

Consideraciones Prácticas

Logros en Factorización

Muchos sistemas modernos basan su seguridad en la supuesta dificultad de un problema matemático, ej. factorización para RSA, logaritmos discretos para Diffie-Hellman, etc. Es importante notar que hasta ahora ninguno de estos problemas tiene dificultad *conocida*.

“La factorización de un número de 125 digits tardará 40 quadrillones de años.” – Ron Rivest, 1977

En Agosto de 1999 un grupo de investigadores terminó de factorizar el número RSA-155 (de 155 dígitos decimales). Tardaron casi 4 meses de tiempo calendario, usando 160 estaciones SGI y Sun de 175-400, 8 procesadores Origin de 250 MHz de SGI, 120 Pentium II de 300-450 MHz, y 4 Compaq Alphas de 500 MHz, un total de aproximadamente 8000 Mips-años.

(Curiosamente, cada avance en factorización es acompañado por uno similar en el cálculo de logaritmos discretos; no está claro por qué.)

¿Cuántos bits de clave debemos usar para RSA? En Noviembre de 2001 un panel del NSA estimó que 1024 bits de clave serían suficientes hasta el año 2015, asumiendo la Ley de Moore y avances razonables en algoritmos de factorización.

Problemas con Sistemas Asimétricos

Los sistemas asimétricos tienen varios problemas:

- Son lentos. Los simétricos son 3 ordenes de magnitud más rápidos.
- Son vulnerables a ataques “texto escogido”. En casos extremos el adversario pueden encriptar todo posible texto original y comparar. Ej. si el texto consiste de un número entre 1 y 5000, esto puede ser muy efectivo. ¿Qué podemos hacer para aumentar la dificultad?

Debido a la lentitud, usualmente se usan claves públicas para acordar una **clave de sesión**, la cual se utiliza con un sistema simétrico. Uno de los ejemplos más conocidos de esta técnica es el sistema PGP, que vemos a continuación.

Números Aleatorios

En muchas aplicaciones de la criptografía, la seguridad depende de que ciertos elementos (ej. claves) sean *aleatorias*. Para que un número o secuencia de bits sea aleatoria, debe generarse a partir de una fuente natural de entropía:

- Un proceso radioactivo
- El voltaje a través de un diodo invertido (ej. el Pentium III)
- `cat /dev/audio | gzip -dc > random`

En la práctica, puede ser difícil acceder a estas fuentes, y tenemos que utilizar procesos determinísticos llamados **PRNG** (*Pseudo-Random Number Generators*) o **PRBG** (*Pseudo-Random Bitsream Generators*).

Las PRNG o PRBG tradicionales fueron desarrollados para el área de la simulación de procesos, donde lo que importa es el comportamiento estadístico. En aplicaciones criptográficas, se agrega el requisito que el producto del proceso no sea sujeto a extrapolación. Por ejemplo, si usamos un PRBG tradicional llamado el *Linear Congruential Generator*, y Manuel logra encontrar parte de la secuencia de bits que produce, puede fácilmente “sincronizarse” con el generador y seguir produciendo los mismos bits que Alicia.

El Generador BBS

Uno de los PRBG más populares y seguros es el de Blum, Blum y Shub, llamado **BBS**:

Sean p y q dos primos tal que $p \equiv q \equiv 3 \pmod{4}$, y $n = pq$. Escoger como *semilla* s_0 , cualquier entero relativamente primo a n . Para $i \geq 0$, definamos:

$$s_{i+1} = s_i^2 \pmod{n}$$

y

$$z_i = s_i \pmod{2} = (s_0^{2^i} \pmod{n}) \pmod{2}$$

La secuencia de bits z_i (derivada de los bits menos significativos de los s_i) es pseudo-aleatoria con alto grado de seguridad. Es más, se puede mejorar la eficiencia sin menoscabo a la seguridad, tomando m bits menos significativos a la vez, donde $m \leq \log_2 \log_2 n$.

También se puede mostrar que $s_i = s_0^{2^i \pmod{(p-1)(q-1)}}$. Esto significa que se puede calcular s_i sin tener que calcular todos los anteriores, lo cual es útil para aplicaciones en las cuales se necesita acceso no-secuencial a los datos, ej. en un archivo o base de datos. (Nótese que sólo necesitamos algunos de los bits menos significativos de s_i .)

Firmas Digitales

¿Qué propiedades tiene una firma convencional (ideal)?

- Es auténtica – convence al observador.
- Es imposible falsificar.
- No es reutilizable para otro documento distinto.
- El documento firmado no es alterable.
- La firma no puede ser repudiada.

Para aplicar estas ideas en un sistema criptográfico simétrico se puede utilizar a Arturo (un árbitro). Arturo comparte una clave secreta K_A con Alicia, y otra, K_B con Bob:

- Alicia manda $C = K_A(M)$ a Arturo.
- Arturo decripta C.
- Arturo manda $K_B([M, \text{“recibí esto de Alicia”}])$ a Bob
- Bob decripta.

Cualquier disputa se resuelve apelando a Arturo, quien mantiene registros de todos los mensajes. El problema práctico es que Arturo tiene que ser completamente confiable y seguro.

Firmas con Sistemas de Clave Pública

En algunos sistemas asimétricos (ej. sistemas de **mochila**) el espacio de C es mayor que el de M , por lo tanto los algoritmos para firmas son distintos a los de privacidad.

En otros sistemas, generalmente basados en **aritmética modular**, los dos espacios coinciden:

1. Alicia encripta M con su clave privada, y lo manda a Bob
2. Bob decripta con la clave pública de Alicia, para verificar la firma

No se necesita el árbitro, ni siquiera en casos de disputas. Este método cumple con todos los atributos de una firma digital, excepto si Alicia trata de repudiar su firma (alegando robo de su clave privada).

Sin embargo, Bob sí puede hacer trampa: reutilizar un documento ya firmado por Alicia. Si cada instancia del documento tiene validez propia (ej. un cheque), tiene que ser protegido con un **timestamp**.

Firmas Digitales Prácticas

Si el documento es largo, calcular la firma puede ser muy ineficiente. Podemos hacer esto:

1. Alicia calcula una firma $F = E(H(M))$, donde $H()$ es una función de hashing unidireccional, usando su clave privada para encriptar.
2. Alicia envía $[M, F]$ a Bob
3. Bob calcula $H(M)$, y decripta F usando la clave pública de Alicia. Si son iguales, la firma es válida.

Importante: si $H()$ no fuese unidireccional, sería fácil crear múltiples documentos con el mismo F , lo cual permitiría copiar (falsificar) la firma múltiples veces.

Usaremos $F_A(M)$ para indicar la firma del mensaje M con la clave privada de Alicia, usando algún algoritmo.

Múltiples Firmas

Usando hashing unidireccional, podemos obtener firmas de Alicia y Bob a un documento que van a mandar a Carol:

1. Alicia calcula su firma, $F_A(M)$
2. Bob calcula su firma, $F_B(M)$, y la manda a Alicia
3. Alicia manda $[M, F_A, F_B]$ a Carol
4. Carol verifica las dos firmas

La ventaja es que Carol puede verificar una de las firmas sin tener que verificar la otra.

Privacidad con Firmas

¿Cómo podemos tener mensajes secretos y firmados?

1. Alicia calcula $E_B(F_A(M))$, encriptando con la clave pública de Bob, y envía esto a Bob (recuerde que esto significa $E_B([M, D_A(H(M))])$)
2. Bob decripta con su clave privada:
 $D_B(E_B(F_A(M))) = F_A(M)$
3. Bob verifica con la clave pública de Alicia y recupera el mensaje: $V_A(F_A(M)) = M$

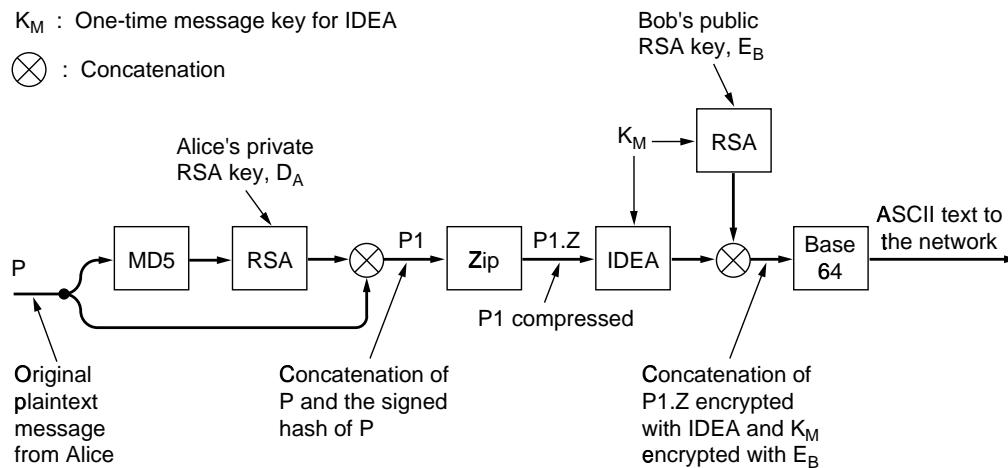
Importante: hay que firmar antes de encriptar, no al revés. Si no, algunos sistemas (ej. RSA) son vulnerables a ciertos tipos de ataque.

En sistemas prácticos, se usan dos juegos de claves, uno para autenticidad y otro para privacidad, y no necesariamente con los mismos algoritmos criptográficos.

Ejemplo: PGP – Pretty Good Privacy

Creado por Phil Zimmermann (<http://www.pgp.net> y <http://www.pgp.com>):

- Libre distribución: el autor quería que el público tuviera acceso a criptografía fuerte (“Si la privacidad es ilegal, sólo los ilegales tendrán privacidad”).
- Disponible para Unix, Mac, Windows ...
- Orientado a correo electrónico privado y/o firmado, pero no limitado a ello.
- **Web of trust** en vez de CA’s.
- Utiliza RSA (para autenticidad), MD5 (para integridad) e IDEA (para privacidad). Versión 5 también ofrece otras alternativas.
- Tiene facilidades para administración personal de claves.
- Versión “internacional” disponible fuera de EE.UU. es idéntica en funcionalidad a la versión “nacional”, e interoperable con la misma.



- Alicia quiere mandar el mensaje P a Bob. PGP calcula un hash de P usando MD5.
- Si Alicia quiere firmar el mensaje, PGP utiliza RSA con la clave privada de Alicia para firmar el resultado del MD5.
- Si Alicia quiere privacidad:
 - La firma, si existe, se concatena con P . Se comprime usando el algoritmo de Ziv y Lempel (conocido como ZIP).
 - Se genera una clave aleatoria de “sesión”, usando el teclado como fuente de entropía. Tanto los tiempos como el texto se utilizan para generar 128 bits, K_M .
 - Un algoritmo simétrico, en modo CFB con un IV de 0, encripta el mensaje comprimido con la clave K_M . Además, K_M es encriptada con la clave pública de Bob.
- El resultado final se codifica en ASCII usando **base64**. Este paso es opcional.

Manejo de Claves:

PGP soporta 3 tamaños de clave RSA: **casual** (384 bits), **comercial** (512 bits), y **militar** (1024 bits).

This whole business of protecting public keys from tampering is the single most difficult problem in practical public key applications. – Phil Zimmermann

PGP mantiene dos estructuras locales en la máquina de Alicia:

- El **llavero privado** contiene uno o más pares de claves públicas y privadas. Alicia puede tener varios pares, cada uno con un identificador (que consiste de los 64 bits más a la derecha de la clave pública). Todo esto se guarda en disco, encriptado a su vez usando una contraseña de longitud arbitrario.
- El **llavero público** contiene las claves públicas de los corresponsales de Alicia, además de sus identificadores respectivos y una indicación del nivel de confianza que Alicia tiene en cada una. Este último dato se usa porque las claves vienen de distintos fuentes. Cada clave obtenida de otra persona está firmada con la clave pública de las personas que la certifican.

Un Ataque contra RSA

- Manuel graba un mensaje m de Bob a Alicia: $c = m^e \pmod n$.
Escoge un número aleatorio $r < n$ y calcula:

$$x = r^e \pmod n$$

$$y = xc \pmod n$$

$$t = r^{-1} \pmod n$$

Si $x = r^e \pmod n$, entonces $r = x^d \pmod n$.

- Ahora Manuel convence a Alicia para que firme y (*ojo*: no un *hash* de y) con su clave privada, lo cual decripta a y . Alicia manda a Manuel: $u = y^d \pmod n$.
- Ahora Manuel calcula:

$$\begin{aligned} tu \pmod n &= r^{-1}y^d \pmod n \\ &= r^{-1}x^d c^d \pmod n \\ &= c^d \pmod n \\ &= m \end{aligned}$$

Ahora Manuel tiene m .

Esto es un ejemplo de un **chosen ciphertext attack**. Hay varias versiones. La moraleja es: *Nunca firmar un documento presentado por un extraño*. En cambio, firmar un “hash” unidireccional del documento no tiene problema.