RSA: Rivest, Shamir, Adleman

27 de octubre de 2016

- Diffie, W., and Hellman, M. *New directions in cryptography.* IEEE Trans. Inform. Theory IT-22, **(Nov. 1976)**, 644-654.
- Rivest, R.; A. Shamir; L. Adleman **(1978)**. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* . Communications of the ACM 21 (2): 120126

  The Original RSA Patent as filed with the U.S. Patent Office by Rivest; Ronald L. (Belmont, MA), Shamir; Adi (Cambridge, MA), Adleman; Leonard M. (Arlington, MA), **December 14, 1977**, U.S. Patent 4,405,829.

1 Criptosistema RSA

Criptosistema RSA: $(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$

- $\mathcal{M} = \mathbb{Z}_n$.
- $\mathcal{C} = \mathbb{Z}_n$.
- $\mathcal{K} = \{(n, p, q, e, d) : n = pq,\ p, q \text{ primos impares },\ ed \equiv 1 \mod \phi(n)\}$.
- Para $k = (n, p, q, e, d) \in \mathcal{K}$, $x, y \in \mathbb{Z}_n$:
  - $e_k(x) = x^e \mod n$,
  - $d_k(y) = y^d \mod n$.

1.1 Puesta en funcionamiento

Cada usuario realiza los siguientes pasos:

- Elige e. Generalmente se toma e primo e
  - igual a 65537 $(2^{16} + 1)$ si su uso es para cifrar.
  - igual a 3 $(2^1 + 1)$ si su uso es para firmar.
- Genera dos números primos aleatorios[1], p y q tales que $(e, p-1) = 1$, $(e, q-1) = 1$.
- Calcula $n = pq$.
- Calcula $d = e^{-1} \mod MCM(p-1, q-1)$. Puede utilizar la identidad de Bezout o el teorema de Euler (Bezout es más eficiente y no se necesita conocer la factorización de n).
- Calcula

$$\begin{array}{ll} d_p \equiv d \mod (p-1), & p_{inv} \equiv p^{-1} \mod q, \\ d_q \equiv d \mod (q-1), & q_{inv} \equiv q^{-1} \mod p, \end{array}$$

  Para recuperar M se calculan las clases de los restos $M_p = C^{d_p} \pmod p$ y $M_q = C^{d_q} \pmod q$, y se resuelve el sistema de congruencias:

$$\left.\begin{array}{ll} M \equiv M_p & \mod p \\ M \equiv M_q & \mod q \end{array}\right\}$$

[1] Ver más adelante recomendaciones

cuya solución es

$$M = M_p q_{inv} q + M_q p_{inv} p \mod n.$$

Resolver el sistema de congruencias es equivalente a calcular

$$h \equiv (M_p - M_q) q_{inv} \mod p$$

y

$$M = M_q + q\, h \mod n$$

- Hace público el par $\{e, n\}$. La clave privada es $\{d, p, q, d_p, d_q, q_{inv}\}$.

Ver RFC3447 Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography

Nota 1.1 *La tabla siguiente, correspondiente a una tarjeta inteligente, muestra la diferencia de tiempos entre usar el teorema chino y no hacerlo. También aparecen los tiempos para DES y ECDSA.*

SLE 66CX322P
136-Kbytes ROM, 5052 bytes RAM, 32-Kbytes EEPROM

Operation	**Modulus**	**Exponent**	**Calculation Time**		
			5 MHz	10 MHz	15 MHz
RSA Encrypt / Signature Verify	1024 bit	17 bit	20 ms	11 ms	7 ms
	2048 bit	17 bit	630 ms	315 ms	210 ms
RSA Decrypt / Signature Generate	1024 bit	1024 bit	820 ms	410 ms	273 ms
RSA Decrypt / Signature Generate using CRT	eq.1024 bit	eq.1024 bit	250 ms	125 ms	83 ms
	eq.2048 bit	eq.2048 bit	1840 ms	920 ms	614 ms
DSA Signature Generate	512 bit	160 bit	97 ms	49 ms	32 ms
DSA Signature Verify	512 bit	160 bit	117 ms	59 ms	39 ms
DSA Signature Generate	1024 bit	160 bit	438 ms	219 ms	146 ms
DSA Signature Verify	1024 bit	160 bit	711 ms	356 ms	237 ms

Operation	**Data Block Length**	**Encryption time, incl. data transfer**		
		5 MHz	10 MHz	15 MHz
56-bit Single DES Encryption	64 bit	23 μs	11 μs	8 μs
112-bit Triple DES Encryption	64 bit	35 μs	17 μs	12 μs

Operation	**Operand Length**	**Calculation Time**		
		5 MHz	10 MHz	15 MHz
EC-DSA GF(2^n) Signature Generate	192 bit	285 ms	142 ms	95 ms
EC-DSA GF(2^n) Signature Verify	192 bit	540 ms	270 ms	180 ms

1.2 Cifrado y descifrado de mensajes

Para enviar un mensaje cifrado a una persona se siguen los pasos:

- Se busca la clave pública $\{e, n\}$ de la persona a la que se quiere enviar el mensaje cifrado.
- Se codifica el mensaje m (cadena de bits) transformándolo en EM (encoded message) que es el que se cifrará con el RSA.
  - ◇ PKCS1-V1-5[2]
    - Generate an octet string PS of length $k - mLen - 3$ (k length of n, mLen length of m) consisting of pseudo-randomly generated nonzero octets. The length of PS will be at least eight octets.
    - Concatenate PS, the message m, and other padding to form an encoded message EM of length k octets as

      $$EM = \text{0x00} \parallel \text{0x02} \parallel \text{PS} \parallel \text{0x00} \parallel m.$$

  - ◇ OAEP (Optimal asymmetric encryption padding)
    - El padding consiste en una cadena de 0 o más bytes 0x00 acabada necesariamente con el byte 0x01.
    - pHash es el hash de los parámetros que se le pasan, si no hay parámetros es el hash de una cadena vacía.
    - mask generation function MGF(Z,l) devuelve una cadena de l bytes generados de la siguiente forma:
      1. Sea T una cadena vacía
      2. Para i=0...$l/hLen$-1
         $T = T \| Hash(Z \| i)$ (i entero de 32 bits)
      3. Salida los primeros l bytes de T.
    - La función $Hash(\cdot)$ que devuelve $hLen$ bytes se ha de acordar previamente al igual que el tamaño de la semilla.

[2] No se recomienda su uso, sólo por compatibilidad

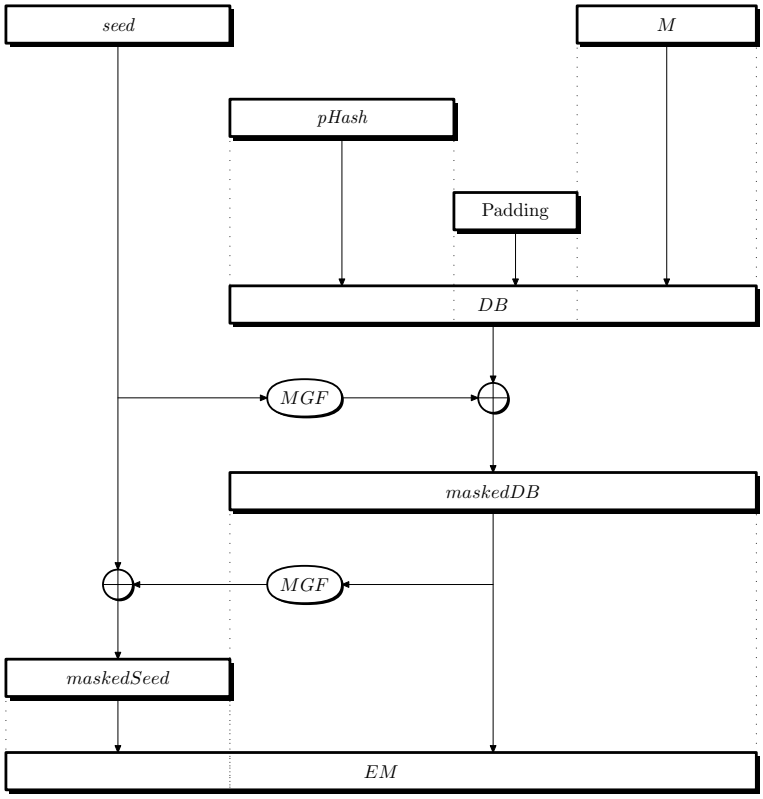


Figure C.1: *EME-OAEP encoding operation. pHash is the hash of the encoding parameters.*

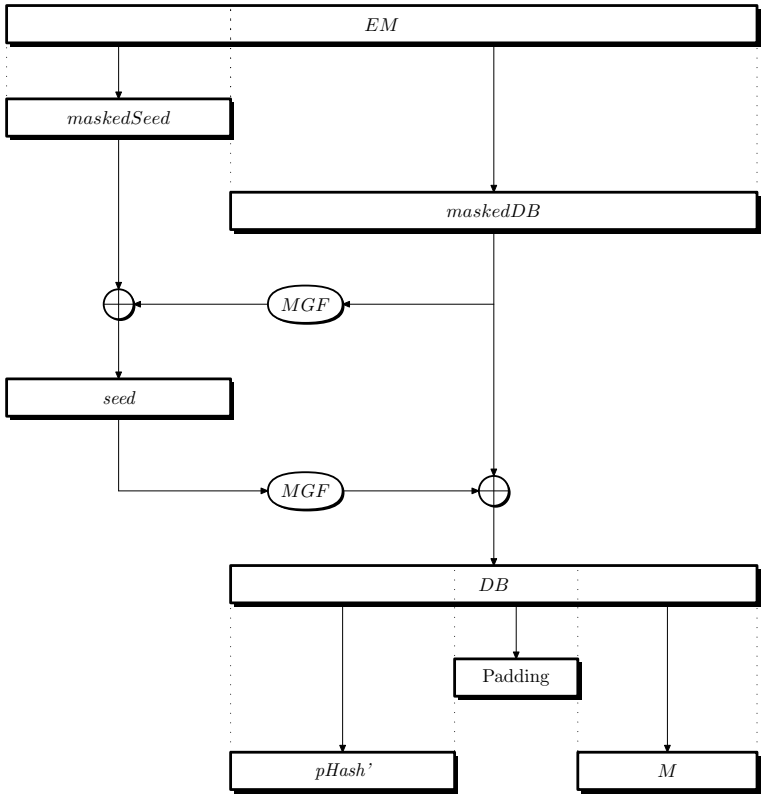


Figure C.2: *EME-OAEP decoding operation. If pHash' equals the hash of the encoding parameters, then decryption is considered successful.*

- M se considera un elemento de $\mathbb{Z}_n$ y se cifra.

Para descifrar un mensaje cifrado recibido se siguen los pasos:

- Se descifra c utilizando la regla de descifrado y la clave privada del receptor del mensaje.
  Utilizando el teorema chino para descifrar el tiempo necesario se reduce aproximadamente a la tercera parte (ver PFC Jordi Gené León y www.semiconductors.philips.com/identification/products/we/security).

- Se comprueba que EM tiene el formato adecuado. Si es así, se extrae m.

Nota 1.2 *EL RSA estaba patentado en USA hasta el 2000, por lo que había que pagar una licencia para utilizarlo allí.*

Nota 1.3 *Lo usual es que los mensajes se cifren utilizando criptosistemas simétricos y que la clave utilizada en dicho criptosistema sea cifrada utilizando un criptosistema asimétrico. La clave del criptosistema simétrico cifrada utilizando un criptosistema asimétrico recibe el nombre de* sobre digital*.*

2 Seguridad

2.1 Ataques al uso del RSA

2.1.1 Exponente compartido

Supongamos que enviamos el mismo mensaje m a tres personas cuyo exponente público es $e = 3$:

$$\begin{aligned} c_1 &\equiv m^3 \mod n_1, \\ c_2 &\equiv m^3 \mod n_2, \\ c_3 &\equiv m^3 \mod n_3. \end{aligned}$$

Si $(n_1, n_2) = ((n_1, n_3) = n_2, n_3) = 1$ por el teorema chino existe una única solución del sistema

$$\begin{aligned} x &\equiv c_1 \mod n_1, \\ x &\equiv c_2 \mod n_2, \\ x &\equiv c_3 \mod n_3, \end{aligned}$$

módulo $n = n_1 n_2 n_3$. Esa solución es exactamente igual (no congruente) a m^3. Podemos hallar m calculando la raíz cúbica en los reales (es fácil y rápido) del resultado obtenido aplicando el chino.

El siguiente teorema nos resume los peligros de compartir el exponente público.

Teorema 2.1 *Sean $c_1, \ldots, c_e$ los criptogramas resultantes de cifrar, con claves públicas RSA $\{e, n_1\}, \ldots, \{e, n_e\}$ tales que $n_1 < \ldots < n_e$ y $(n_i, n_j) = 1$ si $i \neq j$, un mismo mensaje $0 \leq m < n_1$.*

Entonces, el tiempo necesario para hallar m a partir de los criptogramas y las claves públicas es $O\left(\log^3(n_1) + \log^2(n)\right)$ siendo $n = \prod_{i=1}^{e} n_i$.

Si los módulos no son relativamente primos, cosa altamente improbable, podemos factorizarlos (su m.c.d nos da el factor primo que comparten.)

2.1.2 Módulo compartido

Debido al coste de la generación de números primos, una gran organización podría estar tentada de distribuir entre sus miembros un par de exponentes, uno público y su correspondiente exponente privado, y un módulo común a todos los usuarios, pero no los valores de los primos que lo componen. De esta forma, se ahorraría la generación de un par de números primos para cada usuario y todos los miembros de la organización dispondrían de la capacidad de cifrar y descifrar mensajes, aunque esta última operación no fuese muy eficiente.

En esta situación podemos distinguir dos tipos de ataques, según su procedencia.

Atacante externo al sistema

Teorema 2.2 *Sean c_1 y c_2 los criptogramas resultantes de cifrar, con las claves públicas RSA $\{e_1, n\}$ y $\{e_2, n\}$ tales que $(e_1, e_2) = 1$, un mismo mensaje $0 \leq m < n$.*

Entonces, el tiempo necesario para hallar m a partir de los criptogramas y las claves públicas es $O\left(\log^3(n)\right)$.

Demostración: Como $(e_1, e_2) = 1$ podemos hallar r y s tales que $e_1 r + e_2 s = 1$. Entonces

$$c_1^r c_2^s \equiv m^{e_1 r} m^{e_2 s} \equiv m^{e_1 r + e_2 s} \equiv m \mod n.$$

Las operaciones más costosas son las exponenciaciones modulares. □

Atacante interno al sistema Un atacante interno dispone de más información que un atacante externo. El conocimiento de un exponente público y su correspondiente exponente privado le permite romper completamente el sistema.

`Cálculo de exponentes privados`

Teorema 2.3 *Sean n un módulo RSA, $3 \leq e_A, d_A < \phi(n)$ enteros tales que $e_A d_A \equiv 1 \mod \phi(n)$, y $3 \leq e_B < \phi(n)$ un entero tal que $(e_B, \phi(n)) = 1$.*

Entonces, el tiempo necesario para hallar un entero d_B tal que $e_B d_B \equiv 1 \mod \phi(n)$ es $O\left(\log^3(n)\right)$.

Demostración: Busquemos un entero α con las siguientes propiedades:

1. $\alpha = k\,\phi(n)$,
2. $(\alpha, e_B) = 1$.

Entonces, por Bezout podremos hallar r y s tales que $\alpha\, r + e_B s = 1$. Como α es un múltiplo de $\phi(n)$, tomando módulo $\phi(n)$ podremos concluir que $e_B s \equiv 1 \mod \phi(n)$.

Definamos

$$g_0 = e_A d_A - 1, \qquad h_0 = (g_0, e_B),$$

y recursivamente

$$g_i = \frac{g_{i-1}}{h_{i-1}}, \qquad h_i = (g_i, e_B).$$

Notemos que g_i siempre es entero ya que h_{i-1} es un divisor de g_{i-1}. Sea t el menor entero tal que $h_t = 1$, (observemos que si $h_k \neq 1$ entonces $h_k \geq 2$ y $g_{k+1} \leq g_k/2$ y, por lo tanto, en un número de pasos del orden de $\log e_A d_A - 1$ tendremos que alguna h_j será 1.)

Tal como hemos tomado t, $h_t = 1$ y, por lo tanto, $(g_t, e_B) = 1$. Por otra parte $g_t = (e_A d_A - 1)/h_0 \ldots h_{t-1} = \lambda\phi(n)/h_0 \ldots h_{t-1}$; como $h_i = (g_i, e_B)$ y $(\phi(n), e_B) = 1$ tenemos que ningún h_i divide a $\phi(n)$ y, por lo tanto, que todos los h_i dividen a λ. En consecuencia, g_t es un múltiplo de $\phi(n)$ (por la misma razón, no puede ser igual a 1).

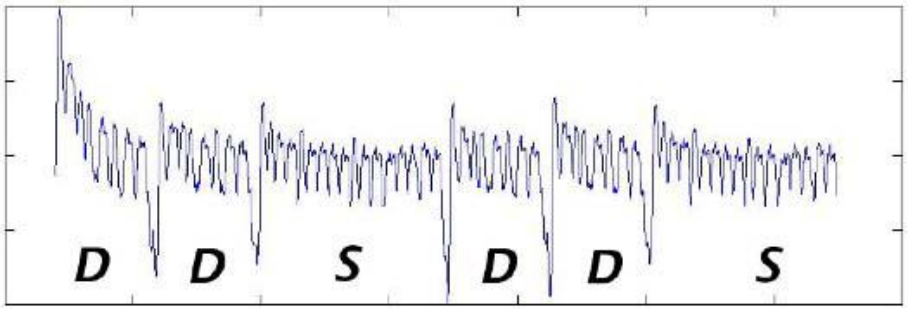
Resumiendo, g_t cumple los requisitos pedidos para α. Las operaciones más costosas son los cálculos de m.c.d y de la identidad de Bezout. □

2.1.3 Timing / Power attacks

Para calcular $a^m \pmod n$, a, $m > 0$ i $n > 0$ enteros, se considera a expressión del exponente en base 2, $m = b_r 2^r + \cdots + b_1 2 + b_0$, donde $b_i \in \{0, 1\}$ y $b_r = 1$, y se usa el siguiente algoritmo:

$y = 1,\ i = r$
mientras $i \geq 0$ **hacer**
  $y = y \cdot y \mod n$
  si $b_i = 1$ **hacer** $y = y \cdot a \mod n$
  $i = i - 1$
salida y

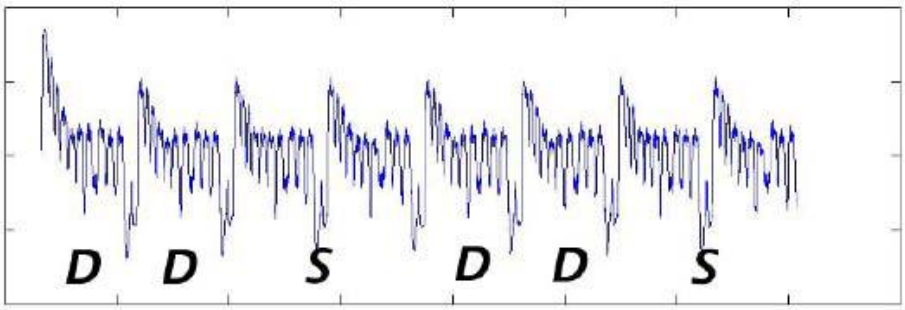
Este algoritmo consume en cada iteración más tiempo/energía si el bit del exponente es 1 que si es 0. Mediante un análisis estadístico del tiempo/energía consumido es posible calcular la el exponente.



Para evitar este ataque se puede utilizar el método de Montgomery para la exponenciación (Montgomery Powering Ladder):

$R_0 = 1,\ R_1 = a,\ i = r$
mientras $i \geq 0$ **hacer**
  si $b_i = 0$ **hacer**
    $R_1 = R_0 \cdot R_1 \mod n$
    $R_0 = R_0 \cdot R_0 \mod n$
  si $b_i = 1$ **hacer**
    $R_0 = R_0 \cdot R_1 \mod n$
    $R_1 = R_1 \cdot R_1 \mod n$
  $i = i - 1$
salida R_0

Este algoritmo consume en cada iteración el mismo tiempo/energía independientemente del valor del bit del exponente.



Para ver que son equivalentes definamos $L_j = \sum_{i=j}^{r} b_i 2^{i-j}$ y $H_j = L_j + 1$, entonces:

$$L_j = 2L_{j+1} + b_j = L_{j+1} + H_{j+1} + b_j - 1 = 2H_{j+1} + b_j - 2$$

y por lo tanto

$$(L_j, H_j) = \begin{cases} (2L_{j+1}, L_{j+1} + H_{j+1}) & \text{si } b_j = 0, \\ (L_{j+1} + H_{j+1}, 2H_{j+1}) & \text{si } b_j = 1, \end{cases}$$

R_0 guarda a^{L_j} y R_1 guarda a^{H_j}.

Tal vez sea más fácil observando que siempre se cumple que $R_1 = a\, R_0$.

2.1.4 Criptograma escogido (reto-respuesta)

Supongamos que un atacante desea descifrar un criptograma $c \equiv m^e \mod n$ cifrado con la clave pública de A.

El atacante elige un entero aleatorio k y calcula $\tilde{c} \equiv ck^e \mod n$. Se pone en contacto con A y le pide que descifre/firme $\tilde{c}$ para que demuestre su identidad. A calculará

$$\tilde{c}^d \equiv (ck^e)^d \equiv (m^e)^d k \equiv m\, k \mod n$$

y enviará $m\, k$ al atacante que sólo tendrá que multiplicar por $k^{-1} \mod n$ para obtener el mensaje m.

Para evitar este ataque no se cifra m sino mensajes del tipo $\tilde{m} = m \| padding$.

2.1.5 Insuficiente control del formato

Bleichenbacher's RSA signature forgery based on implementation error

Suponganos que para firmar un mensaje el formato de los datos a firmar es:

`0x00 0x01 0xFF...0xFF 0x00` ‖ `Codificación ASN.1 del hash del mensaje`

La Codificación ASN.1 lleva el tamaño del hash. Algunas implementaciones no comprueban que tras el hash del mensaje no hay nada más. Esto permite construir un mensaje a firmar con la siguiente estructura:

`0x00 0x01 0xFF...0xFF 0x00` ‖ `hash del mensaje en ASN.1`‖ `Algo`

de forma que sea un cubo exacto (fácil de calcular, no es fácil calcular raíces módulo N). La raíz cúbica pasaría como firma válida de cualquiera que tenga un exponente público igual a 3 y módulo suficientemente grande[3] ya que al calcular el cubo (verificar la firma) obtendría el hash correcto de la codificación ASN.1 pero no comprobaría que detrás de ella hay más bits.

2.1.6 Error provocado

Supongamos un dispositivo que calcula una firma utilizando el teorema chino al que inducimos a que, una vez calculada correctamente la firma módulo p, cometa un error al calcular la firma módulo q. El dispositivo devolverá una firma S correcta módulo p pero incorrecta módulo q. Entonces

$$p = \text{mcd}(n, S^e - m)$$

siendo m el mensaje firmado.

[3]el número de bits de `Algo` será ligeramente mayor que 2/3 (si el exponente es 3) del número de bits del módulo, por lo tanto `0x00 0x01 0xFF...0xFF 0x00` ‖ `hash del mensaje en ASN.1` ha de tener algo menos de 1/3 del número de bits del módulo.

2.2 Recomendaciones

1. El tamaño de p y q debe de ser de unos 1576 bits y el de n no debe ser inferior a 3072 bits, recomendándose módulos de tamaño mayor.

2. $|p - q|$ no debe de ser pequeño.

3. p y q deben ser primos fuertes. p es un número primo fuerte si es primo y además:

   (a) $p - 1$ tiene un factor primo grande, r;

   (b) $p + 1$ tiene un factor primo grande;

   (c) $r - 1$ tiene un factor primo grande.

4. No hay que compartir módulo.

5. El exponente público no debe ser excesivamente pequeño (17 bits como mínimo) y *reconocible*.

6. Tener mucho cuidado en cómo se utiliza una primitiva, lo que en una situación puede ser inocuo en otra puede comprometer el sistema.

7. Una clave sólo ha de tener un uso (cifrado, firma, autentificación...)