

El criptosistema RSA

Ronald Rivest

Adi Shamir

Leonard Adleman



1976: Whitfield Diffie y Martin Hellman introducen criptografía de llave pública.



Se comienzan a buscar métodos con los cuales se pueda utilizar encriptación de llave pública



1977: Ronald Rivest, Adi Shamir y Leonard Adleman proponen lo que resultó ser el esquema criptográfico asimétrico más utilizado (RSA)

- RSA fue patentado en USA (pero no en el resto del mundo) hasta 2000.
- Es mayormente utilizado para:
 - Encriptación de pedazos pequeños de información (especialmente transporte de llaves).
 - Firmas electrónicas

- Algunas veces es más lento que cifrados como el AES.
- En la práctica RSA se utiliza frecuentemente en conjunto con un cifrado simétrico (por ejemplo el AES).

Cifrado y descifrado

Estos se realizan en el anillo $\mathbb{Z}_n = \{0, 1, \dots, n-1\}$

Cifra
RSA ~~encripta~~ texto plano x considerándolo como un elemento en \mathbb{Z}_n
 \Rightarrow el valor binario de x debe ser menor que n .

Cifrado RSA: Dada la llave pública $(n, e) = K_{\text{pub}}$ y el texto plano x , la función encriptación es

$$y = e_{K_{\text{pub}}}(x) \equiv x^e \pmod{n}$$

donde $x, y \in \mathbb{Z}_n$

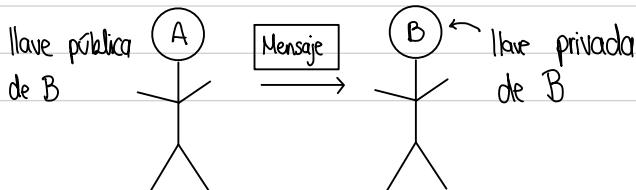
Descifrado RSA: Dada la llave privada $d = K_{\text{pr}}$ y el texto cifrado y , la función de descifrado es

$$x = d_{K_{\text{pr}}}(y) \equiv y^d \pmod{n}$$

donde $x, y \in \mathbb{Z}_n$

x, y, n, d suelen ser números muy grandes (~ 1024 bits o más)

A valor e se le llama exponente de encriptación o exponente público, y al valor d exponente de descifrado o exponente privado.



Requisitos para un criptosistema RSA.

1. Debe ser computacionalmente no viable determinar la llave privada d dados los valores n y e de la llave pública. (De lo contrario, conoceríamos la llave pública y la llave privada \Rightarrow somamente inseguro)
2. No encriptar más de l bits con una encriptación RSA, donde l es la longitud en bits de n . (Esto dado que x sólo es único hasta el tamaño del módulo n)
3. Deber relativamente fácil calcular $x^e \bmod n$ y $y^d \bmod n$ (para cifrar y descifrar). (Necesitamos un método para calcular exponentiación de números grandes de forma rápida).
4. Para n dado deben existir varios pares de llave privada y llave pública. (De lo contrario es susceptible a un ataque mediante fuerza bruta)

7.3 Generación de llaves y prueba de correctitud

- La generación de llaves puede ser compleja, esto depende del esquema de llave pública.

Algunos pasos involucrados en el cálculo de llaves privadas y públicas para un Criptosistema RSA:

1. Elegir dos números primos p, q grandes
(Ejemplo: $p=11, q=13$)
2. Calcular $n=p \cdot q$
(Ejemplo: $n=(11) \cdot (13) = 143$)

3. Calcular $\phi(n) = (p-1)(q-1)$

(Ejemplo: $\phi(143) = (11-1)(13-1) = (10)(12) = 120$)

4. Seleccionar el exponente público $e \in \{1, 2, \dots, \phi(n)-1\}$ tal que
 $\gcd(e, \phi(n)) = 1$

(Ejemplo: $\gcd(e, 120) = 1 \Rightarrow e \in \{1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, 59, 61, 67, 71, 73, 77, 79, 83, 89, 91, 95, 97, 101, 103, 107, 109, 113, 115, 119\}$)
Seleccionamos $e = 71$)

5. Seleccionar la llave privada d tal que

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

(Ejemplo: $d \cdot 71 \equiv 1 \pmod{120} \Rightarrow d \cdot 71 = 120 \cdot x + 1, x \in \mathbb{N}$)
Notemos que si $d = 71 \Rightarrow (71)(71) = 5041 = 5040 + 1 = (120)(42) + 1 \equiv 1 \pmod{120}$
Seleccionamos $d = 71$)

⇒ Obtenemos la llave pública $K_{\text{pub}} = (n, e)$ y la llave privada $K_{\text{pr}} = (d)$.

Ejemplo: $K_{\text{pub}} = (143, 71)$ y $K_{\text{pr}} = 71$

Observación

- La condición $\gcd(e, \phi(n)) = 1$ asegura que el inverso de e existe módulo $\phi(n) \Rightarrow$ siempre existe una llave privada d .
- El cálculo de las llaves d y e puede realizarse al mismo tiempo usando la versión extendida del algoritmo de Euclides

- En la práctica, se realiza el siguiente proceso:

(1) Se selecciona un parámetro público e con $0 < e < \phi(n)$

Nota: Si $e = 1 \Rightarrow y = z^e \pmod{n} = z \pmod{n}$
función de encriptación

$\Rightarrow z = y$. Debemos excluir el caso $e = 1$.

(2) Se aplica el algoritmo de Euclides extendido conociendo n y e para obtener la relación

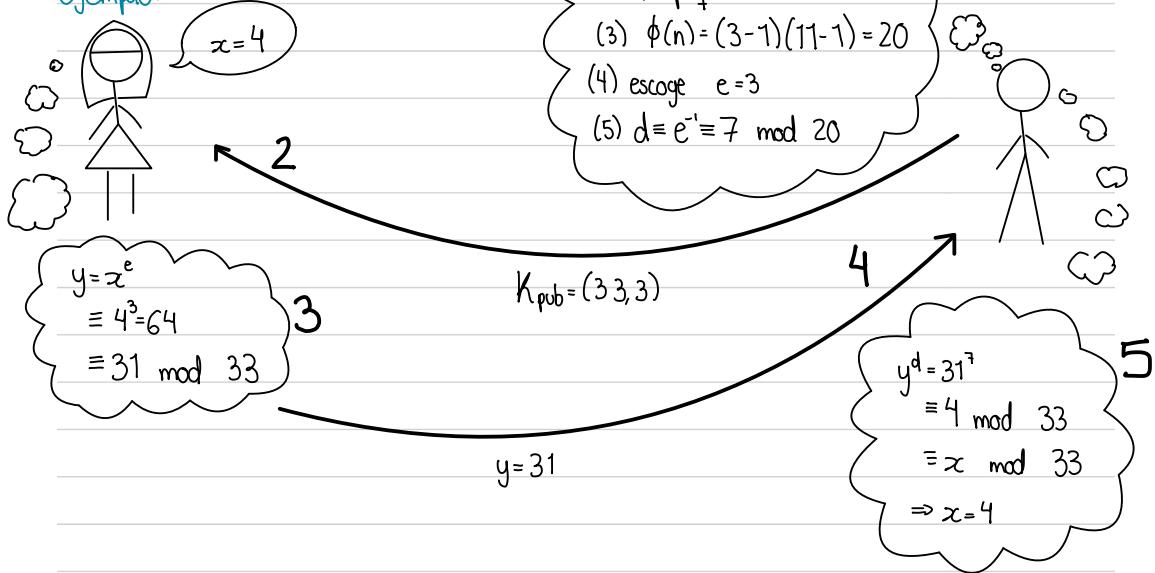
$$\gcd(\phi(n), e) = s \cdot \phi(n) + t \cdot e$$

(3) Evaluamos: si $\gcd(e, \phi(n)) = 1 \Rightarrow e$ es una llave pública válida
De lo contrario, volvemos al paso (1).

(4) Si e es llave pública $\Rightarrow t$ es el inverso de e

Observación: Nunca usamos s .

Ejemplo:



Nota: En la práctica, RSA utiliza parámetros más grandes.

Un RSA módulo n debe ser de por lo menos 2048 bits de largo, lo cual hace que p y q tengan longitud de 1024 bits

Ejemplo: Si n es de 1024 bits se pueden usar, por ejemplo, los siguientes parámetros.

- $p = E0DFD2C2A288ACEBC705EFAB30E4447541A8C5A47A37185C5A9$
- $CB98389CE4DE19199AA3069B404FD98C801568CB9170EB712BF$
- $10B4955CE9C9DC8CE6855C6123_h$
- $q = EBE0FCF21866FD9A9F0D72F7994875A8D92E67AEE4B515136B2$
- $A778A8048B149828AEA30BD0BA34B977982A3D42168F594CA99$
- $F3981DDABFAB2369F229640115_h$
- $n = CF33188211FDF6052DBBB1A37235E0ABB5978A45C71FD381A91$
- $AD12FC76DA0544C47568AC83D855D47CA8D8A779579AB72E635$
- $D0B0AAC22D28341E998E90F82122A2C06090F43A37E0203C2B$
- $72E401FD06890EC8EAD4F07E686E906F01B2468AE7B30CBD670$
- $255C1FEDE1A2762CF4392C0759499CC0ABECFF008728D9A11ADF_h$
- $e = 40B028E1E4CCF07537643101FF72444A0BE1D7682F1EDB553E3$
- $AB4F6DD8293CA1945DB12D796AE9244D60565C2EB692A89B888$
- $1D58D278562ED60066DD8211E67315CF89857167206120405B0$
- $8B54D10D4EC4ED4253C75FA74098FE3F7FB751FF5121353C554$
- $391E114C85B56A9725E9BD5685D6C9C7EED8EE442366353DC39_h$
- $d = C21A93EE751A8D4FBFD77285D79D6768C58EBF283743D2889A3$
- $95F266C78F4A28E86F545960C2CE01EB8AD5246905163B28D0B$
- $8BAABB959CC03F4EC499186168AE9ED6D88058898907E61C7CC$
- $CC584D65D801CFE32DFC983707F87F5AA6AE4B9E77B9CE630E2$
- $C0DF05841B5E4984D059A35D7270D500514891F7B77B804BED81_h$

Observación

Para cifrar y descifrar el mensaje usamos que

$$d_{K_{pr}}(y) = d_{K_{pr}}(e_{K_{pub}}(x)) \equiv (x^e)^d \equiv x^{de} \equiv x \pmod{n} \quad (*)$$

En esto se basa RSA. Hay que verificar que funciona

Prueba de correctitud

Queremos verificar que el descifrado es la inversa de la función de cifrado.
(i.e. $d_{K_{priv}}(e_{K_{pub}}(x)) = x$)

Comenzaremos construyendo la regla entre las llaves privada y pública
 $d \cdot e \equiv 1 \pmod{\phi(n)}$

Notemos que

$$d \cdot e \equiv 1 \pmod{\phi(n)} \iff d \cdot e = 1 + t \cdot \phi(n) \text{ para cierto } t \in \mathbb{Z}.$$

Sustituyendo esto en (*) tenemos que

$$d_{K_{priv}}(y) \equiv x^{de} \equiv x^{1+t\phi(n)} \equiv x^{t\phi(n)} \cdot x = (x^{\phi(n)})^t \cdot x \pmod{n} \quad (*)$$

Entonces debemos probar que $x \equiv (x^{\phi(n)})^t \cdot x \pmod{n}$

Observación

Teorema (Euler)

Si $\gcd(x, n) = 1$ entonces $1 \equiv x^{\phi(n)} \pmod{n}$.

De esto, $1 \equiv 1^t \equiv (x^{\phi(n)})^t \pmod{n}$ para todo $t \in \mathbb{Z}$

Dividiremos la prueba en dos casos

$$\begin{cases} \text{Caso 1: } \gcd(x, n) = 1 \\ \text{Caso 2: } \gcd(x, n) = \gcd(x, p \cdot q) \neq 1 \end{cases}$$

Caso 1: $\gcd(x, n) = 1$

Podemos usar el teorema previo
 $\Rightarrow 1 \equiv (x^{\phi(n)})^t \pmod{n}$

Sustituimos esto en la ecación (*) y obtenemos

$$d_{K_p}(y) \equiv (x^{\phi(n)})^t \cdot x \equiv 1 \cdot x \equiv x \pmod{n} \quad \checkmark$$

Caso 2: $\gcd(x, n) = \gcd(x, p \cdot q) \neq 1$

Como q y p son primos entonces uno de ellos debe de dividir a x , esto es

$$x = r \cdot p \quad \text{ó} \quad x = s \cdot q$$

con $r, s \in \mathbb{Z}$ tales que $r \nmid q$ y $s \nmid p$.

Sin pérdida de generalidad podemos suponer $x = r \cdot p$
 $\Rightarrow \gcd(x, q) = 1$

Usamos ent. el teorema previo obteniendo

$$1 \equiv 1^t \equiv (x^{\phi(q)})^t \pmod{q}$$

donde t es cualquier entero.

Luego,

$$(x^{\phi(n)})^t \equiv (x^{(q-1)(p-1)})^t \equiv ((x^{\phi(q)})^t)^{p-1} \equiv 1^{(p-1)} = 1 \pmod{q}$$

$$\Rightarrow (x^{\phi(n)})^t = 1 + u \cdot q, \text{ para cierto } u \in \mathbb{Z}$$

De esto,

$$\begin{aligned} x \cdot (x^{\phi(n)})^t &= x(1 + u \cdot q) = x + x \cdot u \cdot q \\ &= x + (r \cdot p) \cdot u \cdot q, \text{ pues } x = r \cdot p \\ &= x + r \cdot u \cdot (p \cdot q) \\ &= x + r \cdot u \cdot n \end{aligned}$$

$$\Rightarrow x \cdot (x^{\phi(n)})^t \equiv x \pmod{n}$$

Sustituyendo esto en (*) tenemos que

$$d_{K_{pr}}(y) = (x^{d(n)})^t \cdot x \equiv x \pmod{n}$$

7.4 Cifrado y descifrado: Exponenciación rápida

Los algoritmos de llave pública están basados en aritmética con números muy largos

⇒ puedes fácilmente terminar con esquemas muy lentos para uso práctico

Recordemos que en RSA

$$\text{cifrado: } e_{K_{pub}}(x) \equiv x^e \pmod{n}$$

$$\text{descifrado: } x = d_{K_{pr}}(y) \equiv y^d \pmod{n}$$

y basado en exponenciación

Una forma directa de realizar exponenciación es la siguiente:

$$x \xrightarrow[\text{SQ}]{\substack{\text{elevar al cuadrado}}} x^2 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^3 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^4 \xrightarrow[\text{MUL}]{\substack{\text{multiplicación}}} x^5 \longrightarrow \dots$$

Como los exponentes e y d suelen ser muy grandes (≥ 2048 bits), hacer esto requiere de alrededor de 2^{2048} multiplicaciones o más

Esto es muchísimo (hay un estimado de 2^{300} átomos en el universo visible)

¿Hay formas más rápidas de hacer exponenciación? Si ☺

Ejemplo: square-and-multiply algorithm

Calcular x^8

$$\text{método directo: } x \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^2 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^3 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^4 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^5 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^6 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^7 \xrightarrow[\text{MUL}]{\substack{\text{MUL}}} x^8$$

$$\text{método s-m: } x \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^2 \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^4 \xrightarrow[\text{SQ}]{\substack{\text{SQ}}} x^8$$

Este algoritmo funciona bien pero solo puede utilizarse para exponentes de la forma 2^i (potencias de 2)

Ejemplo: Calcular x^{26} (26 no es potencia de 2)

método directo: $x \xrightarrow{\text{SQ}} x^2 \xrightarrow{\text{MUL}} x^3 \xrightarrow{\text{MUL}} x^4 \rightarrow \dots \xrightarrow{\text{MUL}} x^{26}$

otro método: $x \xrightarrow{\text{SQ}} x^2 \xrightarrow{\text{MUL}} x^3 \xrightarrow{\text{SQ}} x^6 \xrightarrow{\text{SQ}} x^{12} \xrightarrow{\text{MUL}} x^{13} \xrightarrow{\text{SQ}} x^{26}$

El segundo método es más rápido y nos da una forma de usar el square-and-multiply algorithm.

¿En qué orden usar SQ y MUL?

Hay un algoritmo que nos ayuda a determinar esto mediante los siguientes pasos:

1: Escanear los bits del exponente de izquierda a derecha.

2: En cada iteración de dicho escaneo

si el bit es $1 \Rightarrow \text{SQ}$ y MUL $\quad \quad \quad$ y después tomamos mod n
si el bit es $0 \Rightarrow \text{SQ}$

Ejemplo:

$$\text{Calculemos } x^{26} = x^{(11010)_2} = x^{(h_4 h_3 h_2 h_1 h_0)_2}$$

Empezamos de izquierda a derecha

Paso 0: $x = x^{(1)}$ (bit $h_4 = 1$)

Paso 1: SQ: $(x^1)^2 = x^2 = x^{(10)_2}$ (bit $h_3 = 1$)

MUL: $x^2 \cdot x = x^3 = x^{(10)_2} \cdot x^{(1)_2} = x^{(11)_2}$

Paso 2: SQ: $(x^3)^2 = x^6 = (x^{10_2})^2 = x^{1100_2}$ (bit $h_2=0$)

Paso 3: SQ: $(x^6)^2 = x^{12} = (x^{10_2})^2 = x^{1100_2}$ (bit $h_1=1$)
 MUL: $x^{12} \cdot x = x^{13} = (x^{100_2}) \cdot x^{1_2} = x^{1101_2}$

Paso 4: SQ: $(x^{13})^2 = x^{26} = (x^{101_2})^2 = x^{11010_2}$ (bit $h_0=0$)

Nota: SQ agrega 0 a la derecha del exponente

MUL cambia el último dígito del exponente de 0 a 1

Con estas dos operaciones vamos escribiendo el exponente en binario.

Pseudocódigo para el square-and-multiply algorithm para exponenciación modular

Paso 0: Tenemos el elemento base x , el exponente $H = \sum_{i=0}^t h_i 2^i$ con $h_i \in \{0, 1\}$ y $h_t=1$; y el módulo que estamos tomando n .

Paso 1: Tomamos $r = x$

Paso 2: Para $i = t-1$ hasta 0

$$r = r^2 \text{ mod } n$$

$$\text{si } h_i = 1$$

$$r = r \cdot x \text{ mod } n$$

Paso 3: Devolver $r = x^H$

¿Cuál es la complejidad de este algoritmo?

Sea H un exponente de longitud $t+1$ ($\log_2 H = t+1$)

- El número de SQ usados es independiente del valor de H (depende únicamente de la longitud de H)

- El número de MUL es igual al peso de Hamming de H (el número de 1's usados en la representación binaria)

Entonces, $\#SQ = t$

$$\# MUL \approx 0.5t \text{ (aproximación/promedio)}$$

¿Cuántas operaciones en promedio se necesitan para una exponenciación con un exponente de 2048 bits?

Usando el método directo: $2^{2048} \approx 10^{600}$ multiplicaciones ! Esto es imposible

Usando el square-and-multiply algoritmo:

$$1.5(2048) = 3072$$

elevaciones al cuadrado y multiplicaciones en promedio

complejidad logarítmica

complejidad lineal