Para estudiar el comportamiento comparado de estos algoritmos de factorización puede utilizarse la función **Timinig** de *Mathematica* que devuelve el tiempo utilizado en la ejecución de un comando.

Ejercicio 4:

Buscar información acerca del algoritmo de factorización mediante criba cuadrática e implementarlo como un módulo *Mathematica*. Estudiar su comportamiento comparado con el resto algoritmos expuestos.