RSA
Clase 3

Criptografía
2019

Instituto de Computación
Facultad de Ingeniería
Universidad de la República

Contenido

RSA: Hard Core Bit

- Hasta ahora asumimos que el problema B_1 de recuperar el texto plano x dada la clave pública $p_k = (N, e)$ y el texto cifrado $y = enc_{pk}(x)$ es difícil.
- Pero podría ser **fácil averiguar información parcial!**

RSA: Hard Core Bit

¿Qué pasa si tenemos una forma fácil de conocer el bit menos significativo (bit de paridad) de x?

llamado "hard core bit de RSA"

- Llamemos, B_0 al problema de computar el bit x_0 menos significativo de x.
- Claramente se cumple que:

$$B_0 \leq_p B_1$$

RSA: Hard Core Bit

Teorema 3.39: $B_0 \equiv_p B_1$

- Nos falta probar que, $B_1 \leq_p B_0$
- es decir, puedo construir un algoritmo probabilístico en tiempo polinomial que en sucesivas llamadas a B_0 (computa el bit menos significativo), resuelve B_1

RSA: Hard Core Bit

Corolario 3.40

- Si RSA es seguro, entonces el bit menos significativo del texto plano también es seguro.
- Entonces,

$$prob\{x_0 = A(N, e, enc_{pk}(x))\} - 1/2$$

  es despreciable para todo atacante A en tiempo polinomial.
- En otras palabras, lo mejor que puede hacer el atacante es tirar una moneda.

RSA: Teorema Chino del Resto

- Según la definición de RSA, Bob puede borrar p y q una vez que determinó $N = pq$ y los exponentes e y d.
- Pero si se almacenan estos números, se puede simplificar el cómputo de desencriptar $x = y^d$ en Z_N.

RSA: Teorema Chino del Resto

Recordemos el Teorema Chino del Resto:
Supongamos que $n_1, n_2, ..., n_k$ son enteros positivos **coprimos dos a dos**.
Entonces, dado los enteros $a_1, a_2, ..., a_k$ existe un único entero x módulo N tal que satisface:

- $x \equiv a_1 \mod n_1$, $x \equiv a_2 \mod n_2$, ..., $x \equiv a_k \mod n_k$
- y N es el producto de $N = n_1 * n_2 * ... * n_k$

RSA: Teorema Chino del Resto

1. Computamos:
   - $d_p = d \bmod (p-1)$
   - $d_q = d \bmod (q-1)$

   Notemos que d puede ser grande y d_p, d_q pequeños.
2. Computamos $x_p = y^{d_p} \bmod (p)$ y $x_q = y^{d_q} \bmod (q)$
3. Utilizamos el teorema Chino del Resto para computar el único valor $x \in Z_N$ que satisface:
   - $x \equiv x_p \bmod (p)$
   - $x \equiv x_q \bmod (q)$

   El resultado satisface $x \equiv y^d \bmod (N)$
   (ver fórmula de Garner)

RSA: Teorema Chino del Resto

Consideraciones:

- En vez de trabajar en el cuerpo Z_N trabajamos en Z_p y Z_q que tienen la mitad de tamaño en bits que N.
- Para desencriptar utilizando el TCR se requieren $3n^3/8$ operaciones. En contraste con Z_N que requiere $3n^3/2$ operaciones.
- Aproximadamente se puede ver que se ahorra un 75 % del trabajo.
- Entonces este método se utiliza en la práctica. En especial, por ejemplo, en las tarjetas inteligentes generando una firma digital basada en RSA.

El ataque de Wiener

- Bob puede estar tentado a utilizar un exponente d chico para mejorar la eficiencia de computo al desencriptar.
- El ataque de Wiener nos muestra que si se utiliza un d chico, un atacante puede quebrar el criptosistema computando el exponente d.

El ataque de Wiener

Teorema 3.37:

- Supongamos que las siguientes hipotesis se satisfacen:
  - $1 \leq d \leq N^{1/4}\sqrt{12}$
  - $p < q < 2p$
  - $1 \leq e < \phi(N)$
- entonces el exponente d puede ser computado en tiempo $O(n^2)$

En otras palabras podemos decir que si N tiene l bits en su representación binaria, el ataque tendrá éxito cuando d tiene menos que $l/4 - 1$ bits y p y q están cerca.

Side Channel Attacks

En este curso no vamos a profundizar en estos ataques, pero debemos conocer su existencia.

- Una primitiva criptográfica (algoritmo) debe ser implementada en un programa que ejecutará en un procesador dado, o de forma más general en un ambiente dado. En consecuencia presentará ciertas caracteristicas de implementación (hardware).
- Los ataques físicos toman ventaja de las carcterísticas de implementación para quebrar el criptosistema.
- Es por esto que estos ataques son menos generales, dado que son específicos a cierta implementación, pero pueden ser mucho más poderosos que el criptoanálisis clásico.
- Es por esto que deben ser tenidos en cuenta a la hora de fabricar los dispositivos que ejecutarán (o almacenaran información de) los algoritmos.

Side Channel Attacks

Los ataques fisicos usualmente se clasifican de la siguiente manera:

- Invasivos o No-invasivos
  - Los ataques invasivos desarman el dispositivo para acceder al núcleo (chip) y de esta forma tener acceso directo a sus componentes. Por ejemplo, puede ser un cable conectado a un puerto para obtener la transferencia de datos.
  - Los ataques no-invasivos explotan solamente la información disponible externa la cuál usualmente se emite de forma involuntaria por el dispositivo. Por ejemplo, tiempo de cómputo, consumo eléctrico, emisión de sonidos, temperatura de la CPU, etc.
- Activo o Pasivo
  - Los ataques activos tratan de manipular el dispositivo para lograr un mal funcionamiento deseado y así explotarlo. Por ejemplo, inducir errores en los cálculos.
  - Los ataques pasivos simplemente observan el comportamiento de los dispositivos durante el procesamiento sin alterarlo.

La paradoja del cumpleaños

- La paradoja del cumpleaños se utiliza para analizar las colisiones de cierta función (por ejemplo, hash).
- En el sentido estricto no es una paradoja en tanto no es una contrdicción lógica sino que contradice el sentido común.
- Supongamos que si consideramos un año calendario, la gente nace de forma aleatoria y uniforme:

  ¿cuántas personas seleccionadas de forma aleatoria y uniforme se necesitan para que al menos dos personas cumplan el mismo día con probabilidad $1/2$**?**

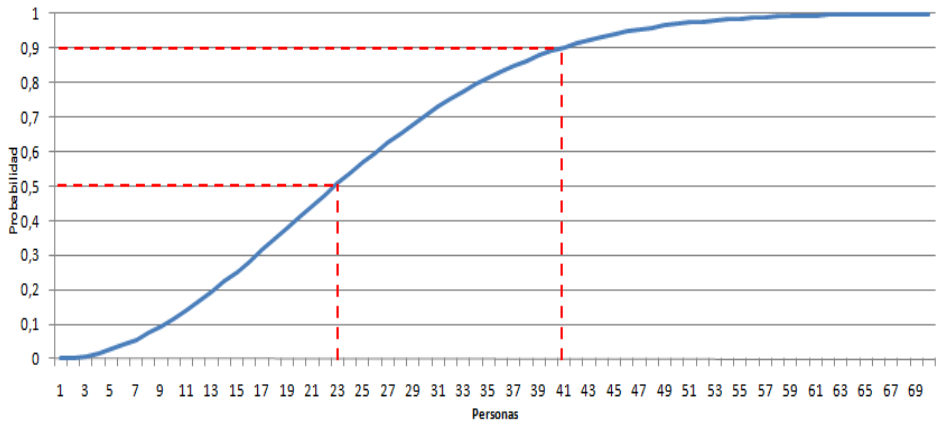
La paradoja del cumpleaños

- Calculemos la probabilidad de que en una habitación con n personas, al menos dos cumplan el mismo día. Asumimos que los 365 días tienen la misma probabilidad.
- Calculemos considerando el complemento es decir que no coincidan los cumpleaños:

$$p = 1 - \frac{365}{365} \cdot \frac{364}{365} \cdots \frac{365-n+1}{365} = 1 - \frac{365!}{365^n (365-n)!}$$

La paradoja del cumpleaños

- La paradoja dice que en un grupo de 23 personas elegidas al azar, al menos dos personas cumplirán el mismo día con probabilidad de 1/2.



¿cuál es la probabilidad en la clase?, ¿tenemos una colisión?

La paradoja del cumpleaños

- Considerando el siguiente algoritmo, si modelamos la función de hash h como la función de cumplir años. Su dominio serán las personas y el codominio los 365 días del año.
- La paradoja del cumpleaños nos está diciendo que encontraremos una colisión con probabilidad de al menos $1/2$ cuando seleccionemos $Q = 23$ personas.

Algorithm 4.3: FIND-COLLISION(h, Q)

choose $\mathcal{X}_0 \subseteq \mathcal{X}, |\mathcal{X}_0| = Q$
for each $x \in \mathcal{X}_0$
  do $y_x \leftarrow h(x)$
if $y_x = y_{x'}$ for some $x' \neq x$
  then return (x, x')
  else return (failure)

La paradoja del cumpleaños

Teorema 3.41

- Si consideramos selecciones aleatorias de m elementos. El esperado número de elecciones hasta que una colisión ocurra es del orden de $O(\sqrt{m})$
- En otras palabras, *hashing* sobre $\sqrt{m}$ elementos aleatorios de cierto dominio X tendrá una colisión con probabilidad aproximada de 0,5.

Ataque de cumpleaños

- Un ataque de cumpleaños impone una cota inferior en el tamaño del resumen (digest) de cierto mensaje (función de Hash).
- Una función de hash de tamaño 40-bit será insegura, dado que una colisión podría encontrarse con probabilidad de $1/2$ con solamente 2^{20} (un millón) de hashes aleatorios.
- Se puede recomendar utilizar tamaños de hash de 128-bit donde un ataque de cumpleaños requerirá aproximadamente 2^{64} hashes.
- Por lo general se utilizan hashes de 160-bit o más grandes.

Algoritmo de Pollard rho

Idea general:

- Sea p el divisor primo más chico de n. Supongamos que existen dos enteros distintos $x, x' \in Z_n$ tal que $x \equiv x' \mod p$.
  Luego se cumple que $p = mcd(x - x', n)$ es un divisor no trivial de n.
- Entonces obtenemos un factor de n computando el mcd de la diferencia de una colisión.

Algoritmo de Pollard rho

Idea general:

- Supongamos que tratamos de factorizar n seleccionando aleatoriamente un subconjunto de $X \subset Z_n$ y luego computando $mcd(x - x', n)$ para todos los valores distintos de $x, x' \in X$.
- Este método será exitoso si y solo si el mapeo $x \longmapsto x \bmod p$ tiene al menos una colisión para $x \in X$.
- Si analizamos la colisión utilizando la paradoja de cumpleaños, podemos decir que si $|X| \approx 1{,}17\sqrt{(p)}$ hay una probabilidad de 0,5 en tener al menos una colisión y en consecuencia en tener un factor no trivial de n.

Algoritmo de Pollard rho

- Sin embargo, para encontrar una colisión de $x \bmod (p) = x' \bmod (p)$ necesitamos computar los $mcd(x - x', n)$.
- No podemos explícitamente computar los valores de $x \bmod (p)$ para $x \in X$ pues el valor de p es desconocido.
- Esto significa que podríamos esperar computar más de $\binom{|X|}{2} > p/2$ $mcd's$ antes de encontrar un factor de n.

Algoritmo de Pollard rho

- El algoritmo de Pollard rho incorpora una variante de la técnica general que vimos que requiere menos computaciones de *mcd* y memoria.
- El algoritmo define la función polinomial con coeficientes enteros $f : Z_N \longrightarrow Z_N$ talque $f(x) = x^2 + 1$.
- Se selecciona aleatoriamente un elemento inicial x_0 y se define $x_i \in Z_N$ como $x_i = f(x_{i-1})$ en Z_N.
- Se asume que el mapeo $x \longmapsto f(x) \bmod p$ se comporta como un mapeo aleatorio. Claramente no es aleatorio lo que significa que el método se basa en un análisis heurístico.
- Entonces, cómo ya dijimos anteriormente, podemos esperar una colisión en Z_P luego de $O(\sqrt{(}p))$ pasos.

Algoritmo de Pollard rho

Veamos un ejemplo.

EXAMPLE 3.43. We want to factor $N = 82\,123$. Starting with $x_0 = 631$, we find the following sequence:

i	$x_i \bmod N$	$x_i \bmod 41$	i	$x_i \bmod N$	$x_i \bmod 41$
0	631	16	6	40 816	21
1	69 670	11	7	80 802	32
2	28 986	40	8	20 459	0
3	69 907	2	9	71 874	1
4	13 166	5	10	6 685	2
5	64 027	26	11	14 313	5

Algoritmo de Pollard rho

- Notemos que la tercer columna no la conocemos cuando ejecutamos el algoritmo, solamente se muestra a efectos de identificar la colisión.
- Se puede observar que la colisión se da entre x_3 y x_{10} donde el $mcd(x_3 - x_{10}) = 41$ divide a N.
- Se puede observar también que luego de darse la colisión se comienzan a repetir los valores, ie, $x_4 = x_{11}$

Algoritmo de Pollard rho

- Esta última observación proviene de un resultado importante:

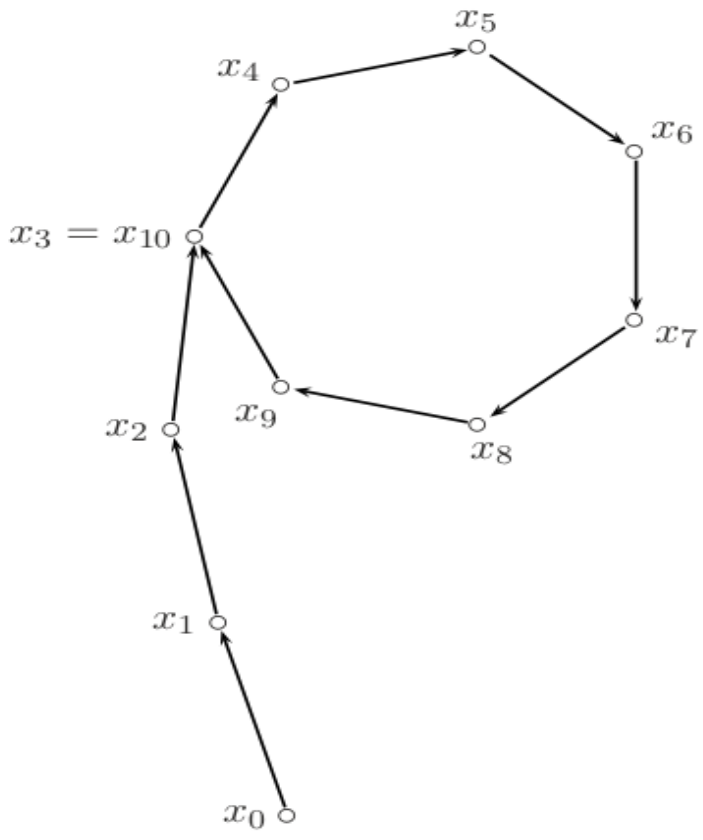
If $x_i \equiv x_j \pmod{p}$, then $x_{i+\delta} \equiv x_{j+\delta} \pmod{p}$ for all integers $\delta \geq 0$.

Denoting $\ell = j - i$, it follows that $x_{i'} \equiv x_{j'} \pmod{p}$ if $j' > i' \geq i$ and $j' - i' \equiv 0 \pmod{\ell}$.

- Esto quiere decir que en cuanto ocurra una colisión, se comenzará a **repetir un ciclo infinito de longitud l de colisiones**.
- Si construimos un grafo G con vertices en Z_p, podremos observar una "cola" y el ciclo infinito de longitud l.
  Este grafo G se parece a la forma de la **letra griega** ρ a lo que debe el nombre del algoritmo.

Algoritmo de Pollard rho

Este grafo G para el ejemplo es:



Pollard rho con el truco de Floyd

- Hemos dicho que nuestro objetivo es descubrir una colisión. No es necesario que descubramos la primer colisión que ocurra.
- Entonces con el objetivo de acelerar el algoritmo, se puede restringir la búsqueda a colisiones de la forma $x_i = x_{2i}$
- En esto se basa la idea de Floyd, utiliza otra secuenca y_i que itera en f el doble de rápido $(2i)$ y solamente almacena los valores actuales de x_i y y_i.

Pollard rho con el truco Floyd

El algoritmo con el truco de Floyd es:

ALGORITHM 3.45. Pollard's ρ algorithm.

Input: An odd composite integer N.
Output: A proper divisor of N, or "failure".

1. $i \leftarrow 0$.
2. $x_0 \xleftarrow{\boxdot} \mathbb{Z}_N$.
3. $y_0 \leftarrow x_0$.
4. Repeat steps 5–8
5. $\quad i \leftarrow i + 1$,
6. $\quad x_i \leftarrow x_{i-1}^2 + 1$ in $\mathbb{Z}_N$,
7. $\quad y_i \leftarrow (y_{i-1}^2 + 1)^2 + 1$ in $\mathbb{Z}_N$,
8. $\quad g \leftarrow \gcd(x_i - y_i, N)$.
9. Until $g \neq 1$.
10. If $g < N$ then return g else return "failure".

Pollard rho con el truco de Floyd

Complejidad:

THEOREM 3.46. *Let $N \in \mathbb{N}$ be a composite n-bit integer, p its smallest prime factor, and $f(x) = x^2 + 1$. Under the assumption that the sequence $(f^{(i)}(x_0))_{i \in \mathbb{N}}$ of iterates of x_0 behaves modulo p like a random sequence, the expected number of iterations in Pollard's algorithm for returning a proper factor of N is $O(\sqrt{p}) \subseteq O(N^{1/4})$, using an expected number of $O(N^{1/4} n^2)$ bit operations.*

Algoritmo de Pollard rho

Analizamos la falla del algoritmo:

- Es posible que el algoritmo pueda fallar en encontrar un factor no trivial de n.
- Esto sucede si y solo si los primeros valores x y x' que satisfacen $x \equiv x' \bmod (p)$ también satisfacen $x \equiv x' \bmod (n)$.
  Esto es equivalente a decir que $x = x'$ porque x y x' están reducidos módulo n.
- Se puede estimar heurísticamente que la probabilidad de que ocurra esto es aproximadamente p/n, lo cuál es chico cuando n es grande, porque $p < \sqrt{n}$
- Si el algoritmo falla de esta manera, es cuestión de seleccionar otro x_0 y comenzar de nuevo o con otra elección de f.

Algoritmo de Pollard rho

Para terminar:

- Recursivamente se puede aplicar el algoritmo para factorizar n completamente.
- Cómo ya mencionamos antes, el algoritmo se prueba heurísticamente dado que claramente f no es aleatoria.
  Es un problema abierto determinar cuando la paradoja de cumpleaños (la cual asume verdadera aleatoriedad) aplica realmente al algoritmo.
- Como se verá más adelante, el mismo enfoque se aplicará en el logaritmo discreto, pero en este contexto la propiedad que se requiere para el ataque de cumpleaños está probada.

The end.