RSA
Clase 1

Criptografía
2019

Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Contenido

RSA

- RSA: Rivest, Shamir y Adleman (1977)
- CriptoSistema de Clave Pública - Privada.
- Alice quiere enviarle un mensaje a Bob.
  - Bob genera un par de claves (p_k, s_k). La clave p_k es pública (publicada en algún diccionario) y la clave s_k es el secreto que guarda Bob.
  - Alice utiliza p_k para encriptar su mensaje.
  - Bob recibe el mensaje cifrado y lo desencripta utilizando s_k

RSA

- En un sistema de cifrado simétrico (AES) tenemos que las claves de encriptado y desencriptado son las mismas.
- En este sistema de clave pública, p_k y s_k son distintas y lo que es importante, la clave s_k no puede ser derivada facilmente a partir de la clave p_k
- Para poder enviar los mensajes, estos deben ser convertidos en string de bits 0 y 1. Por ejemplo para textos es usual utilizar la tabla ASCII (extendida) para convertir las letras en strings de 7-bits (8-bits).

RSA : Idea

Supongamos que Alice le quiere enviar un mensaje (cadena de bits) a Bob. Sea n el parámetro de seguridad de RSA.

- Alice particiona su mensaje en bloques de $n-1$ bits y transmite los bloques de forma separada. Sea x un bloque: $x_0, ..., x_{n-2}$
- Interpretamos el bloque como la representación binaria del número natural $x = \sum_{0 \leq i \leq n-1} x_i * 2^i$.

RSA: Idea

- Bob selecciona de forma aleatoria y uniforme dos números primos p y q con $n/2$ bits cada uno. De forma que producto $N = pq$ tiene n bits.
- También selecciona de forma aleatoria y uniforme un entero e talque $1 \leq e \leq N$ y $mcd(e, (p-1), (q-1)) = 1$
- La clave pública de Bos es $p_k = (N, e)$
- Alice recupera esta clave pública de un diccionario y envía su mensaje x encriptado $y = x^e$ en Z_N a Bob.
  En otras palabras, el resto de la división entera de x^e por N.

RSA: Idea

- La magia ahora es que Bob puede recuperar x a partir de y con la información secreta p y q.
- En otra palabras, Bob posee la clave secreta d para desencriptar. Más adelante veremos como calcular d.

RSA: Idea

El parámetro de seguridad n:

security parameter n,
distinct random primes p and q of at least $n/2$ bits,
and so that $N = pq$ has n bits,
$L = \varphi(N) = (p-1)(q-1)$,
$e, d \in \mathbb{Z}_L$ with $ed = 1$ in $\mathbb{Z}_L$,
plaintext x, ciphertext y, decryption z, all in $\mathbb{Z}_N$,
$y = x^e$, $z = y^d$.

RSA: Idea

Generación de clave:

Input: Security parameter n.
Output: secret key sk and public key pk.

1. Choose two distinct primes p and q at random with $2^{(n-1)/2} < p$, $q < 2^{n/2}$.
2. $N \leftarrow p \cdot q$, $L \leftarrow (p-1)(q-1)$. [N is an n-bit number, and $L = \varphi(N)$ is the value of Euler's φ function.]
3. Choose $e \xleftarrow{\boxdot} \{2, \ldots, L-2\}$ at random, coprime to L.
4. Calculate the inverse d of e in $\mathbb{Z}_L$.
5. Publish the public key $\mathsf{pk} = (N, e)$ and keep $\mathsf{sk} = (N, d)$ as the secret key.

RSA: Idea

Función de encriptado y desencriptado:

Encryption enc.

Input: $x \in \mathbb{Z}_N$, $\mathsf{pk} = (N, e)$.
Output: $\mathrm{enc}_{\mathsf{pk}}(x) \in \mathbb{Z}_N$.

6. $y \leftarrow x^e$ in $\mathbb{Z}_N$.
7. Return $\mathrm{enc}_{\mathsf{pk}}(x) = y$.

Decryption dec.

Input: $y \in \mathbb{Z}_N$, $\mathsf{sk} = (N, d)$.
Output: $\mathrm{dec}_{\mathsf{sk}}(y) \in \mathbb{Z}_N$.

8. $z \leftarrow y^d$ in $\mathbb{Z}_N$.
9. Return $\mathrm{dec}_{\mathsf{sk}}(y) = z$.

RSA: Ejemplo Encriptado

Veamos un ejemplo. Tomemos $n = 6$.

- Elegimos p y q tal que: $p = 5$ y $q = 11$. Entonces, $N = 55$ un número de 6 bits.
- Entonces, $L = (5 - 1)(11 - 1) = 40$
- Elegimos de forma aleatoria y uniforme, $e = 13$ coprimo con L.
- Utilizando el Algoritmo Extendido de Euclides calculamos $e^{-}1$: $d = e^{-}1 = 37$ en Z_L
- Entonces Bob publica su clave pública $d = (N, e) = (55, 13)$

RSA: Ejemplo Encriptado

Ahora, Alice quiere enviarle un mensaje $x = 7$ a Bob.

- Alice calcula $y = x^e = 7^{13}$ en Z_{55}
- El método obvio de realizar esto es: computar el entero 7^{13} y tomar el resto de la división por 55.
- !Pero este método puede ser impracticable para valores de n muy grandes!
- Lo que hacemos es utilizar el algoritmo "repeated squaring" que utiliza menos de $2m$ operaciones para una exponenciación de m-bits en Z_N

Repeated Squaring Algorithm

CryptoSchool 15.48.

ALGORITHM 15.48. Repeated squaring.

Input: A group G, a base $x \in G$, and an exponent $e \in \mathbb{Z}$ with $1 \leq e < \#G$.
Output: $x^e \in G$.

1. Let $\sum_{0 \leq i < n} e_i 2^i$ be the binary representation of e with $e_0, \ldots, e_{n-1} \in \{0, 1\}$, $e_{n-1} = 1$, and n its bit length.
2. $y \leftarrow x$.
3. For i from $n - 2$ downto 0 do steps 4-5
4. $\quad y \leftarrow y^2$.
5. $\quad$ If $e_i = 1$, then $y \leftarrow y \cdot x$.
6. Return y.

Repeated Squaring Algorithm

CryptoSchool 15.50.

COROLLARY 15.50. *Let N and e have bit length at most n, and $x \in \mathbb{Z}_N$. Then x^e in $\mathbb{Z}_N$ can be computed with $O(n^3)$ bit operations.* □

Repeated Squaring Algorithm

Calculemos 7^{13} utilizando el algoritmo:

instruction	value	exp	bit	in $\mathbb{Z}_{55}$
$y_0 \longleftarrow x$	x	1	1	7
$y_1 \longleftarrow y_0^2$	x^2	10		49
$y_2 \longleftarrow y_1 \cdot y_0$	x^3	11	1	13
$y_3 \longleftarrow y_2^2$	x^6	110	0	4
$y_4 \longleftarrow y_3^2$	x^{12}	1100		16
$y_5 \longleftarrow y_4 \cdot y_0$	x^{13}	1101	1	2

RSA: Ejemplo Encriptado

Finalmente, Alice termina enviando a Bob:

$$y = 2 = enc_{p_k}(7) = 7^{13} \text{ en } Z_{55}$$

- Bob recibe $y = 2$ y de forma análoga computa utilizando su secreto $d = 37$.
- Es decir, 2^{37} en Z_{55} lo cual le da $z = 7$.
- Observamos que $z = x$, en otras palabras, recupera el mensaje x que Alice le envío a partir de y utilizando su secreto d.

RSA: Cuestiones de Correctitud, Seguridad y Eficiencia

Teniendo una idea general del funcionamiento de RSA, debemos responder ciertas cuestiones que hacen a la correctitud, seguridad y eficiencia del algoritmo:

- Correctitud: ¿es correcto el desencriptado de un mensaje cifrado?
- Eficiencia: ¿se puede calcular de forma eficiente ...
  - seleccionar de forma aleatoria y uniforme números primos grandes?
  - el secreto d a partir de la clave pública e?
  - potencias módulo N? Esto tiene que ser calculado para cada mensaje.
- Seguridad:
  - Supongamos que Eve escucha la comunicación entre Alice y Bob. Es decir, Eve escucha y y por supuesto la información públic (N, e).
  - Eve quiere computar x. Pero,
  - ¿cuánto tiempo le lleva calcular este valor?
  - ¿esto es suficientemente difícil como para proveer seguridad?

Eficiencia y Problema Difícil

Recordemos las nociones de eficiencia y problema difícil (CS 15.29):

DEFINITION 15.29. *(i) For two functions* $f, g\colon \mathbb{N} \longrightarrow \mathbb{R}$, *we say that* $f \in \mathrm{O}(g)$ *if there exist* c *and* N *so that*

$$|f(n)| \leq c \cdot |g(n)| \text{ for all } n \geq N.$$

Thus $\mathrm{O}(g)$ *consists of those functions which grow in absolute value at most like a constant multiple of* f.

(ii) A function $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ *is* polynomially bounded *if there exists some* k *so that* $f \in \mathrm{O}(n^k)$, *where* n *is the input of* f. *We then write* $f \in \mathrm{poly}(n)$.

Eficiencia y Problema Difícil

(iii) *A probabilistic algorithm is* efficient *if its expected runtime, depending on the input length n and measured in basic operations (say, bit operations or, for some algebraic problems, operations in the ground domain), is polynomially bounded in n. We also call it* a polynomial-time algorithm.

(iv) *A computational problem X is* easy *if there exists a probabilistic polynomial-time algorithm that solves it; otherwise X is* hard.

Eficiencia y Problema Difícil

- "Tiempo Polinomial" es un concepto central en el análisis de algoritmos.
- Si se dobla la capacidad computacional, el tamaño de los problemas resolubles en tiempo $O(n^k)$ se incrementa notablemente, desde n a $2^{1/k}n$, mientrás que para el tiempo exponencial solamente se incrementa desde n a $n+1$.

Recomendaciones Prácticas

NIST (2012b) y BSI (2014) publican recomendaciones de seguridad teniendo en cuenta el largo de claves. Comparan el costo actual de ataques conocidos con requerimientos de seguridad semejantes entre los distintos criptosistemas.

- Para RSA la recomendación es utilizar tamaño de clave mayor o igual a 3000.
- Nótese que a comparable nivel de seguridad, curvas elípticas utiliza longitudes de claves mucho menores que RSA.

	s	AES	RSA	ECC	hash
legacy	80		1024	160	160
near-term	128	128	3000	256	256
long-term	256	256	15360	512	512

Recomendaciones Prácticas

- Es importante tener en cuenta que debemos estar atentos a los avances matemáticos o computacionales que se logren en cuanto a la resolución del problema difícil subyacente en el que radica la seguridad del criptosistema.
- Ejemplos: computación cuántica, logaritmo discreto, factorización de números grandes, etc.
- Todas estas recomendaciones se basan en la suposición que no se esperan grandes avances en este tipo de criptoanálisis.
- Por otro lado sería ingenúo pensar que los mejores algoritmos son siempre conocidos publicamente.

Análisis de RSA: Correctitud

- Correctitud: Para cada mensaje x, el desencriptado del encriptado de x, es igual a x.
- Por definición, $z = x^{ed}$ en Z_N y sabemos que

$$ed = 1 + k\phi(N) \text{ para algún } k \text{ en } Z$$

Se debe demostrar que para cualquier x en Z_N la igualdad $z = x$ es cierta.

Análisis de RSA: Correctitud

- La prueba se puede ver en CS 3.1 donde se hace una distinción de casos según $mcd(x, N)$ sea igual a 1 o no.
- Lo interesante es ver cuando $mcd(x, N) \neq 1$. En otras palabras, el mcd es p ó q (ó N). Esto debe pasar con probabilidad despreciable, sino de otra forma Alice o Eve pueden recuperar el secreto de Bob facilmente.
- Afortunadamente esto efectivamente ocurre con probabilidad despreciable para un elemento aleatorio x:

$$1 - \frac{\varphi(N)}{N} = 1 - \frac{(p-1)(q-1)}{N} = \frac{p+q-1}{N} \approx \frac{2 \cdot 2^{n/2}}{2^n} = 2^{-n/2+1}.$$

Análisis de RSA: Correctitud

- Notemos la propiedad de "multiplicatividad" de RSA:

$$(x * w)^d = x^d * w^d$$

es decir,

$$e(m_1) * e(m_2) = e(m_1 * m_2)$$

Infinidad de números primos

- Uno de las cuestiones de eficiencia que planteamos fue cómo seleccionar de forma aleatoria y uniforme un número primo p muy grande.
- Comenzaremos con el teorema de los números primos:

Hay infinitos números primos

PROOF. We assume that the set of primes is finite and denote it as $P = \{p_1, ..., p_k\}$. The number $p = p_1 \cdot p_2 \cdots p_k + 1$ is coprime to each of the primes in P. On the other hand, according to the fundamental theorem of arithmetic on unique prime factorization, p has some prime factors, which must be in P. This contradiction proves the theorem. □

Infinidad de números primos

Uno de los tipos famosos de números primos es cuando el número de *Marsenne* es primo.

- son números de la forma $M_n = 2^n - 1$ con n en N.
- Se puede ver que con esta forma no necesariamente se es primo, ejemplo, $M_4 = 15$. En realidad si n es compuesto también lo es M_n.
- Entonces cuando se buscan primos de esta forma, n debe ser primo.
- A la fecha de diciembre 2018, solamente 51 números primos de esta forma son conocidos.

Infinidad de números primos

- Vamos a ver que hay números primos que se consideran "safe" y "strong" en cuanto cumplen con ciertas propiedades.
- Un número primo se dice seguro si es de la forma $2p + 1$ con p primo.
- Por ejemplo, los números asociados a los números primos de Sophie Germain se dicen seguros.
- Este tipo de números son los que se usan en la Criptografía. Por ejemplo a la hora de elegir, p y q para RSA.
  Vamos a ver más adelante que si elegimos estos factores de forma segura, factorizar utilizando el algoritmo "Pholard's p' es computacionalmente inviable.
- Pero tener en cuenta que es un problema abierto el que varios de estos tipos de números sean infinitos o no.

Test de primalidad

- Para encontrar números primos en un determinado rango, seleccionamos de forma aleatoria y uniforme un número N y le aplicamos tests de primalidad.
- Por definición un número N es primo si no tiene divisores entre 2 y $N-1$. Si tiene un divisor, este divisor es menor que $\sqrt{N}$.
- Entonces para deteriminar si un número es primo, el algoritmo trivial es probar con todos los factores primos entre 2 y $\sqrt{N}$.
  Esto es computacionalmente sumamente costoso.
- Entonces, una idea es reemplazar la propiedad de la definición por una propiedad que la cumplan los números primos pero que no la cumplan (con suficiente probabilidad) los números compuestos.

Test de Fermat

- Pequeño Teorema de Fermat (CS 15.56): Si N es un número primo, luego para cualquier $x \neq 0$ en Z_N tenemos que $x^{N-1} = 1$ en Z_N
- Basandonos en esta propiedad se puede escribir el siguiente test probabilístico:

ALGORITHM 3.6. Fermat test.

Input: A number $N \in \mathbb{Z}$ with $N \geq 2$.
Output: Either "N is composite", or "N is possibly prime".

1. $x \xleftarrow{\boxdot} \{1, \ldots, N-1\}$.
2. $g \leftarrow \gcd(x, N)$. If $g \neq 1$, then Return "N is composite".
3. $y \leftarrow x^{N-1}$ in $\mathbb{Z}_N$.
4. If $y \neq 1$, then Return "N is composite".
5. Return "N is possibly prime".

Test de Fermat

- Si N es primo, el test correctamente responderá que N es posiblemente primo.
- Estamos buscando por primos, entonces podemos decir que en este test no hay falsos negativos.
  En otras palabras, no va a suceder que para un número me diga que es NO es primo cuando lo es.
- Por otro lado puede suceder que el test me diga que un número es posiblemente primo cuando no lo es.
  Esto quiere decir un falso positivo.

Test de Fermat

Ahora supongamos que N es compuesto. Un elemento x de Z_N es llamado:

- **testigo (Fermat witness)** para el número compuesto N si $x^{N-1} \neq 1$ en Z_N
- ó **mentiroso (Fermat liar)** en el otro caso.

Test de Fermat

- El grupo L_N compuesto por **(Fermat liars)** es un subgrupo de Z_N.
- Por el teorema de Lagrange:
  - ó L_N es igual a Z_N (no hay ningún testigo de Fermat, lo cual no quiere decir que sea primo)
  - ó como mucho tiene la mitad de elementos.
- Entonces si hay al menos un testigo de Fermat, los hay muchos y como mucho la mitad de elementos de Z_N.

Test de Fermat

- **Carmichael numbers** son los números N compuestos para los cuales no hay ningún testigo de Fermat.
- El test de Fermat falla para estos números N, retornando incorrectamente que son posiblemente primos excepto para los cuales se cumpla que sean coprimos con N (que son despreciablemente pocos) en el paso 2.
- Los primeros números de Carmichael son:
  $561 = 3 * 11 * 17$, $1105 = 5 * 3 * 17$, $1729 = 7 * 13 * 19$.

The end.