MA-53G Intr. a la Criptografía **Semestre Otoño, 1998**

Cátedra No. 14 : 5 de Mayo

Transcriptor: Manuel Reyes[1]

1 Implementaci'on RSA

Tenemos los siguientes problemas de implementación :

1. Elegir p y q primos grandes.
2. Calcular $n = pq$ y $\varphi(n) = (p-1)(q-1)$.
3. Elegir un exponente de encriptación a tal que $0 < a < \varphi(n)$ y donde a sea primo relativo con $\varphi(n)$.
4. Encontrar $b \in$ { 1 ... φ(n)-1 } tal que: $ab \equiv 1 (\bmod\ \varphi(n))$
5. Publicar (n, a) (No publicar en cualquier Newsgroup.)
6. Calcular $x^c \bmod n$, donde c es a o b.

Las partes 2. y 3. sabemos como implementar eficientemente. En la parte 4 usamos el Algoritmo de Euclides. Para implementar 6. utilizamos el método de los cuadrados repetidos, es decir, el siguiente algoritmo:

$z = 1$
for $i = l$ downto 0
    $z = z^2$
    if $c_i = 1$ then $z = zx \bmod n$

Donde $c = c_1 ... c_l$ es la representación binaria de c.

Ejemplo: $x = 3$, $n = 5$, $c = 5 (= 101 = c_3 c_2 c_1)$

i	c_i	$z (= xz \,(\bmod n),\ if\ c_i = 1)$	$z^2 \,(\bmod n)$
		1	1
3	1	$1 \cdot 3 \,(\bmod 5) = 3$	$3^2 \,(\bmod 5) = 9 \,(\bmod 5) = 4$
2	0	4	$4^2 \,(\bmod 5) = 16 \,(\bmod 5) = 1$
1	1	$1 \cdot 3 \,(\bmod 5) = 3$	

$3^5 \,(\bmod 5) = 243 \,(\bmod 5) = 3$.

[1] La primera parte completando el cap'itulo 2 de la clase 10 de 1996, por Ricardo Valenzuela

Todos los números a multiplicar tienen $O(logn)$ bits, además en cada iteración tenemos dos multiplicaciones, por lo tanto, con l iteraciones tenemos que el algoritmo toma $O(2l(logn)^2)$ operaciones binarias.

Para implementar la parte 1, es necesario estudiar algoritmos de generaci'on de primos, equivalentemente testear si un n'umero es primo o descomponerlo.

Hoy en día sabemos factorizar números de hasta 130 dígitos. Se recomienda usar números p y q de alrededor de 100 dígitos para que n quede de alrededor de 200 dígitos. Las implementaciones en hardware de RSA que existen utilizan un módulo n de 512 bits ($\approx$ 154 dígitos). Estas implementaciones logran encriptar 600 Kbits/seg. Por otro lado DES implementado a nivel de hardware logra encriptar 1 Gbit/seg. Seg'un Schneier DES es 100 veces m'as r'apido que RSA a nivel de software y 1000 veces m'as r'apido a nivel de hardware.

Existen muchas situaciones donde se quiere saber si un número n grande es primo. Por ejemplo, en la implementación de RSA y en varios criptosistemas, necesitamos encontrar un primo elegido al azar suficientemente grande.

Un test de primalidad es una prueba probabilista para saber si un número es o no primo. Si un número pasa la prueba, éste puede ser primo. Si un número pasa una gran cantidad de estos test, entonces es muy probable que sea primo. Y por lo tanto si un número no pasa la prueba, éste es definitivamente no primo. Empíricamente, se sabe que es más costoso (en tiempo) factorizar un número grande no primo (fallo en el test de primalidad) que encontrar uno primo del mismo orden de magnitud.

La seguridad de RSA se basa en el hecho de que es mucho más fácil encontrar dos primos extremadamente grandes p y q que, conociendo $n = pq$, encontrar los dos factores p y q.

1.1 Algoritmo de b'usqueda de n'umeros primos

¿C'omo se encuentra un n'umero primo en un intervalo $\{N+1,...,2N\}$ con N un natural (N m'as o menos como 2^l).

Supongamos que tenemos un algoritmo A probabilista de tipo Montecarlo tal que

$$A(n) = \{ \begin{array}{ll} 1 & \text{si } n \text{ es primo} \\ 0 & \text{con probabilidad } \geq 1-\xi \text{ si } n \text{ es compuesto} \end{array}$$

Este algoritmo permite encontrar un n'umero primo en el conjunto $\{N+1,...,2N\}$ mediante el siguiente algoritmo de tipo Montecarlo

i) Elegir $n \in \{N+1,...,2N\}$, n impar al azar (uniformemente).

ii) Se ejecuta k veces en la entrada n el algoritmo A.

iii) Si las k ejecuciones entregan las salidas "1", entonces ser'a primo con alta probabilidad. En caso contrario, si alguna ejecuci'on responde "0", entonces ser'a compuesto.

Estimaci'on de la probabilidad de error de un algoritmo de tipo Montecarlo

Encontremos pues la probabilidad de que nuestro n'umero sea realmente primo, dado que el algoritmo respondi'o k veces "1". Definamos las afirmaciones:

E: un entero $n \in \{N+1, ..., 2N\}$, n impar es compuesto.
F: el algoritmo A entrega como salida 1, k veces.

Consideremos

$$P(E/F) = \frac{P(F/E)P(E)}{P(F/E)P(E) + P(F/E^c)P(E^c)}$$

y $P(F/E^c) = 1$, $P(F/E) \geq \xi^k$ por independencia de las k ejecuciones. Por otro lado, si definimos $\pi(x) = |\{p/1 < p \leq x, \quad \text{primo}\}|$, tendremos

$$P(E^c) = \frac{\pi(2N) - \pi(N)}{N/2},$$

por la elecci'on con probabilidad uniforme sobre los impares del conjunto $\{N+1, ..., 2N\}$.

Para continuar con las estimaciones aprovechemos el siguiente teorema, que es uno de los grandes logros de la maten'atica del siglo pasado.

Teorema (de los n'umeros primos):

$$\lim_{n \to \infty} \frac{\pi(n)}{n/\ln(n)} = 1$$

Observaci'on: La aproximaci'on $n/\ln n$ de $\pi(n)$ es buena para valores de n bastante razonables. Por ejemplo $\pi(n) > n/\ln(n)$ para todo $n \geq 17$ y $\pi(n) < n/(\ln(n) - 4)$ para todo $n \geq 55$.

Volvamos a nuestras estimaciones. Tenemos

$$\begin{aligned} P(E^c) &\approx 2(\frac{2}{\ln N} - \frac{1}{\ln n}) \\ &\approx \frac{2}{\ln N} \end{aligned}$$

de donde $P(E) = 1 - \frac{2}{\ln N}$ y con ello

$$P(E/F) \approx \frac{\epsilon^k (1 - \frac{2}{\ln N})}{\epsilon^k (1 - \frac{2}{\ln N}) + \frac{2}{\ln N}} \leq \epsilon^k \frac{\ln N}{2}$$

Eligiendo

$$k = \frac{\ln(\frac{2}{\ln N}\delta)}{\ln \epsilon}$$

se tiene que $P(E/F) \leq \delta$.

Asi que, si queremos que nuestro candidato a primo lo sea con probabilidad de al menos $1 - \delta$, debemos usar $k = \frac{\ln(\frac{2}{\ln N}\delta)}{\ln \epsilon}$ veces nuestro algoritmo.