Instituto de Matemáticas, UNAM
Publicaciones Preliminares
No. **631** (marzo 1999)

Breve Introducción a Criptografía y Seguridad

(Versión Preliminar)*

Sergio Rajsbaum†

Resumen

Se presenta una breve introducción a los conceptos básicos de seguridad y criptografía, pretendiendo ilustrar, por un lado, la complejidad de esta área que abarca desde conceptos matemáticos sofisticados, hasta serios problemas de software y sistemas, pasando por problemas de diseño y análisis de algoritmos, e inclusive consideraciones legales, éticas y comerciales, y donde todo esto debe interactuar como un todo para lograr un sistema seguro. Por otro lado, se espera lograr que se aprecie la riqueza y lo fascinante del área. Finalmente, se pretende destacar la importancia, cada vez mayor, que la seguridad y criptografía van tomando ya inclusive en sistemas de uso cotidiano.

1 Introducción

La seguridad de los sistemas computacionales siempre ha sido una preocupación, como debe ser en cualquier obra de ingeniería suficientemente compleja. Es necesario tener en cuenta los riesgos ocasionados por las fallas originadas en el inevitable error humano, por los márgenes de error que el mismo diseño del sistema contempla, e inclusive por ataques maliciosos. Sin embargo, en las últimas décadas, con el advenimiento de los sistemas distribuidos, y los avances en redes de computadoras, los problemas de seguridad se han vuelto mucho más complejos y variados. Hoy en día es importante que incluso el usuario común de un sistema distribuido tenga un entendimiento básico de seguridad y criptografía, que apenas hace algunas décadas era motivo de anécdotas fascinantes solamente con espías y militares como participantes [15]. De hecho, son sólo unos 20 años que hay investigación abierta y generalizada en todo el mundo en criptología. Y sin embargo, se cree que todavía se hace mucho más investigación en criptología en secreto por agencias gubernamentales, que pública, y que el conocimiento que se tiene

*Versión final por aparecer en *Soluciones Avanzadas*.

†`rajsbaum@matem.unam.mx`, `http://www.matem.unam.mx/~rajsbaum`. Instituto de Matemáticas, UNAM, Ciudad Universitaria, D. F. 04510, México.

en estas agencias está más adelantado que el que se tiene publicado (en criptología, no en seguridad en general). Por ejemplo, se dice que la NSA (National Security Agency de EUA), –de la cual inclusive su existencia era considerada información clasificada en un principio– es el sitio con más matemáticos en el mundo, y el mayor comprador de hardware en el mundo, y que tiene capacidad de romper muchos de los sistemas usados en la práctica. Por otro lado, el simple uso de una tarjeta de cajero automático o un teléfono celular, involucra protocolos criptográficos; cada día son más los usos cotidianos de la criptografía.

Se presenta aquí un breve panorama de conceptos básicos de seguridad y criptografía, y proveyendo algunos apuntadores a la increíble, y velozmente creciente bibliografía del tema. Sería imposible pretender cubrir todos los aspectos de este fascinante tema, o profundizar en alguno de ellos, en un documento de este tamaño. Más bien, el objetivo es, por un lado, despertar el interés del lector, y por otro lado, mostrar la magnitud y complejidad de éste, que se puede ya considerar un campo de estudio por sí mismo, *seguridad de sistemas computacionales*. Inclusive algunos aspectos importantes de la seguridad de un sistema no son discutidos aquí, para no alargar aún más la exposición, como son: la seguridad física de los equipos de cómputo y sus instalaciones, técnicas de manejo de personal, técnicas de "tapping", (intercepción de líneas de comunicación) los aspectos de evaluación de riesgos, aspectos regulatorios, legales y éticos, estandarización, patentes, administración y sistemas particulares (unix, windows), etcétera. Se puede leer al respecto de algunos de estos puntos en [4, 12], por ejemplo. Pero es necesario enfatizar que la funcionalidad y la seguridad de un sistema depende de todos y cada uno de sus componentes, por lo que no es posible menospreciar ningún aspecto.

Para dar sólo una idea de lo que involucra el área de seguridad, considérese el concepto de una *firma digital*. Ya existe una variedad de maneras de implementarse usando criptografía, y que ya están en uso actualmente. Es posible tomar un documento electrónico, y crear una firma digital que tiene todas las propiedades que uno espera de un documento de papel firmado, e inclusive mejoradas. Inmediatamente se puede apreciar la extensión del problema que significa construir un sistema que permita el uso de firmas digitales a nivel nacional o internacional. Involucra una diversidad de aspectos, algunos de los cuales se discutirán más adelante: el diseño de los algoritmos criptográficos para firmar; la administración de las llaves privadas para hacer las firmas, y la de las llaves públicas para verificar la validez de éstas; protocolos para generación de llaves, y para verificación, para dos o más participantes; renovación de llaves; políticas de uso; mecanismos de recuperación en caso de problemas; integración a diversas plataformas de cómputo; estandarización, etcétera. Surgen preguntas como ¿cuál es el valor legal de una firma digital? ¿cómo hacer notarios electrónicos? ¿se podrá comprobar que una firma digital fue falsificada? ¿cómo fijar responsabilidades cuando firmas digitales son producidas automáticamente (como ya sucede en ocasiones hoy en día) por una programa de computadora?

Finalmente, es necesario destacar la importancia de estar constantemente actualizándose. Al ir detectando nuevos problemas de seguridad en un sistema, se van diseñando modificaciones que los resuelven. En ocasiones las modificaciones conllevan cambios más sustanciales, o incluso conceptuales a las estrategias de seguridad. Nuevos virus surgen. Nuevas precauciones que tomar en el uso de algoritmos criptográficos. Algunos algoritmos dejan de ser seguros, otros nuevos aparecen. Nuevas recomendaciones en cuanto al tamaño de llaves secretas, dependiendo de niveles de seguridad requeridos. Nuevas técnicas de hardware y software. Nuevas tecnologías que requieren de nuevos conceptos (piénsese por ejemplo en web o comercio electrónico, que se discuten más adelante). Una buena fuente de información acerca de nuevos errores y correcciones son las páginas web de compañías que producen el software correspondiente. CERT [6] es un centro en la universidad de Carnegie Mellon, dedicado a ayudar a detectar y prevenir problemas de seguridad, facilitando comunicación entre expertos y desarrollo de soluciones.

En la primera parte de este documento se introducen consideraciones generales de seguridad en sistemas distribuidos. En la segunda parte se describen las piezas básicas a partir de las cuales se construyen los sistemas de seguridad; es decir, los algoritmos criptográficos. A partir de ellos se construyen protocolos –también descritos en esta parte– que pueden involucrar a dos o más participantes, y que proveen de bloques útiles para construir las capas superiores del sistema. Finalmente, en la tercera parte, se presentan algunos ejemplos de sistemas en los cuales la seguridad tiene un papel fundamental, y que usan las herramientas descritas en la segunda parte.

2 Seguridad de Sistemas Distribuidos

El diseño de un sistema seguro es un problema muy complejo. Se tienen que tomar en cuenta todas los componentes del sistema. De nada sirve instalar tres chapas en la puerta de una casa, si hay una puerta atrás fácil de abrir, o si se deja una copia de las llaves abajo del tapete de entrada por si se llegan a olvidar las llaves. Además, los problemas de seguridad plantean requerimientos contradictorios. Por un lado, quisiéramos que la casa fuera lo más segura posible, pero por otro lado, lo que queremos es usar la casa de la manera más cómoda y eficiente. Por un lado, queremos poner una puerta de hierro con varias chapas, por otro lado, queremos una puerta barata y bonita, de fácil acceso. Si le ponemos cinco chapas, probablemente terminemos usando sólo una, por la incomodidad de tener que cargar tantas llaves, y manipular tantas chapas cada vez.

La seguridad de un sistema no está garantizada solamente porque se utilicen algoritmos criptográficos de muy alta calidad. Depende también del diseño de la arquitectura del sistema que especifica la manera en que estos algoritmos interactúan con el resto de los componentes del sistema. Y aquí hay que incluir también a todos los componentes humanos, como operadores y usuarios finales. Muchas veces los ataques que más preocupan son de intimidación y corrupción del dueño de una llave secreta, errores en el sistema operativo o en software de aplicación, errores humanos de operación, problemas de tipo sociológico, etcétera. Existen ejemplos en la industria bancaria de sistemas seguros criptográficamente, que fallan al exponerlos a las amenazas del mundo real [1]. Los Aliados rompieron códigos alemanes en gran parte explotando errores de operadores [23].

En sistemas tan grandes y complejos como los que existen ahora, los problemas de seguridad son enormes. Sistemas de software en un estado que deja mucho que desear, de cientos de miles de líneas de programación, de las cuales cualquiera es un punto de riesgo potencial a la seguridad del sistema. Errores que no son malintencionados pueden resultar en el futuro en problemas y se vuelve muy difícil saber de donde provienen. Hay que tener cuidado inclusive con componentes que pudieran parecer irrelevantes a la seguridad del sistema. Por ejemplo, el uso de métodos de generación de números aleatorios de poca calidad es uno de los puntos de ataques potenciales.

Las lecciones que ha dado la historia, desde lo descrito en el clásico de Kahn [15] hasta la fecha, es que se encuentran fallas hasta en los sistemas considerados más seguros, y que es necesario respetar las limitaciones de la ciencia-arte de la criptografía, a pesar de sus impresionantes avances. No existen recetas de cocina para resolver los problemas de seguridad, solamente el uso de ingeniería muy cuidadosa, y un continuo monitoreo del funcionamiento del sistema, con depuraciones y correcciones ya sea descubiertas por maleantes, o por académicos en búsqueda de mejores soluciones de seguridad.

> *Un sistema puede funcionar bien solamente cuando todos las partes interesadas (usuarios, administradores) están involucradas; tanto bien informadas de los problemas y soluciones de seguridad, como que tomen responsabilidad personal en el buen funcionamiento del sistema.*

Es necesario mantener informados a usuarios de las políticas de seguridad, y el uso adecuado de los sistemas de seguridad. El ejemplo más común es el del uso de passwords (cómo se deben elegir, con qué frecuencia se deben cambiar, etcétera), que es donde existe un gran número de violaciones a los sistemas, debido a una mala elección de passwords. Un punto importante aquí es el del control de usuarios: a quién se le da clave en el sistema, y por cuanto tiempo (cuándo se dan de baja claves).

Es necesario mantenerse continuamente actualizados de los problemas de seguridad nuevos que aparecen y sus soluciones; el software de seguridad debe ser actualizado constantemente. Por ejemplo, existen programas para protección de virus que permiten conectarse a través de Internet al fabricante y automáticamente bajar código con la información de los nuevos virus conocidos.

Políticas y procedimientos de seguridad

Tradicionalmente los temas de seguridad y criptografía habían estado confinados a las esferas militares y de alto gobierno. Por una lado, el valor de los recursos que se desea proteger en estos ámbitos es muy grande; ya sea en el costo de vidas humanas y recursos materiales que están en juego en una guerra, o en el riesgo que se incurre al filtrarse información que se maneja por altos funcionarios de gobierno. Además, en ámbitos de este tipo era usual la transmisión de información entre lugares distantes, lo que daba lugar a que personas no autorizadas intentaran obtener o incluso modificar la información. El mundo de las computadoras ha cambiado drásticamente en los últimos 25 años, de consistir en su mayoría de computadoras aisladas y administradas centralmente, a un mundo en el que millones de usuarios tienen sus computadoras interconectadas y funcionando bajo sistemas distribuidos. Antes la mayoría de los sistemas tenían un control centralizado de qué usuarios tenían acceso a los recursos, teniendo a los sistemas en cuartos cerrados bajo llave. Ahora es posible conectarse desde una computadora portátil en un cuarto, a otra en el otro lado del océano. En los 1990's una enorme cantidad de sistemas están conectados a Internet, desde todo el mundo. Viruses y gusanos pueden pasar de una computadora a otra. La situación es similar a la de un criminal que puede intentar el acceso ilícito a una casa buscando puertas y ventanas, pero complicado enormemente por al menos dos factores: la complejidad de los sistemas computacionales (en contraste con una casa), y el poder de cómputo que le permite al criminal el ataque a muchas computadoras en cuestión de minutos.

Es necesario entender los riesgos que existen, así como el valor de los recursos que se desean proteger. Las *políticas de seguridad* definen lo que la empresa espera acerca del uso apropiado de las computadoras y la red, y los *procedimientos de seguridad* intentan prevenir y responder a los ataques. Es decir, el plan para lidiar con la seguridad del sistema. El enfoque básico es el siguiente [10]:

1. Considera lo que quieres proteger.
2. Considera de qué lo quieres proteger.
3. Determina que tan posibles son los riesgos.
4. Implementa medidas que protejan tus recursos de manera eficiente en cuanto a costo.
5. Revisa el proceso continuamente, y mejora las cosas cada vez que son encontradas debilidades.

Un documento muy valioso es el *The Site Security Handbook* del Internet Task Force. Esta biblia del administrador de sistemas es una guía de como establecer políticas y procedimientos de seguridad para sistemas conectados a Internet.

Veamos algunos ejemplos de escenarios que ilustran el tipo de cuestiones con las cuales hay que lidiar:

- El administrador del sistema recibe una llamada avisándole que desde su sistema se ha distribuido un email a EU y Europa con una nota racista.
- Un usuario llama para reportar que no se puede conectar al sistema desde su casa el fin de semana. El encargado del sistema tampoco lo puede hacer. Después de rebootear el sistema en modo de usuario único, se da cuenta que el archivo de passwords ha desaparecido. El lunes se da cuenta que ha habido transferencia de archivos de su sistema a otro en una universidad. El martes se descubren copias de los archivos en una máquina de

esa universidad. Una semana después, descubre qué archivos de inicializacion han sido alterados.

- Se recibe una llamada avisando que alguien ha logrado meterse a una máquina del gobierno desde tu computadora. Se te pide una copia de tus archivos de las cuentas de usuarios para ayudar a descubrir al atacante. Una semana después, te dan una lista de tus máquinas que han sido penetradas.

- Un reportero te llama para decirte que ha habido un intruso en tu máquina. Tres días después te das cuenta que alguien sí se metió a tu máquina. El director tenía como password el nombre de su esposa.

Niveles de Seguridad

Hay que estar concientes y definir políticas en cuanto al nivel de seguridad que se quiere tener. Por ejemplo, originalmente UNIX tomaba el punto de vista de que el sistema se usa en un medio de relativa confianza (e.g. un medio académico), y relativo bajo valor de la información (e.g. no se manejan datos cuya pérdida implicaría millones de pesos de costo). Esto implica, por ejemplo, que una persona puede ver archivos de otra, a menos de que explícitamente lo cambie.

Relacionado a esto último, es común que conexiones remotas y correo no van encriptadas, por lo que puede haber problemas de privacidad e inclusive corrupción. Claro que uno puede elegir enviar un documento encriptándolo, pero esto sería como si en el sistema de correos todas las cartas fueran abiertas, y solamente al enviar información muy sensible se eligiera enviarla en un sobre. En primer lugar, al ver una carta cerrada, inmediatamente despertaría sospechas, en el mejor caso de simple curiosidad, y en el peor caso de un fraude o un crimen. En segundo lugar, el procedimiento de enviar una carta ensobretada sería más complicado. Éste es el caso en Internet hoy en día, donde para enviar un mensaje encriptado hay que quedar de acuerdo con el destinatario en una serie de parámetros que incluyen protocolos de encriptamiento y sus parámetros.

Por otro lado, es importante evaluar cuidadosamente el nivel de seguridad deseado. El problema es que mientras se implante un nivel mayor, el sistema se vuelve menos flexible, más difícil de usar y costoso. Estos últimos puntos implican que hay que tomar decisiones del nivel de seguridad que se quiere, e informar a la gente de esto, para que estén concientes de las implicaciones.

Por ejemplo, cada quien puede tomar decisiones personales de acuerdo al tipo de información que se quiere proteger. Para esto existe software para encriptar tanto archivos como comunicaciones a través de la red (PGP y ssh por ejemplo), de tal manera que se puedan transmitir y sólo sean legibles al destinatario aunque alguien más tuviera acceso a ellos.

> *Siempre hay que estar concientes que no existe tal cosa como un sistema 100% seguro.*

3 Herramientas Criptográficas

Bajamos ahora desde las consideraciones de alto nivel discutidas en la sección anterior, hasta las capas de más abajo, que proveen los fundamentos para la construcción. En esta sección describimos brevemente las herramientas básicas que la criptografía provee para poder construir un sistema de cómputo seguro, y qué tipo de garantías de seguridad son las que estas herramientas proveen. Algunas referencias excelentes son [17, 16, 21].

Bruce Schneier ilustra claramente la idea de la criptografía moderna [21]:

> *Si tomo una carta, la meto en una caja fuerte, y escondo la caja en algún lugar de Nueva York, y después te digo que leas la carta, eso no es seguridad. Es obscuridad. Por otro lado, si tomo una carta, la meto en una caja fuerte, y luego te doy la caja junto con sus especificaciones de diseño, y además te doy cientos de cajas idénticas a ésta con sus combinaciones para abrirlas, para que tú y los demás cerrajeros expertos del mundo puedan estudiar su mecanismo –y aún así no puedes abrir la caja y leer la carta– eso es seguridad.*

3.1 Conceptos básicos

La *criptografía* es el arte de la escritura secreta. La operación primordial de la criptografía es la de transformar un mensaje en un texto ininteligible para todos, excepto para el destinatario, que es el único que puede efectuar la transformación inversa y ver el mensaje original.

En la criptografía actual, estas transformaciones son efectuadas por computadoras, mediante programas, cuya descripción general es en forma de *algoritmos* (el programa es la implementación del algoritmo en un lenguaje de programación específico). Los mensajes consisten de secuencias de 0's y 1's, o bits, que representan letras del alfabeto de un texto. De esta forma, un mensaje M se transforma en un *criptograma* C mediante un algoritmo de *encriptamiento*, y de regreso a M mediante un algoritmo de *decriptamiento*. Podemos pensar en el mensaje como un número, escrito en binario, y en los algoritmos como funciones; el algoritmo de encriptamiento transforma el número correspondiente a M en otro número, correspondiente a C. La manera en la que un algoritmo logra que solamente el destinatario pueda descifrar el criptograma es utilizando *llaves*. Éstas son palabras o secuencias de caracteres que solamente los participantes intencionados poseen, y ellos no revelan a nadie más.

3.1.1 Dos métodos de encriptamiento sencillos

Se dice que Julio Cesar utilizó el siguiente método para la comunicación secreta de sus tropas. Cuando deseaba enviar un mensaje que consistía de una secuencia de letras, cambiaba cada una de las letras por la letra que le seguía en el alfabeto tres lugares después, considerando al alfabeto de forma cíclica (la "a" viene después de la "z"). La palabra "romano" era encriptada y enviada como “urpdqr”. El destinatario del mensaje, que conocía el método de encriptamiento usado, podía decriptar el mensaje utilizando la operación inversa. Esta estrategia de encriptamiento se puede generalizar a un corrimiento arbitrario de las letras del alfabeto, de tamaño k; es decir, cada letra se transforma en la que le sigue k posiciones después en el alfabeto. El destinatario debe conocer la *llave* para decriptar el mensaje, que es el número k.

Ahora se sabe que este método de encriptamiento no es seguro. Mas en general, considérese un método de encriptamiento que sustituya cada letra del alfabeto por alguna otra. Es decir, el método se puede describir mediante una tabla, que a cada letra le asigna una letra, de tal manera que no hay dos letras que se les asignen la misma letra. Un método de este tipo no es seguro, porque se le pueden aplicar ataques del tipo de análisis estadístico, entre otros. En un lenguaje como el español, si analizamos un texto cualquiera suficientemente grande, se puede ver que cada una de las letras aparece con cierta frecuencia, y que hay letras que aparecen mucho más veces que otras. Por lo tanto, teniendo acceso a suficiente texto encriptado, es posible contar cual es la letra que ocurre con más frecuencia, y suponer que ésta corresponde en la tabla a la letra de mayor ocurrencia en español. Otro ejemplo de este tipo de criptoanálisis es identificar las parejas de letras que ocurren con más frecuencia.

Esta discusión ilustra como algoritmos de encriptamiento del tipo de César dejan filtrar información acerca del mensaje original a un posible atacante. Un sistema de encriptamiento óptimo, no debería dejar filtrar ninguna información acerca del mensaje original (por ejemplo, el primer bit del mensaje, la suma de sus bits, etcétera) a alguien que no conoce la llave de decriptamiento.

Veamos un sistema de encriptamiento perfecto. En el sistema de *Vernam*, o *one-time-pad*, las dos personas que quieren intercambiar información secreta, Alicia y Benito, primero se juntan y quedan de acuerdo en una secuencia de caracteres. Por ejemplo, un poema de Neruda. Después, cada quien se va a su casa, y mantienen en secreto el texto elegido. Cuando una de ellas, digamos Alicia, quiere enviarle un mensaje a Benito, toma cada una de las letras y una por una se las va sumando a las letras del poema: la letra del poema indica el corrimiento que hace en la correspondiente letra del mensaje. Este sistema es perfectamente seguro, porque un atacante no puede obtener ninguna información acerca del mensaje original; dado un criptograma, es posible decriptarlo al mensaje que sea, eligiendo un texto secreto adecuado. Nótese que el texto elegido no necesariamente debe estar formado de palabras, puede ser una secuencia de letras cualesquiera.

El problema es que la seguridad perfecta de este sistema depende de que Alicia y Benito sólo intercambien un mensaje, y sólo de longitud menor o igual al texto secreto elegido. Este texto representa la llave secreta. El reto de la criptografía moderna es el diseñar métodos de encriptamiento que permitan enviar un buen número de mensajes con una llave relativamente pequeña.

3.1.2 Algoritmos criptográficos

Existen tres tipos principales de funciones criptográficas: de llave privada, de llave pública y de huella digital. En esta sección describimos qué es cada una de ellas, y dejamos para después las operaciones de seguridad que se pueden construir a partir de ellas.

Criptografía Privada

En la *criptografía privada* –la usada desde la antigüedad– si Alicia y Benito desean comunicarse en forma privada, entonces comparten una llave secreta K_{AB}, es decir, una secuencia de caracteres (que la computadora representa como una secuencia de bits) que sólo ellos dos conocen. Cuando Alicia desea enviarle a Benito un mensaje M, ejecuta en su computadora (la de Alicia) un algoritmo de encriptamiento, E, dándole como entrada M y K_{AB}, y obteniendo como salida un critpograma C:

$$C := E(M, K_{AB}).$$

Alicia envía a Benito C. Suponiendo que estaban de acuerdo en K_{AB}, y en el algoritmo de encriptamiento, entonces Benito, al recibir C, puede efectuar la operación inversa en su computadora, y obtener el mensaje original M:

$$E := D(C, K_{AB}).$$

De la misma forma, Benito puede enviarle mensajes privados a Alicia: utilizando el algoritmo E y la llave K_{AB}. Alicia y Benito deben guardar la llave con mucho cuidado, ya que cualquiera que logre averiguarla podría descifrar sus mensajes.

El exponente moderno más prominente de sistema de criptografía privada es DES, desde su introducción en 1975, y que estuvo basado en uno desarrollado por IBM a principios de los 70's, llamado Lucifer. El DES ha sido muy popular y funcionó bien, siendo el estandard de facto por mas de 20 años, debido a su seguridad y eficiencia, habiendo implementaciones en hardware que encriptan millones de mensajes por segundo. A principios de los 90's, debido al impresionante avance en el poder de las computadoras, empezó a haber preocupación de la seguridad de DES, y empezaron a surgir otras alternativas de algortimos, algunas de las más populares, extensiones de DES mismo, que sigue siendo muy usado. DES encripta mensajes de 64 bits en criptogramas de 64 bits, usando una llave que es cualquier secuencia (a excepción de algunas pocas que son poco seguras) de 56 bits.

Criptografía Pública

En la *AFIPS National Computer Conference* de 1976 Diffie y Hellman presentaron una maravillosa idea que sorprendió a muchos: la posibilidad de usar una llave para encriptar, y otra distinta para decriptar, que resulta tener una variedad de aplicaciones interesantes. En la *criptografía pública*, no es necesario generar una llave para cada pareja de participantes que deseen comunicarse privadamente. Alicia genera una pareja de llaves, $K\,\mathrm{priv}_A$ y $K\,\mathrm{pub}_A$. La llave $K\,\mathrm{priv}_A$ la mantiene en secreto, mientras que la llave $K\,\mathrm{pub}_A$ la publica en algún directorio. Cualquier persona que desee comunicarse con Alicia, busca en el directorio público su llave $K\,\mathrm{pub}_A$, y ejecuta un algoritmo de encriptamiento de llave pública:

$$C := E(M, K\,\mathrm{pub}_A),$$

para enviarle C a Alicia. Pero sólo Alicia conoce $K\,\mathrm{priv}_A$ y puede descifrar el mensaje:

$$M := D(C, K\,\mathrm{priv}_A).$$

El precio a pagar es que los algoritmos de criptografía pública que se conocen, son mucho más lentos que los de llave privada (en el orden de 1000 veces mas lento que DES). Tanto así que usualmente no son utilizados para encriptar datos. Un uso típico de los algoritmos de llave pública es para encriptar llaves privadas. Una vez intercambiadas las llaves privadas, se usan para encriptar los datos mediante criptografía privada.

El sistema de criptografía pública más popular es RSA, usado en muchos sistemas comerciales, por navegadores web (por ejemplo, Netscape), privacidad y autenticidad de correo electrónico, y sistemas de pago por tarjetas de crédito. Publicado originalmente en el número de agosto de 1977 en *Scientific American*, por Ron Rivest, Adi Shamir y Len Adleman, RSA se ha estudiado escrupulosamente, y se han encontrado una variedad de ataques interesantes [3]. Sin embargo, ninguno de estos es devastador, y más bien ilustran los peligros de usar RSA de manera inadecuada; el sistema sigue resistiendo la prueba de seguridad que sólo el tiempo puede dar. No hay muchos otros sistemas de llave pública que a la vez sean eficientes y se consideren seguros, aparte de los otros dos conocidos como ElGamal y Rabin.

La idea de RSA es aprovechar el hecho de que se cree que factorizar números enteros grandes es muy difícil. Se eligen dos primos grandes (digamos del orden de 100 dígitos), p y q, y se toma el producto $n = pq$. Luego se elige la llave de encriptamiento, e, tal que e y $(p-1)(q-1)$ son primos relativos. La llave de decriptamiento es el entero d tal que $ed \equiv 1 \pmod{(p-1)(q-1)}$.

Para encriptar un mensaje M (de menos de 200 dígitos, en el caso de que n así lo sea), se evalúa

$$C = M^e \pmod{n},$$

mientras que, usando propiedades relativamente básicas de teoría de números, se puede ver que el criptograma C es decriptado evaluando

$$M = C^d \pmod{n}.$$

La seguridad de RSA se basa en que se cree que violar el sistema es tan difícil como factorizar n, y que para hacer esto no parece haber métodos sustancialmente mejores que probar dividir n por todos los números menores a él, lo cual, para valores de n suficientemente grandes, toma demasiado tiempo incluso con las computadoras más poderosas.

Funciones de Huella Digital

El tercer tipo de función criptográfica es el llamado *huella digital* (en inglés *hash* o *message digest*). Un algoritmo de huella digital, H, transforma un mensaje largo, M, en uno más corto, $H(M)$, que es el huella digital. En contraste con los otros dos tipos de funciones criptográficas, este tipo de función no utiliza llaves. La manera en que se hace la transformación es unidireccional: cualquiera puede utilizar el algoritmo para hacer huella digital un mensaje, y esto toma poco tiempo. Es claro que habrá muchos mensajes que resultan en el mismo huella digital, ya que, como los mensajes son de mayor longitud que las huellas digitales, hay más mensajes que huellas digitales. Una función huella digital tiene la interesante propiedad de que dada una huella digital c es imposible, o más precisamente, toma demasiado tiempo, buscar un mensaje m tal que $H(m) = c$. También, la propiedad de que es computacionalmente imposible hallar dos mensajes que resulten en la misma huella digital. La palabra "imposible" es en el sentido que se explica en la siguiente sección: esencialmente, la única manera es probando todas las posibilidades, las cuales son demasiadas, si se elige la longitud de la huella digital suficientemente grande. Una elección típica es 128 bits. De esta manera, la huella digital de un mensaje funciona como una especie de huella digital (como si fuera la del dedo) del mensaje original. Algunas de las funciones de huella digital más populares son ambas de 1992: la MD5 de 128 bits, propuesta por Ronald Rivest (usada por ejemplo en Netscape) y la SHA de 160 bits, de NIST (*National Institute of Standards and Technology*) y NSA (*National Security Agency*).

Generadores Pseudo-aleatorios

Quizás menos conocido en la construcción de sistemas criptográficos es la importancia de generar una secuencia de bits aleatorios. Además de para generar las llaves del sistema, con frecuencia es útil disponer de números aleatorios. En ambos casos, la seguridad del sistema se puede comprometer si los números elegidos no son realmente aleatorios. Considérese el caso extremo en el cual para generar la llave se usa un generador que siempre produce números pares. Un maleante puede utilizar la información de que el bit menos significativo de la llave es 0 para violar el sistema.

Los algoritmos generadores de números aleatorios funcionan a partir de una *semilla* que es un número que se les da como entrada, y como salida producen una secuencia de bits que pretenden ser aleatorios. La dificultad consiste en generar una secuencia lo más larga posible, a partir de semillas pequeñas. En realidad es imposible generar números perfectamente aleatorios, ya que la computadora misma es una máquina determinista. Para una semilla dada, cualquier algoritmo producirá una secuencia de bits, que si es suficientemente larga, necesariamente se vuelve periódica, y claramente una secuencia periódica no es aleatoria. De hecho, los generadores aleatorios que vienen incluidos en sistemas operativos están muy lejos de ser perfectamente aleatorios; para aplicaciones como juegos o actividades del sistema operativo funcionan suficientemente bien.

Para criptografía es necesario garantizar que el generador de bits aleatorios sea lo mejor posible, en vista de que perfectamente aleatorio es imposible de lograr. La seguridad del generador depende de que sea sumamente difícil predecir la secuencia de bits producida, por un intruso que no conoce la semilla que la generó. En este sentido es que se dice que se trata de un generador *pseudo-aleatorio seguro*. Existen muchas propuestas de estos generadores, como RC4, SEAL, BBS's basado en residuos cuadráticos, otros basados en RSA, y en la dificultad del logaritmo discreto.

3.1.3 Seguridad

La criptografía se ha usado desde los egipcios, hebreos y romanos. Por miles de años ha sido un arte, en el cual era un reto inventar cada vez métodos más ingeniosos, que a su vez eran violados por cada vez más ingeniosos maleantes. La criptografía finalmente está pasando de ser un arte a ser una ciencia. Aunque aún quedan problemas fundamentales por resolver, y muy importantes ya que sigue sin conocerse un sistema criptográfico demostrablemente inviolable, ya existen maneras precisas de definir la seguridad de un sistema, y métodos matemáticos poderosos que dan evidencia de que un criptosistema satisface ciertos requerimientos de seguridad. De hecho, ya existe una gran variedad de diseños teóricos de protocolos de seguridad basados en criptografía; el reto actual es encontrar implementaciones eficientes de estos métodos.

Seguridad Computacional

Como vimos antes, en un sistema criptográfico como el de Vernam, es posible lograr una seguridad absoluta, pero a costa de una llave secreta del mismo tamaño que todos los mensajes que se desean enviar. Un sistema como este, que no es vulnerable a un atacante inclusive con acceso a tiempo y recursos ilimitados se le llama *incondicionalmente seguro*. Desafortunadamente, el único sistema que se conoce incondicionalmente seguro es el de Vernam. En la práctica deseamos utilizar una llave secreta relativamente pequeña, y con ella encriptar una buena cantidad de mensajes. En la criptografía clásica, se pretendía que un sistema de este tipo fuera seguro, tratando de mantener secreto el método de encriptamiento. En cambio, en la criptografía moderna, se pasa al concepto de *seguridad computacional*.

El diseño del sistema criptográfico pretende que la única forma de que algún intruso que no conoce la llave secreta K_{AB} (o $K\,\mathrm{priv}_A$ en el caso de un sistema de llave pública) pueda descifrar el criptograma C es probando todos sus posibles valores. Es decir, no es imposible descifrar un criptograma; simplemente tarda demasiado tiempo. En funciones huella digital, no es que no se pueda encontrar un mensaje que produzca una huella digital dada, el problema es que toma demasiado tiempo buscar uno. De manera similar, la seguridad de un generador pseudo-aleatorio no está basada en que sea imposible predecir la secuencia de bits generados; siempre se puede probar todas las semillas posibles, hasta encontrar la que produjo la secuencia.

A diferencia de la criptografía clásica, en la criptografía moderna la seguridad de un sistema no está basada en mantener los algoritmos en secreto, sino en el tiempo que le tomaría a un intruso descifrar el criptograma, sin conocer la llave. El tiempo que toma un algoritmo, es decir, su *complejidad de tiempo*, es objeto de estudio de la *teoría de la complejidad*, un área de las ciencias de la computación de considerable profundidad matemática. Esta rama de la computación pretende encontrar evidencias matemáticas concretas de por qué es imposible que una computadora pueda descifrar un criptograma de otra manera más que, esencialmente, probando todas las posibles llaves. Algunas referencias recomendadas son [13, 20].

Por lo tanto, nos interesa identificar problemas *difíciles computacionalmente*. Es decir, identificar un problema para el cual podemos demostrar que cualquier algoritmo le toma demasiado tiempo en resolverlo. La situación actualmente es un tanto curiosa: hasta la fecha no se ha logrado construir un sistema criptográfico y demostrar que el romperlo es equivalente a resolver algún problema computacionalmente difícil. Lo que sí se ha logrado hacer, es obtener evidencia matemática fuerte de que ciertos problemas probablemente sean difíciles, y probablemente sean equivalentes a romper el sistema. Por lo tanto, la prueba final de la seguridad de un sistema sigue siendo empírica: que mucha gente intente romperlo y no se logre. La gran ventaja de la criptografía moderna es que cualquiera puede estudiar, o intentar romper los algoritmos utilizados, ya que son públicos.

Vale la pena hacer un breve comentario para el que ha oído hablar de los problemas NP-completos. Uno podría pensar que un problema NP-completo sería un buen candidato para ser utilizado en un criptosistema. Resulta que esto no necesariamente es cierto. El problema es que para probar que un problema es NP-completo, es necesario solamente probar que es difícil en *el peor caso*. Para que un problema sea útil en criptografía no es suficiente que contenga algunas instancias difíciles; es necesario que la mayoría de sus instancias lo sean. Es decir, hay que trabajar con *complejidad promedio*.

La Importancia de la Llave Secreta

Se sigue de esta discusión que la elección de la llave es de importancia fundamental. En primer lugar, se debe elegir en forma lo más aleatoria posible. La primera forma en que comúnmente los maleantes intentan violar un sistema es probando usar como llave todas las palabras del diccionario, así como variante y otras que pudiera haber utilizado el usuario, como su nombre, o su fecha de nacimiento.

En segundo lugar, la longitud de la llave es un parámetro crítico en la seguridad del sistema. Si la llave consiste de k bits, entonces habrá que probar 2^k posibilidades en el peor caso (o $2^k/2 = 2^{k-1}$ en promedio). Para elegir k hay que tomar el número de llaves por unidad de tiempo que una computadora puede probar hoy en día, y calcular cuánto tiempo le tomaría hacerlo. Elegimos la k de acuerdo al nivel de seguridad requerido por la aplicación específica. Si la información es importante solamente durante una semana, es suficiente elegir una k que haga que cualquier computadora le tome, digamos, un mes en buscar todas las posibilidades. Es por esto que se cree que la NSA tiene capacidad de romper muchos de los criptosistemas usados, basándose en su gigantesca capacidad de adquirir hardware. Para dar una idea más concreta de las cifras involucradas, consideremos una llave de 128 bits (así lo especifica el algoritmo de llave privada IDEA, por ejemplo). Esto nos da 2^{128} (3.4×10^{38}) llaves posibles. Si existiera una computadora que pudiera probar 100 millones de llaves distintas en un segundo, y si contáramos con 100 millones de estas computadoras para poder hacer una búsqueda en paralelo, tomaría 10^{13} años probar todas las posibles llaves. Lo cual es aproximadamente 1000 veces más que la edad del universo, que actualmente se estima en 1.2×10^{10} años.

Entonces, ¿por que no simplemente elegimos una longitud de llave muy grande? Por ejemplo, 1000 bits. Por dos tipos de razones, principalmente. Por un lado, mientras más larga sea la llave más difícil se vuelve guardarla en secreto. Por otro lado, mientras más larga sea la llave, los algoritmos de encriptamiento y decriptamiento de vuelven más lentos.

Es importante hacer notar que un ataque de *fuerza bruta* como éste, en el que simplemente se prueban todas las posibles llaves, no es necesariamente el mejor para un algoritmo dado. Además, la velocidad de búsqueda de las computadoras va aumentando con los avances de las tecnologías tanto de hardware como de software. Es por esto que muchos sistemas criptográficos permiten variar la longitud de la llave. Se diseñan de tal forma que el aumentarla en un bit, se incrementa sólo un poco el tiempo de ejecución de los algoritmos para aquellos que conocen la llave secreta, pero para el maleante que no la conoce, el tiempo de búsqueda se incrementa al doble: $2^{k+1} = 2 \cdot 2^k$.

Resumiendo, es importante que el usuario elija sus llaves de manera adecuada, y que las proteja para que no sean averiguadas por intrusos. Más aun, es importante cambiar de llaves de vez en cuando. También, que la longitud de la llave utilizada por el sistema vaya de acuerdo a la importancia de la información resguardada, y que se cuide de incrementar la longitud con el pasar de los años.

3.2 Métodos de Seguridad a partir de criptografía

Utilizando las principales funciones criptográficas – de llave privada, de llave pública, huella digital y generadores pseudo-aleatorios – es posible crear una variedad de operaciones básicas de seguridad, algunas de ellas ingeniosas y sorprendentes. Éstas constituyen las herramientas básicas sobre las cuales se construye un sistema de cómputo seguro. No describiremos cuál de las funciones es usada para cada una de estas operaciones, ni cómo, ni en caso de poderse usar más de una, cómo elegir cuál. Simplemente, a continuación describimos en forma muy breve, algunas de las operaciones más importantes, y algunos ejemplos de implementaciones. Algunas referencias excelentes al respecto son [14, 21].

1. Transmisión sobre un canal inseguro.

   Usualmente es imposible evitar que un maleante observe transmisiones de información, ya sea que el medio fuera líneas de teléfono, redes locales, ondas electromagnéticas. Inclusive fibras ópticas, que en el pasado se pensaba eran imposibles de observar sin destruir la comunicación. El uso clásico de la criptografía es el de enviar la información encriptada, de tal manera que un intruso no pueda descifrarla, a pesar de poder verla en el medio de comunicación.

2. Almacenamiento de información en un medio inseguro.

   Utilizando criptografía es posible almacenar información en cinta o disco, de tal manera que a pesar de que maleantes logren llegar a ella, no puedan descifrarla. Sólo el dueño de la llave secreta lo puede hacer. De hecho, esto tiene la gran conveniencia de que es igual a donde esté físicamente almacenada la información; las cintas no necesitan estar en una caja fuerte para evitar lecturas no autorizadas (esto evita modificar la información, más no su destrucción).

3. Autenticación.

   Cuando pensamos en autenticar a una persona, consideramos asegurarnos de su identidad a partir de algo que la persona *tiene,* por ejemplo, una llave, una tarjeta de identidad, o algun otro dispositivo físico. O bien, nos podemos tratar de asegurar de su identidad a partir de algo que la persona *sabe,* como un número secreto, un datos personal, etc. Finalmente, consideramos lo que la persona *es,* utilizando sus huellas digitales, voz, imágenes de su retina o de su cara.

   Mediante las funciones criptográficas básicas es posible lograr que Alicia pueda probar a Benito que posee un cierto conocimiento, sin revelar cuál es éste. La criptografía permite que Benito esté realmente seguro de que es con Alicia con la que está hablando. Existe una gran variedad de maneras de lograr esto (y cada vez se inventan nuevas), para distintas aplicaciones y con distintas garantías de seguridad [19].

4. Verificación de integridad.

   Es posible añadir a un mensaje un certificado que permite detectar si el mensaje fue cambiado. Esto es similar al objetivo de los códigos de corrección de errores, pero funciona contra intrusos maliciosos, y no solamente contra errores aleatorios del medio de comunicación.

5. Firmas digitales.

   Éstas tienen la misma función que las firmas a mano. Sirven para garantizar que la persona que dice haber enviado un mensaje M, realmente lo envió. Una firma digital de Benito sobre M garantiza que Benito fue el creador de M.

   Existen también firmas digitales con otras propiedades, algunas que fortalezen su seguridad y otras que logran propiedades distintas. Por ejemplo, las *firmas fail-stop,* permiten detectar falsificaciones. Veremos que en las aplicaciones de comercio electrónico resultan útiles las *firmas ciegas*, que permiten firmar un documento sin averiguar su contenido. Pero concentrémonos en las firmas digitales simples.

   Un método de firma digital muy popular es el basado en el criptosistema de llave pública RSA. Otro es el estandar DSA emitido por NIST en 1994. Veamos cómo se puede usar RSA para firmar un documento M. Simplemente Alicia encripta M utilizando su llave privada:

   $$C = M^e \pmod{n},$$

y la firma es la pareja (M, C). La única persona que pudo haber producido C a partir de M es Alicia, ya que sólo ella conoce la llave secreta que encripta M en C. Más aún, Alicia no tiene manera de modificar un mensaje firmado por ella misma, ya que no puede encontrar otro mensaje M' que al encriptarlo con su llave secreta dé el mismo valor C. Por otro lado, utilizando la llave pública correspondiente, cualquier persona puede verificar que la firma es válida, evaluando

$$C^d \pmod n.$$

y verificando que el valor obtenido, sea igual al documento original, M. Es decir, la firma esta asociada a M solamente y no se puede usar para firmar otros documentos. Alicia no puede negar haber firmado el documento, ya que sólo el poseedor de la correspondiente llave secreta –Alicia– pudo haber generado la firma.

En la práctica es necesario tomar algunas precauciones. Por ejemplo, se puede añadir al mensaje firmado la fecha y hora o un número de serie, para garantizar que el documento firmado sea único. También asuntos de índole práctico, en el sentido que el firmar y verificar la firma sea un proceso rápido. Para esto usualmente no se firma el documento original M, sino una huella digital $H(M)$, que es mucho más corta.

6. Palabras de accesos (passwords).

   Un problema común de autenticación, es cuando un usuario teclea su palabra secreta de acceso en una terminal, y la computadora tiene que corroborar si es la palabra correcta. Es posible lograr esto sin que la computadora sepa cuál es esa palabra, y sin que la tenga almacenada en ningún lado.

7. Huellas digitales, firmas y ensobretamiento.

   Las huellas digitales son útiles para garantizar que un programa no ha sido modificado inadvertidamente. En lugar de almacenar en un lugar seguro todo el programa, se almacena en lugar seguro nada más la huella digital del programa, que no es más que un huella digital de su código, y por lo tanto mucho más pequeña.

   La operación de firmar digitalmente un documento grande puede ser muy costosa en términos de tiempo. Es posible firmar un huella digital del documento, en forma más veloz, ya que los algoritmos para hacer huella digital son muy veloces.

   Cuando se envía un documento por Internet, es deseable *ensobretarlo*. Esto se logra primero firmando el mensaje digitalmente y luego encriptándolo. La firma garantiza la autenticidad del mensaje, mientras que la encripción garantiza su privacidad. Se podría pensar en invertir el orden de las operaciones, lo cual sería como enviar una carta en un sobre, y en lugar de firmar la carta, firmar el sobre. Pero esto no es lo conveniente, ya que puede ser menos seguro, en especial si se usa RSA, además de que introduce dificultades legales.

8. Distribución segura de secretos.

   Si Alicia tiene n hermanos, puede partir un secreto (por ejemplo, la llave digital de su caja fuerte) en n pedazos, y repartir un pedazo a cada hermano, de tal manera que los pedazos no den ninguna información acerca del secreto de Alicia. Más aún, Alicia puede elegir un parámetro de seguridad, k, tal que solamente si al menos k de ellos se ponen de acuerdo y juntan sus pedazos, entonces pueden recuperar el secreto. Pero cualquier intento por menos de k hermanos de juntar sus pedazos no les da ninguna información.

9. Compromiso de un bit.

   Alicia quiere elegir el valor de un bit, y avisarle a Benito que ya lo eligió, pero revelarle el valor más tarde. Benito quiere estar seguro de que más tarde Alicia no cambie el valor que eligió al principio. Alicia quiere estar segura que Benito no averigua su bit antes de tiempo. Es como si Alicia quisiera meter su bit en un sobre, y dárselo a Benito, de manera que solamente más tarde, en presencia de ambos se abre el sobre. Existen diversas maneras de implementar esto, sin necesidad de un sobre de papel y de juntarse en un sólo lugar a abrirlo. La siguiente es una propuesta de Moni Naor basada en generadores pseudo-aleatorios.

   (a) Benito genera una cadena aleatoria de bits R_B y se la envía a Alicia.
   (b) Alicia genera una semilla aleatoria y se la da al generador pseudo-aleatorio. Para cada bit de R_B, le envía a Benito:
       i. la salida del generador pseudo-aleatorio en caso de que el bit de R_B sea 0, o
       ii. el OR-exclusivo de su bit compromiso y el bit de la salida del generador, si el bit de Benito en R_B es 1.
   (c) Al momento de descubrir su bit, Alicia le envía a Benito la semilla elegida.
   (d) Benito reproduce la secuencia pseudo-aleatoria con esa semilla, y verifica que Alicia había ejecutado el protocolo correctamente.

10. Volados justos.

    Otro simpático e importante problema que se ha estudiado en criptología es el de cómo echar un volado sin tener una moneda. Se tienen dos o más participantes, cada uno en un lugar diferente, y solamente pudiendo comunicarse mediante mensajes a través de una red de computadoras, y quieren generar un bit aleatorio de manera que ninguno pueda influenciar el resultado. En este sentido el protocolo debe ser "justo", la probabilidad de que el bit sea 0 deber ser lo más cercana posible a $1/2$, independientemente de los esfuerzos tramposos que pudieran tratar de hacer los participantes. Una solución clásica utiliza el problema del compromiso de un bit:

    (a) Alicia elige un bit aleatorio, y le envía el compromiso correspondiente a Benito.
    (b) Benito trata de adivinar el bit, enviándole el valor sugerido a Alicia.
    (c) Alicia revela el valor de su compromiso. El resultado del volado es 0 si la adivinanza de Benito fue correcta, de otro modo es 1.

4 Sistemas

Para llegar a utilizar métodos criptográficos eficientemente, cualquiera que fuese la aplicación, es necesario construir una infraestructura y un sistema general adecuado. Por un lado, un sistema involucra aspectos desde redes y comunicación, pasando por algoritmos criptográficos entre dos participantes, protocolos basados en éstos que pueden involucrar a dos o más participantes, hasta sistemas de bases de datos, interfases, web, e inclusive estándares y convenciones de cómo se debe usar el sistema, incluidas consideraciones éticas y legales. Por otro lado, las aplicaciones pueden abarcar diversos intereses, como bancarios y financieros, sistemas de reservaciones, médicos, militares, o simplemente de correo electrónico seguro y confiable entre usuarios de Internet.

A continuación se discuten tres sistemas. El primero es del tipo *facilidades generales:* "infraestructuras de llave pública", que son útiles para construir otros sistemas que involucran seguridad. Otro ejemplo de este tipo, podría ser un sistema de servidores de tiempo, para accesar la hora y poner sellos de tiempo a documentos. El segundo, "comercio electrónico", es del tipo *servicios*, donde el sistema tiene que ver con una aplicación específica, como podría ser también, un sistema médico, uno de declaración de impuestos, uno de compra-venta de acciones, o uno de votaciones electrónicas. El tercer sistema discutido es web, que es un sistema de *uso general*, pero de más alto nivel, ya que utiliza facilidades como las de infraestructuras de llave pública. Quizás otro ejemplo de este tipo podría ser un sistema de telefonía celular. Cada uno de los tres sistemas considerados ilustra la manera en que surgen nuevos conceptos y problemas de seguridad, y como con frecuencia es necesario inventar nuevas técnicas para resolverlos.

4.1 Infraestructuras de Llave Pública

Quizás el problema más importante para lograr construir sistemas seguros y eficientes que se usen a nivel global, es el de tener una infraestructura para la administración y distribución de llaves públicas. Una especie de directorio amarillo donde buscar la llave pública de una persona, pero mucho más sofisticado. Ha habido mucha actividad de investigación relacionada en los últimos años, y diversas propuestas de como lograrlo, para distintos escenarios de aplicación, con algunas compañías ya dando servicios relacionados, e iniciativas a nivel nacional en distintos países (como el Programa de Infraestructura de Llave Pública de NIST en EUA, y el de Europa). Algunos de estos sistemas y compañías son, PGP, PEM, Entrust, Verisign, SDSI de MIT, y buenas referencias relacionadas [11, 9].

Recordemos que en un criptosistema de llave pública un usuario, digamos Alicia, genera una pareja de llaves privada y pública, $K\,\mathrm{priv}_A$ y $K\,\mathrm{pub}_A$, guardando en secreto la primera, que le sirve para encriptar o firmar mensajes, y anunciando públicamente la segunda, que sirve para enviarle mensajes encriptados a Alicia, o para verificar documentos firmados por ella.

La firma de Alicia sobre un documento M en realidad consiste de M, de su encriptamiento $E(M, K\,\mathrm{priv}_A)$, y de la llave $K\,\mathrm{pub}_A$. El procedimiento de verificación por parte de Benito consiste en leer M, tomar $K\,\mathrm{pub}_A$, y decriptar $M' := D(C, K\,\mathrm{priv}_A)$, para compararlo con M, esperando sea $M = M'$. En el caso de que no lo sea, rechaza la firma como inválida. En el caso de que si sea $M = M'$, Benito sabe que

La llave utilizada para firmar M es la llave privada correspondiente a $K\,\mathrm{pub}_A$.

Es decir, que la única persona que pudo haber firmado el documento, es aquella persona que posee $K\,\mathrm{priv}_A$. Pero queda el problema de cómo sabe Bentito que esa persona es Alicia. Esto se resuelve en comunidades pequeñas, haciendo que Alicia y Benito se junten, verifiquen su identidad (visualmente, con una credencial), e intercambien llaves públicas. El problema de comunidades grandes es cómo lograr que exista esta asociación:

$$K\,\mathrm{pub}_A \leftrightarrow \text{ Alicia}$$

de manera confiable, flexible y eficiente, que es el núcleo de una infraestructura de llave pública.

Una solución está en la noción de *certificado*. Un documento que asevera la asociación entre un usuario, Alicia, y su llave pública, y que está firmado por una *autoridad certificadora*, AC. El usuario Benito que recibe una firma de Alicia, primero obtiene un certificado para la llave pública de Alicia, $K\,\mathrm{pub}_A$, firmado por alguna AC, y luego verifica el documento firmado con $K\,\mathrm{pub}_A$. Un certificado usualmente contiene información adicional, como el tiempo de validez de la llave pública, un número identificador único del certificado, el nombre del algoritmo criptográfico de la llave pública, etcétera. Existe un estandar, X.509 de ISO, que especifica la forma y uso de los certificados y demás componentes de la infraestructura, que veremos a continuación.

Una autoridad certificadora AC es una entidad de confianza, que emite certificados. Asegura que la llave $K\,\mathrm{pub}_A$ efectivamente es de Alicia, mediante una firma usando una llave secreta, $K\,\mathrm{priv}_{AC}$. En realidad, Alicia podría tener varias llaves: una asociada a su empleo, bajo un cargo específico; otra personal; otra de su club deportivo, etcétera. Por lo tanto, la AC se encarga de asociar un identificador único a cada participante con su llave pública, y entonces, Alicia podría tener varios de estos identificadores. El procedimiento puede ser que Alicia se presenta físicamente en una oficina de la AC, con varias identificaciones oficiales, se le toman sus datos y firma, una fotografía, etcétera, y ahí mismo se genera sus llaves.

Un componente importante de la infraestructura es el *directorio*, que tiene almacenados los certificados. Cuando Benito se quiere comunicar con Alicia, primero obtiene su certificado del directorio, y luego verifica su autenticidad. El directorio mismo debe satisfacer requerimientos de seguridad. Y esto se puede lograr siempre y cuando Benito y Alicia trabajen bajo la misma AC; es decir, Benito debe conocer la llave pública de la AC, $K\,\mathrm{pub}_{AC}$.

Por otro lado, es importante tener en cuenta que es responsabilidad de Alicia proteger su llave privada, $K\,\mathrm{priv}_A$, ya que si un maleante la logra averiguar, entonces se puede hacer pasar por Alicia; desde el punto de vista del sistema, aquel que sabe $K\,\mathrm{priv}_A$ *es* Alicia. Ahora bien, mientras más importante sea el uso que se le pretende dar a $K\,\mathrm{priv}_A$, ésta deberá ser más grande, y definitivamente no es algo que se pueda uno aprender de memoria, por lo que es necesario tener provisiones para almacenar la llave encriptada o en un lugar seguro.

Más aún, el sistema entero depende de que la llave privada de la AC sea segura; que se mantenga protegida y que sea suficientemente grande para evitar cualquier ataque.

Finalmente, es necesario otro componente en el sistema, para el caso en que Alicia pierde su llave privada, o sospecha que se la han robado, o simplemente para el caso de que Alicia deja su trabajo y ya no está autorizada a usar la llave secreta para firmar documentos. En la *lista de revocación de certificados*, LRC, se almacenan los certificados inválidos, no expirados.

Ahora bien, es necesario establecer diversas políticas de uso de la infraestructura. En particular, una AC define claramente su política de emisión de certificados. Debe especificar, tamaños de llaves, software utilizado, a quién se le emite un certificado, requerimientos de autenticación de las personas, frecuencia de emisión de la LRC, tiempo de validez de certificados, controles de seguridad en lugares de registro, generación de llaves, etcétera.

Si se quiere que el sistema sea flexible, robusto y expandible, será conveniente tener más de una autoridad certificadora. Distintas organizaciones necesitarán tener sus propias AC, definiendo sus políticas de emisión de certificados de acuerdo a sus necesidades. Estas organizaciones pueden ser gubernamentales, privadas, públicas, y de distintos países, por lo que es clara la imposibilidad de tener una sola AC, o una coordinación estrecha entre ellas. Los requerimientos de seguridad de un club deportivo que emite certificados serán muy diferentes a los de una agencia gubernamental. Pero para que el sistema pueda funcionar a nivel global, es necesario definir algunas políticas generales básicas, y una *arquitectura* de la infraestructura. En particular, surge la pregunta de quién certifica la llave de una AC.

Por ejemplo, el sistema de correo electrónico seguro PGP diseñado por Philip Zimmermann [24] supone que se trata de un sistema que los usuarios van adoptando informalmente, y sin autoridades certificadoras. Los usuarios se certifican unos a otros, y cada quien decide en quien confiar.

Una propuesta que se ha manejado para una infraestructura a nivel nacional, dentro del esquema de X.509, es tener una jerarquía de AC's en forma de árbol. La CA de más arriba tiene una llave secreta de muy alta seguridad y tamaño. Esta CA se encarga de certificar y establecer políticas de emisión de certificados a CA's en el siguiente nivel. Éstas a su vez certifican a otras, y así sucesivamente, hasta las CA's que certifican individuos. Esta arquitectura permite tener líneas claras de autoridad, navegación eficiente de la estructura, y es relativamente flexible y extendible.

Una propuesta alternativa interesante y original es SDSI, encabezada por Ron Rivest y Butler Lampson de MIT, y la relacionada SPKI, que ya se han unido. Esta propuesta intenta disminuir complejidades del esquema anterior, evitando problemas de espacios de nombres globales y sobrespecificaciones. El sistema se centra en las llaves públicas como entidades *principales*. No existen "individuos" sino solamente llaves públicas. Un principal se representa mediante:

(Principal:
(Public-Key:
(Algorithm: RSA-with-SHA1)
(N: =Gt802Tbz9HKm067=)
(E: #11)))

Y algunas de las ideas es no tener un jerarquía global, que cada principal pueda certificar, y tener espacios de nombres locales.

4.2 Comercio Electrónico

Existe ya la tecnología de criptografía (principalmente criptosistemas y protocolos) para diseñar sistemas de comercio electrónico más eficientes y seguros, y con mejores propiedades de anonimato que los tradicionales. Ya existen prototipos de *carteras electrónicas*, cuya combinación de hardware y software permite a un usuario mantener dinero electrónico, cuya validez está certificada por alguna autoridad financiera. Puede pagar una cantidad a otro usuario, de manera que de su cartera electrónica se reduce la cantidad, y se aumenta a la cartera electrónica del destinatario, garantizando propiedades difíciles de mantener con dinero de papel, como el falsificar dinero o recuperar el dinero de una cartera robada. Existen también sistemas para manejo de *cheques electrónicos* que evitan los problemas con los cheques de papel sin fondos, o sistemas de *tarjetas de crédito electrónicas* que evitan el uso no autorizado de un número de tarjeta de crédito. Una fuente de información acerca de comercio electrónico, sistemas de pago, transferencias de fondos, y otros sistemas relacionados es
http://www.semper.org/sirene/outsideworld/ecommerce.html. Aquí hay ligas a los sistemas que se describen a continuación. Recientemente apareció en [2] una reseña del estado del arte en sistemas de comercio electrónico.

En los sistemas de comercio electrónico siempre se identifican tres tipos de participantes. El *pagador* que transfiere una cantidad al *cobrador*, y la *institución financiera* que garantiza la validez del dinero involucrado, de diversas maneras, según el tipo de transacción y de sistema. Existen dos clases principales de sistemas, en cuanto al flujo de información se refiere. En sistemas de tipo *dinero efectivo*, primero el pagador y la institución se comunican, para tomar dinero de la cuenta del pagador y transferirlo a su cartera electrónica, que usualmente es algún tipo de dispositivo que combina hardware y software, con capacidad de comunicarse con la cartera electrónica del cobrador. Después, el pagador se pone en contacto con el cobrador, y se realiza la transferencia del dinero de la cartera electrónica de uno a la del otro. Finalmente, el cobrador puede hacer la transferencia de su cartera a su correspondiente institución financiera, o en algunos sistemas, inclusive directamente a otra cartera electrónica.

La otra clase de sistemas son los de tipo *cheques*. Aquí el pagador se comunica primero con el cobrador, enviándole algún tipo de "forma de pago" electrónica, como un cheque, un voucher de tarjeta de crédito, o de tarjeta de cajero electrónica. Tanto en sistemas de dinero efectivo como de cheques, hay una interacción directa entre el pagador y el cobrador, pero también hay algunos otros sistemas en los cuales esto no sucede, como en los de transferencia electrónica de fondos.

Es importante distinguir entre sistemas *fuera de línea* y sistemas *en línea*. En un sistema del primer tipo, es posible realizar una transacción entre el pagador y cobrador, sin necesidad de intervención de una institución financiera en ese momento. Esto es más deseable, pero mas difícil de lograr que en los sistemas en línea, que son la mayoría de los propuestos hasta el momento. Mientras que los sistemas en línea imponen mayores demandas en la infraestructura de comunicaciones, los sistemas fuera de línea requieren de dispositivos físicos confiables de tipo cartera electrónica para evitar duplicación de pagos.

Los requerimientos de seguridad se pueden clasificar de la siguiente manera.

Integridad. Un sistema no debe permitir tomar dinero del pagador sin su autorización. En ocasiones se requiere autorización del cobrador antes de depositarle dinero. El concepto clave es el de *autorización*, que puede incluir la cantidad y la descripción del bien intercambiado y condiciones del contrato, o solamente la especificación de la cantidad de dinero en cuestión, como en el caso de dinero electrónico. La manera en que se garantiza que el pagador ha autorizado la transacción, puede variar: mediante algún canal externo, como fax o voz; mediante el uso de passwords o PINs (como en el caso de cajeros electrónicos), o mediante firmas digitales.

Confidencialidad. En ocasiones es deseable proteger la identidad de las partes involucradas en la transacción, así como la cantidad de dinero y el tipo de bien intercambiado. En el caso de dinero electrónico, es deseable mantener las mismas propiedades de confidencialidad que tiene el dinero físico. Una moneda puede ser transferida del pagador al cobrador y de éste a otro, sin quedar rastros de por cuáles manos ha pasado la moneda o en qué transacciones ha sido utilizada. Lograr este requerimiento es más difícil de lo que podría parecer, en particular en vista de requerimientos opuestos que demandan que una moneda electrónica sea gastada una sola vez, y recuperable en caso de robos. Claramente, no es suficiente que un pagador transfiera un archivo que represente a la moneda al cobrador, ya que no habría nada que evite al pagador transferir el mismo archivo a otro cobrador, duplicando así la moneda. Inclusive en un sistema más sofisticado, un pagador que transfiere una moneda a un cobrador, podría reinicializar su sistema para pagar la misma moneda a otro cobrador. Es por esto que usualmente estos sistemas incluyen algún componente de hardware sellado, el cual ni siquiera el dueño puede modificar. Por otro lado, es difícil lograr la confidencialidad del pagador y volver su moneda anónima, y a la vez lograr que en caso de pérdida de la cartera electrónica u otras disputas, pueda recuperar el dinero. Claramente, en todos estos requerimientos las técnicas criptográficas juegan un papel fundamental.

Confiabilidad. Todo esto se complica aún más al requerir que el sistema debe funcionar correctamente y sin fallas en todo momento del día y de la noche. Un usuario debe poder ejecutar su transacción en cualquier momento, a pesar de cualquier tipo de falla en la red de comunicaciones. Es inevitable que en algún momento llegue a haber alguna falla en la red, en cuyo caso se debe garantizar la *atomicidad* de la transacción: o se ejecuta completamente, o no se ejecuta, pero no queda en algún estado intermedio.

Una de las propuestas más interesantes es el sistema CAFE, permitiendo pagos fuera de línea con garantía de confidencialidad, y recuperación de monedas perdidas (a expensas de pérdida de confidencialidad), además de permitir al cobrador utilizar la moneda recibida para pagar a su vez a otra persona sin necesidad de comunicación intermedia con la institución financiera. Esta propuesta europea utiliza tecnología de *firmas ciegas* en las que se firma un mensaje sin saber exactamente cual es su contenido. El pagador que desea retirar una moneda de su banco, genera una nueva moneda con un número de serie al azar, y la "encapsula" enviándola al banco para ser firmada ciegamente. Si el pagador es autorizado a retirar la moneda, el banco la firma y se la regresa. El pagador la extrae del encapsulamiento, y la puede enviar al cobrador. Cuando el cobrador la deposita en el banco, éste registra el número de serie para evitar su reutilización. Como el banco no podía ver el número de serie al firmarla, no puede relacionar la moneda con el retiro anterior del pagador, logrando así ambas propiedades: confidencialidad y evitar duplicación de pagos.

Otro sistema importante es SET, propuesto por MasterCard, Visa, GTE, IBM, Microsoft, Netscape, SAIC, Terisa y Verisign, para transacciones en línea basadas en tarjeta de crédito, y buscando preservar las relaciones actuales entre pagadores, comerciantes y bancos, mediante la transferencia segura de números de tarjetas de crédito, e interactuando con la infraestructura financiera existente. Al comienzo de una transacción, el comerciante se autentifica al pagador, enviando toda la información de pago del bien en cuestión. Entonces el pagador genera la forma de pago, estableciendo mediante criptografía propiedades como la confidencialidad del número de tarjeta de crédito, asociando la información de la orden con la del pago, y privacidad del comprador. El pagador firma esta forma de pago y la envía al comerciante. Éste al recibirla la envía a su banco para ser autorizada, el cual después de verificar la validez de toda la información contenida, envía un mensaje de autorización o de rechazo tanto al pagador como al comerciante. Aparentemente SET será pronto el estandar para compras mediante tarjeta de crédito en Internet.

Es importante notar que existen también sistemas para *micro-pagos*, en los cuales el diseño está optimizado para la realización de grandes cantidades de transacciones pero de montos muy pequeños cada una. Por ejemplo, se piensa en el cobro de unos centavos por el acceso a una página de web.

4.3 Web y Java

Al descubrir la enorme importancia del Web, se empezó a construir sistemas que no sólo permitían traer en forma transparente una página web con texto e imágenes, sino que convirtieran a la página en un objeto dinámico. Una página web puede incluir el código ejecutable de un programa, que extiende sus capacidades de despliegue de información y funcionalidad. Esto ocasiona que, en forma transparente al usuario, un navegador de web pueda estar bajando un programa (asociado a una página) de algún sitio desconocido y ejecutándolo en su máquina.

Navegadores y en general cualquier tecnología de programación que intente trabajar con código distribuible por medio de Internet para su ejecución local, se enfrenta a serios retos de seguridad: por un lado, para que una aplicación resulte de utilidad, debe poder ser capaz de accesar recursos de la máquina local o remota, o de alguna otra máquina en la red. Por otra parte, el hecho de que el código deba viajar por la red, lo expone a todo tipo de ataques. A esto se agrega que el código puede ser generado por un compilador con errores, o bien, el código puede crearse expresamente para causar daños o molestias.

La solución a estos problemas es más complicada de lo que pudiera parecer. Se tiene que considerar, por un lado, los mecanismos de seguridad necesarios al diseño del lenguaje en el cual se escribe el código distribuible; por otro lado, se debe proteger el código cuando viaja por la red; es necesaria la verificación del origen en la red del código; verificar que el código haya sido generado por un compilador valido. Se deben tener precauciones para proteger al ambiente de ejecución y controlar los requerimientos del código respecto a recursos auxiliares tales como archivos locales, bibliotecas, y dispositivos periféricos.

Al nivel de lenguaje, por ejemplo, deben tomarse decisiones tales como: permitir o no el acceso a memoria, los tipos de datos, apuntadores, el paso de parámetros, recolección de basura, etcétera, que tienen que ver con seguridad. La generación de código involucra la verificación de los esquemas de seguridad del lenguaje en tiempo de compilación o de ejecución, pero también asegurarse que el código que viaja por la red y que se tiene que ejecutar en el ambiente local, haya sido generado por un compilador válido. Igualmente, debe planearse el tipo de verificación que se efectuará al momento de compilar y el que se deja para el momento de ejecución.

Para proteger al código en su viaje por la red se requiere pensar en mecanismos de seguridad, tales como: cifrado de datos por medio de algoritmos criptográficos, esquemas de firma digital, certificados, etcétera. La verificación del origen de red legal del código, es decir, asegurarse que provenga de donde dice provenir, tiene que ver con esquemas de autenticación que deben implementarse para prevenir suplantaciones. Y considerar los correspondientes mecanismos de administración de llaves.

El aspecto que tiene que ver con la ejecución, involucra un navegador de red o visualizador de páginas. Como puerta de acceso a todos los recursos locales que la aplicación pueda requerir, el visualizador mismo debe implementar su propio esquema de seguridad que garantice que el acceso a los recursos será hecho de acuerdo a las reglas establecidas y a los permisos concedidos a la aplicación en particular. El navegador se convierte en un guardián de los recursos de la máquina local, y sus políticas de seguridad se añaden a las del sistema operativo, y deben interactuar correctamente con las de éste.

Cualquier solución de seguridad plantea requerimientos contradictorios. Considérese el ejemplo de un editor de texto que se baja por la red. Para que el código haga algo útil, requiere accesar recursos del sistema donde está corriendo. Al darle acceso a recursos, también se abre la posibilidad de usarlos en forma indebida y causar daños o molestias. Por lo tanto, es necesario restringir el acceso a los recursos del sistema. Pero, si el acceso se restringe demasiado, entonces se limita la utilidad del editor. Si no se le restringe el acceso al sistema de archivos, entonces, una versión maligna podría modificar archivos locales; pero si no se le da acceso al sistema de archivos, el editor pierde su utilidad.

Una solución de seguridad debe identificar: los *recursos* que se deben proteger y los *ataques* de los cuales se quiere proteger estos recursos. Los recursos que usualmente se consideran, son: sistema de archivos, red, RAM, dispositivos de salida, dispositivos de entrada, control de procesos, ambientes de usuario, llamadas al sistema. Los ataques se pueden clasificar en ataques de: privacidad, negación de servicio, integridad y molestias, entre otros. Finalmente, es necesario considerar los posibles *escenarios* de ataques a los recursos. Algunos ejemplos son: privacidad (enviar por email información acerca del usuario o la computadora), negación de servicio (abrir un número excesivo de ventanas, crear procesos de alta prioridad, utilizar el sistema local para correr procesos ajenos), integridad (corrupción de información, matar procesos, alterar memoria), molestias (desplegar anuncios groseros, hacer ruidos).

Es claro entonces que se está hablando de un sistema complejo que involucra desde algoritmos criptográficos, protocolos de nivel intermedio, seguridad del sistema operativo, e infraestructuras de distribución de llaves, políticas de seguridad, aspectos legales y éticos, etcétera. Y que todo esto debe ser integrado en un todo que garantice la operación confiable y segura del sistema. Antes de pasar a discutir la propuesta de Java, veamos un ejemplo de un ataque ingenioso que ilustra la problemática.

El Ataque del Simulacro

Vale la pena recordar un famoso caso de un ingenioso ataque que ejemplifica dificultades que debe enfrentar un sistema seguro de web. La idea de esta clase de ataque *spoofing* consiste en crear un contexto aparente para llevar a la víctima a tomar una decisión de seguridad inapropiada. Un ejemplo clásico de este tipo de ataques es el de los cajeros automáticos falsos [8], en el cual la fachada de un cajero automático hace creer a la víctima que se trata de uno real. La víctima inserta su tarjeta y teclea su número secreto, lo cual permite al cajero falso copiar esa información confidencial; una vez hecho esto, devuelve la tarjeta a la víctima diciéndole que el cajero no funciona en ese momento, y ésta ni siquiera se da cuenta del ataque.

Otro ataque de este estilo bien conocido, a veces llamado de *caballo de Troya*, es el de la terminal que pide un password, corriendo un programa falso, que despliega el letrero de manera igual al sistema original. El usuario, le da su password, el programa lo lee y graba para uso posterior, le contesta al usuario que escribió mal su password ("que raro, pensé que lo había yo escrito bien, pero bueno, ahora lo escribo otra vez"), y despliega otra vez el letrero de pedir password, pero esta vez el real, para que el usuario pueda entrar a sesión sin darse cuenta que algo malo sucedió.

El ingenioso ataque tipo spoofing propuesto por Felten, Balfanz, Dean, y Wallach ilustra las dificultades en crear un sistema web seguro. Es también un ataque de la clase llamada *hombre en medio* en la literatura de seguridad. El atacante es capaz de monitorear las páginas que la víctima visita, el contenido de sus interacciones con éstas (como la información en formas enviadas y recibidas), e inclusive puede modificar la información enviada o recibida; todo esto sin que la víctima se dé cuenta. Además, es un ataque muy efectivo, contra el cual no ayudan técnicas usuales de encriptamiento, y el cual es difícil de detectar; para una discusión en más detalle ver [7].

El ataque comienza cuando la víctima visita una página web del atacante, http://www.maloso.com/. Ahí se encuentra una liga que le interesa, digamos a Bancomer, y a partir de ahí la visita. El problema es que el atacante ha modificado la liga a Bancomer en su página, anteponiéndole su propia dirección:
http://www.maloso.com/http://www.bancomer.com.mx/, de manera que la solicitud de la víctima en lugar de ir a Bancomer, va a Maloso. Éste al recibirla se da cuenta que la solicitud es a Bancomer, por el URL enviado, pide la página correspondiente a Bancomer, y la reenvía a la víctima. Más aún, en lugar de reenviar la página de Bancomer tal cual a la víctima, la modifica para que todas las ligas en ella tengan antepuesto http://www.maloso.com/. Así, la víctima ve la página de Bancomer, aparentemente sin ningún problema. Pero al llamar otra página a partir de una liga de ésta, el pedido es enviado otra vez a Maloso, y de nuevo éste se encarga de pedir la página requerida, modificar sus ligas, y reenviarla a la víctima. Es decir, la víctima va navegando por todo el web, sin siquiera darse cuenta de que todas sus interacciones están pasando por Maloso, y posiblemente su contenido es alterado.

La Propuesta de Java

Java, como lenguaje de programación y como conjunto de herramientas para desarrollo de aplicaciones en red, propone su propio modelo de seguridad, con el objetivo primordial de evitar que un programa dañe el sistema de cómputo del usuario, y de que lea información privada del sistema [5, 18]. Conforme Java va evolucionando, el modelo de seguridad ha ido cambiando, e incorporando más funcionalidades que logran otras metas de seguridad además de las dos primordiales, algunas de éstas ya incluidas en la versión 1.2. Por ejemplo, *autenticación* de la identidad de los participantes; *encriptación* de la información transmitida; *auditación* de operaciones críticas para poderse analizar en caso de problemas; *controles* que eviten que un programa consuma demasiados recursos. Proveyendo de facilidades de empaquetamiento y firma de archivos, generación y administración de llaves y certificados, huellas digitales (message digests).

El modelo de seguridad utiliza como concepto central la *caja de arena* o *sandbox*, que define el espacio donde un programa puede operar. De manera análoga, la caja de arena o corral donde se deja a un niño jugando define una región y recursos (número y tipo de juguetes) que son suficientes para que se entretenga, mientras que protege al resto de la casa de él. Las componentes básicas del modelo de seguridad de Java permiten definir una caja de arena. Cada una de éstas tiene una función específica de seguridad, que se relacionan con las demás como un todo.

1. Lenguaje y Compilador.- A este nivel, Java introduce, desde el diseño del lenguaje del compilador, la idea de ofrecer seguridad a partir de un lenguaje robusto y de restricciones

a lo que se puede hacer a nivel de programación y de garantizar que esas restricciones se verifiquen a tiempo de compilación.

Java es un lenguaje fuertemente tipificado. La planificación de memoria no se deja al compilador sino que se hace hasta el momento de ejecución. La liberación de objetos en memoria que han dejado de ser referenciados, la hace Java de forma automática por medio de un recolector de basura. No se permite el uso de apuntadores.

El compilador de Java verifica que los programas no asignen ni manejen memoria simulando apuntadores ilegales. Es el intérprete quien decide dónde ligar campos y métodos a memoria. Rechaza código que no cumpla con las reglas del lenguaje relacionadas a su naturaleza fuertemente tipificada. Verifica que las operaciones sobre arreglos se ajusten a los límites sobre el tamaño de éstos. Verifica que el acceso a clases, variables, interfaces y métodos sea consistente con su declaración, evitando de esta manera que se usen de modo no previsto. Verifica que todo acceso a objeto se haga a través de su interfaz pública, asegurando con ello que, por ejemplo, los métodos de un objeto *private* permanezcan privados. El compilador evita que se lean variables no inicializadas y que se modifiquen constantes.

2. Verificador de Bytecodes.- Antes de ejecutar cualquier contenido ejecutable importado por la red, Java verifica que ese código (bytecodes) cumpla con las restricciones iniciales de seguridad impuestas por el modelo.

3. Cargador de Clases.- provee una clase que es requerida de accesar por la máquina virtual de Java. Para definir y cargar la clase, ejecuta varios pasos, como: si ya había sido cargada, la entrega inmediatamente; consulta al administrador de seguridad para verificar que el programa tiene permiso de accesar la clase; se encarga de que clases bajadas de la red no sobreescriban a clases locales; el código de bytes es enviado al verificador de bytecodes; carga otras clases referenciadas por la recién cargada. Se encarga de evitar conflictos de nombres de distintas clases. Java permite identificar el origen de una clase, si está firmada, y los permisos que la clase tiene otorgados.

4. Administrador de Seguridad, Controlador de Acceso.- El administrador de seguridad es el encargado de determinar la mayoría de los parámetros de la caja de arena. Cada vez que un programa desea hacer una operación potencialmente peligrosa, le pide permiso al administrador de seguridad. Java ofrece un paquete de clases que configuran las restricciones de seguridad que el usuario quiere imponer a cada aplicación en particular para accesar los recursos locales. Las aplicaciones Java, por omisión, no tienen ningún administrador de seguridad; cada aplicación puede implementar una política de seguridad específica, configurando al administrador de seguridad. Todo esto lo puede lograr el administrador, utilizando el controlador de acceso.

Es importante notar que el navegador o browser *no es una parte de Java. Hay navegadores que ni siquiera están escritos en Java. La seguridad de un navegador no está garantizada por el hecho de que Java sea seguro, sino de su propio diseño* *de seguridad, de cómo interactúa con el sistema operativo de la máquina donde se ejecuta, y de cómo usa el modelo de seguridad de Java, si es que está escrito en Java.*

Como ejemplo de las restricciones de seguridad implementadas sobre un visualizador, listaremos algunas típicas de Netscape 2.0 para un applet que se carga de la red:

1. No puede crear un cargador de clases o un administrador de seguridad.
2. No puede crear clases en el espacio de nombres local.
3. No puede accesar paquetes locales fuera de los estándares.
4. No puede accesar archivos y directorios del sistema local.
5. Sólo puede hacer conexiones de red al host de donde se cargó.
6. No puede crear o instalar manejadores de protocolos o implementar sockets.
7. No puede leer, del sistema local, información sobre el usuario.
8. No puede modificar utilerías del sistema local.
9. No puede terminar otros programas o el sistema mismo.
10. No puede accesar hilos de control (*threads*) fuera de su grupo.

Java ofrece la opción de firma de applets y archivos, para garantizar que no se carguen applets o archivos sin ser verificados por un esquema de firma digital. Otro problema serio es que cuando se carga una aplicación para ser ejecutada localmente, esta aplicación está integrada por varios archivos, cada uno de los cuales se cargan uno a uno, con la consecuente espera. Como solución, Java ofrece empaquetar en un sólo archivo, todos los archivos relacionados a una aplicación, ahorrando así tiempo de espera. Además este archivo integrado se puede firmar, lo que equivale a firmar todo el applet. La facilidad que otorga este beneficio integrado, se llama Java ARchive (JAR) y está disponible a partir de JDK 1.1

Java implementa adicionalmente algoritmos criptográficos de llave pública y de llave secreta, así como también algoritmos para certificar archivos y documentos como una facilidad para que el usuario configure su propio esquema de control y transferencia de archivos y documentos.

El modelo de la caja de arena, y su uso en un navegador como Netscape, tiene la limitación de que un applets bajado de la red está sujeto a estrictas restricciones de seguridad (como no poder accesar el sistema local de archivos o iniciar conexiones a sitios distintos del cual fue bajado), que limitan su utilidad, y estas limitaciones son las mismas para cualquier applet, independientemente de su origen. La propuesta de Wallach [22] permite otorgar permisos condicionales a programas escritos en Java. Un administrador de seguridad examina las solicitudes de acceso a recursos del programa, y decide si otorgarlas o no, dependiendo de las políticas de seguridad definidas por el usuario, y utilizando tecnología de firmas digitales.

References

[1] R. J. Anderson, "Why cryptosistems fail", *Communications of the ACM*, v. 37, n. 11, Nov. 1994, 32–40.

[2] N. Asokan, P. A. Janson, M. Steiner, M. Waidner, "The state of the art in electronic payment systems", IEEE *Computer*, septiembre 1997, 28–35.

[3] Dan Boneh, "Twenty years of attacks on the RSA cryptosystem", *Notices of the American Mathematical Society*, febrero 1999, 203–213.

[4] James Arlin Cooper, *Computer and Communications Security*, McGraw-Hill, 1989.

[5] Java Technology Home Page, http://java.sun.com/security

[6] CERT Advisories, http://www.cert.org

[7] Edward W. Felten, Dirk Balfanz, Drew Dean, and Dan S. Wallach, "Web Spoofing: An Internet Con Game", 20th National Information Systems Security Conference, Oct. 1997.

[8] P. G. Neumann, *Computer-Related Risks*, ACM Press, New York, 1995.

[9] Jalal Feghhi, Peter Williams, Jalil Feghhi, *Digital Certificates: Applied Internet Security*, Addison-Wesley, 1998.

[10] M. Fites, P. Kratz, A. Brebner, "Control and Security of Computer Information Systems", Computer Science Press, 1989.

[11] Warwick Ford, Michael S. Baum, *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*, Prentice Hall, 1997.

[12] Simson Garfinkel, Gene Spafford, *Practical Unix & Internet Security*, O'Reilly & Assoc., 2nd ed., 1996.

[13] Oded Goldreich, R. L. Graham (Editor), B. Korte (Editor), *Modern Cryptography, Probabilistic Proofs and Pseudorandomness (Algorithms and Combinatiorics, Vol 17)*, Springer Verlag, February 1999.

[14] Charlie Kaufman, Radia Perlman, Mike Speciner, *Network Security: Private Communication in a Public World*, Prentice Hall, 1995.

[15] D. Kahn, *The Codebreakers: The Story of Secret Writing*, New York, Macmillan Pub., 1967.

[16] Neal Koblitz, *A Course in Number Theory and Cryptography (Graduate Texts in Mathematics, No 114)*, Springer-Verlag, 1994.

[17] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of Applied Cryptography (CRC Press Series on Discrete Mathematics and Its Applications)*, CRC PRess, 1996.

[18] Scott Oaks, *Java Security*, O'Reilly & Assoc., 1998.

[19] Rolf Oppliger, *Authentication Systems for Secure Networks,* Artech House, 1996.

[20] Michael Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

[21] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, John Wiley & Sons, 1995.

[22] Dan S. Wallach, "A New Approach to Mobile Code Security", PhD Thesis, Princeton University, enero 1999.

[23] G. Welchman, *The Hut Six Story: Breaking the Enigma Codes*, New York, McGraw-Hill, 1982.

[24] Philip R. Zimmermann, *The Official Pgp User's Guide*, MIT Press, June 1995.