Criptografía y Seguridad en Computadores

Manuel José Lucena López

Departamento de Informática

Escuela Politécnica Superior

Universidad de Jaén

e-mail: mlucena@ujaen.es

–¿Qué significa *habla, amigo y entra*? –preguntó Merry.

–Es bastante claro –dijo Gimli–. Si eres un amigo, dices la contraseña y las puertas se abren y puedes entrar.

–Sí –dijo Gandalf–, es probable que estas puertas estén gobernadas por palabras...

El Señor de Los Anillos

J.R.R. Tolkien

Copyright

Agradecimientos

A Loles, ella sabe por qué.

A los chicos de Kriptópolis, por darme esta oportunidad.

A mis alumnos, por aguantarme cada año.

A todos los que alguna vez han compartido sus conocimientos, por enriquecernos a todos.

Prefacio

El presente documento ha sido elaborado originalmente como apoyo a la asignatura "Criptografía y Seguridad en Computadores", de 3^o Curso de *Ingeniería Técnica en Informática de Gestión*, de la Universidad de Jaén.

No se pretende que estos apuntes sustituyan a la bibliografía de la asignatura, ni a las clases teóricas, sino que sirvan más bien como complemento a los apuntes que el alumno debe tomar en clase. Asímismo, no debe considerarse un documento definitivo y exento de errores, si bien ha sido elaborado con detenimiento y revisado exhaustivamente. El autor pretende que sea mejorado y ampliado con cierta frecuencia, lo que probablemente desembocará en sucesivas versiones, y para ello nadie mejor que los propios lectores para plantear dudas, buscar errores, y sugerir mejoras.

Jaén, 29 de junio de 1999.

Índice General

Parte I

Preliminares

Capítulo 1

Introducción

A lo largo de 1995 y principios de 1996, los profesores José Ignacio Peláez Sánchez, Antonio Sánchez Solana y Manuel Lucena López elaboraron una Colección de Apuntes para la asignatura 'Criptografía y Seguridad en Computadores'. Varios años han pasado desde entonces, y, como cabía esperar en una disciplina de tan rápida evolución, las cosas han cambiado. Algunos algoritmos han perdido parte de su interés (como es el caso de DES, que fue *vencido* el verano de 1998), nuevas técnicas han surgido o se han popularizado (PGP es un claro ejemplo de que para el usuario de a pie se puede conseguir auténtica privacidad), temas que antes tenían un interés limitado se han convertido en fundamentales (la rápida expansión de Internet obliga no sólo al profesional, sino al usuario medio, a tener ciertos conocimientos básicos sobre seguridad), etc.

Por esta razón la asignatura debe evolucionar para absorber la demanda de conocimientos criptográficos que un profesional de la Informática debe poseer. La presente colección de apuntes viene a cubrir ese vacío.

1.1 Algunas notas sobre la Historia de la Criptografía

La Criptografía moderna nace al mismo tiempo que las computadoras. Durante la Segunda Guerra Mundial, en un lugar llamado Bletchley Park, un grupo de científicos entre los que se encontraba Alan Turing, trabajaba en el proyecto ULTRA tratando de descifrar los mensajes enviados por el ejército alemán con la más sofisticada máquina de codificación ideada hasta entonces: la máquina ENIGMA. Este grupo de científicos empleaba el que hoy se considera el primer computador (aunque esta información permaneció en secreto hasta mediados de los 70). Su uso y la llegada del polaco Marian Rejewski tras la invasión de Polonia cambiarían el curso de la Historia.

Desde entonces hasta hoy ha habido un crecimiento espectacular de la tecnología criptográfica, si bien la mayor parte de estos avances se mantenían (y se siguen manteniendo, según algunos) en secreto. Financiadas fundamentalmente por la NSA (Agencia Nacional de

Seguridad de los EE.UU.), la mayor parte de las investigaciones hasta hace relativamente poco tiempo han sido tratadas como secretos militares. Sin embargo en los últimos años, investigaciones serias llevadas a cabo en universidades han logrado que la criptografía sea una ciencia al alcance de todo el mundo, y que se convierta en la piedra angular de asuntos tan importantes como el comercio en Internet.

Muchas son las voces que claman por la disponibilidad pública de la criptografía. La experiencia ha demostrado que la única manera de tener buenos algoritmos es que éstos sean públicos, para que puedan ser sometidos al escrutinio de toda la comunidad científica. Casos claros de oscurantismo y de sus consecuencias han sido la caída del algoritmo que emplean los teléfonos GSM en menos de cuarenta y ocho horas desde que se filtró su código. Además se puso en evidencia la deliberada *debilitación* del algoritmo que los gobiernos habían impuesto a sus creadores para facilitar las escuchas por parte de sus servicios de espionaje. Otro ejemplo son los graves problemas de seguridad que presentaba el protocolo de comunicaciones seguras punto a punto que Microsoft incluía en Windows NT. La seguridad no debe basarse en mantener los algoritmos ocultos, puesto que éstos, tarde o temprano, acaban siendo analizados y descritos, sino en su resistencia demostrada tanto teórica como prácticamente, y la única manera de demostrar la resistencia de un algoritmo es sometiéndolo a todo tipo de ataques.

Es imposible desligar la Criptografía moderna de todas las consideraciones políticas, filosóficas y morales que suscita. Recordemos, por ejemplo, que el software criptográfico está sujeto en EE.UU. a las mismas leyes que el armamento nuclear, y que en Europa se pretende elaborar legislaciones parecidas. Una consecuencia inmediata de esto es que las versiones que se exportan de los exploradores de Internet más extendidos (Netscape y Explorer), incorporan una seguridad *débil*, impidiendo, por ejemplo, establecer conexiones realmente seguras para conectarse a un banco (aunque no se informa al usuario de ello). Otra de las líneas de debate es la intención de algunos gobiernos de almacenar todas las claves privadas de sus ciudadanos y considerar ilegales aquellas que no estén registradas. Es como pedirnos a todos que le demos a la policía una copia de las llaves de nuestra casa (con el pretexto, por supuesto, de luchar contra el terrorismo y el narcotráfico).

En el ojo del huracán se encuentra actualmente la red *Echelon*, que constituye una de las mayores amenazas contra la libertad de toda la Historia de la Humanidad. Básicamente se trata de una red, creada por la NSA en 1980 (sus *precursoras* datan de 1952) en colaboración con Gran Bretaña, Australia y Nueva Zelanda, para monitorizar prácticamente todas las comunicaciones electrónicas (teléfono, e-mail y fax principalmente) del planeta, y buscar de manera automática ciertas palabras clave. La información obtenida iría a la NSA, que luego podría a su vez brindársela a otros países. En el momento de escribir este documento, (marzo de 1999) el Parlamento Europeo ha sido interpelado por un comité de expertos para que se compruebe la posible vulneración de derechos fundamentales por parte de esta red (más información en http://jya.com/stoa-atpc.htm). El pretexto es, nuevamente, la lucha contra el terrorismo, pero podría ser empleado tanto para espionaje industrial (como presuntamente ha hecho durante años el Gobierno Francés, poniendo a disposición de sus empresas secretos robados a empresas extranjeras), como para el *control* de aquellas personas que pueden representar amenazas políticas a la *estabilidad* de la sociedad moderna. Sin embargo, parece

que las intenciones de la Unión Eurpea son otras: el despliegue de su propia red de vigilancia electrónica, llamada *Enfopol*, en la que lleva trabajando unos diez años.

1.2 Números *Grandes*

Los algoritmos criptográficos emplean claves con un elevado número de bits, y usualmente se mide su calidad por la cantidad de esfuerzo que se necesita para romperlos. El tipo de ataque más simple es la *fuerza bruta*, que simplemente trata de ir probando una a una todas las claves. Por ejemplo, el algoritmo DES tiene 2^{56} posibles claves. ¿Cuánto tiempo nos llevaría probarlas todas si, por ejemplo, dispusiéramos de un computador capaz de hacer un millón de operaciones por segundo? Tardaríamos... ¡más de 2200 años! Pero ¿y si la clave del ejemplo anterior tuviera 128 bits? El tiempo requerido sería de 10^{24} años.

Es interesante dedicar un apartado a tratar de fijar en nuestra imaginación la magnitud real de este tipo de números. En la tabla 1.1 podemos observar algunas magnitudes que nos ayudarán a comprender mejor la auténtica magnitud de muchos de los números que veremos en estos apuntes. Observándola podremos apreciar que 10^{24} años es aproximadamente cien billones de veces la edad del universo (y eso con un ordenador capaz de ejecutar el algoritmo de codificación completo un millón de veces por segundo). Esto nos debería disuadir de emplear mecanismos basados en la fuerza bruta para *reventar* claves de 128 bits.

Para manejar la tabla con mayor rapidez, recordemos que un millón es aproximadamente 2^{20}, y que un año tiene más o menos 2^{24} segundos. Recorrer completamente un espacio de claves de, por ejemplo, 256 bits a razón de un millón por segundo supone $2^{256-44} = 2^{212}$ años.

Valor	Número
Probabilidad de ser fulminado por un rayo (por día)	1 entre 9.000.000.000 (2^{33})
Probabilidad de ganar el primer premio de la Primitiva	1 entre 13.983.816 (2^{23})
Probabilidad de ganar el primer premio de la Primitiva y caer fulminado por un rayo el mismo día	1 entre 2^{56}
Tiempo hasta la próxima glaciación	14.000 (2^{14}) años
Tiempo hasta que el Sol estalle	10^9 (2^{30}) años
Edad del Planeta Tierra	10^9 (2^{30}) años
Edad del Universo	10^{10} (2^{34}) años
Número de átomos en el Planeta Tierra	10^{51} (2^{170})
Número de átomos en el Sol	10^{57} (2^{190})
Número de átomos en la Vía Láctea	10^{67} (2^{223})
Número de átomos en el Universo (excluyendo materia oscura)	10^{77} (2^{265})
Volumen del Universo	10^{84} (2^{280}) cm^3
Si el Universo fuera Cerrado:	
Vida total del Universo	10^{11} (2^{37}) años
Si el Universo fuera Abierto:	
Tiempo hasta que una estrella *ligera* se enfría	10^{11} (2^{47}) años
Tiempo hasta que los planetas se desprenden de una estrella	10^{15} (2^{50}) años
Tiempo hasta que las estrellas se desprenden de las galaxias	10^{19} (2^{64}) años
Tiempo hasta que las órbitas decaen por radiación gravitacional	10^{20} (2^{67}) años
Tiempo hasta que los agujeros negros se desvanecen (Proceso Hawking)	10^{64} (2^{213}) años
Tiempo hasta que toda la materia se vuelva líquida	10^{65} (2^{216}) años
Tiempo hasta que toda la materia se vuelva sólida	$10^{10^{26}}$ años
Tiempo hasta que toda la materia colapse en agujeros negros	$10^{10^{76}}$ años

Tabla 1.1: Números *grandes*

Capítulo 2

Conceptos Básicos sobre Criptografía

2.1 Criptografía

Según el Diccionario de la Real Academia, la palabra Criptografia proviene del griego $\kappa\rho\upsilon\pi\tau\acute{o}\varsigma$, que significa oculto, y $\gamma\rho\acute{a}\varphi\epsilon\iota\nu$, que significa escritura, y su definición es: *"Arte de escribir con clave secreta o de un modo enigmático"*. Obviamente la Criptografía hace años que dejó de ser un arte para convertirse en una técnica, o más bien un conglomerado de técnicas, que tratan sobre la protección (ocultamiento frente a observadores no autorizados) de la información. Entre las disciplinas que engloba cabe destacar la Teoría de la Información, la Teoría de Números (o Matemática Discreta, que estudia las propiedades de los números enteros), y la Complejidad Algorítmica.

Existen dos documentos fundamentales, uno escrito por Claude Shannon en 1948 *("A Mathematical Theory of Communication")*, en el que se sentaban las bases de la Teoría de la Información, y que junto con otro artículo posterior del mismo autor sirvió de base para la Criptografía moderna. El otro trabajo, publicado por Whitfield Diffie en 1975, introducía el concepto de Criptografía de Llave Pública, abriendo enormemente el abanico de aplicación de esta disciplina.

2.2 Criptosistema

Definiremos un criptosistema como una quíntupla (M, C, K, E, D), donde:

- M representa el conjunto de todos los mensajes sin cifrar (lo que se denomina texto plano, o *plaintext*) que pueden ser enviados.
- C representa el conjunto de todos los posibles mensajes cifrados, o criptogramas.

- K representa el conjunto de claves que se pueden emplear en el criptosistema.
- E es el conjunto de *transformaciones de cifrado* o familia de funciones que se aplica a cada elemento de M para obtener un elemento de C. Existe una transformación diferente E_k para cada valor posible de la clave k.
- D es el conjunto de *transformaciones de descifrado*, análogo a E.

Todo criptosistema ha de cumplir la siguiente condición:

$$D_k(E_k(m)) = m \tag{2.1}$$

es decir, que si tenemos un mensaje m, lo ciframos empleando la clave k y luego lo desciframos empleando la misma clave, obtenemos de nuevo el mensaje original m.

Existen dos tipos fundamentales de criptosistemas:

- *Criptosistemas simétricos o de clave privada.* Son aquellos que emplean la misma clave k tanto para cifrar como para descifrar. Presentan el inconveniente de que para ser empleados en canales de comunicación la clave k debe estar tanto en el emisior como en el receptor, lo cual nos lleva preguntarnos cómo transmitir la clave de forma segura.
- *Criptosistemas asimétricos o de llave pública*, que emplean una doble clave (k_p, k_P). k_p se conoce como *clave privada* y k_P se conoce como *clave pública.* Una de ellas sirve para la transformación E de cifrado y la otra para la transformación D de descifrado. En muchos casos son intercambiables, esto es, si empleamos una para cifrar la otra sirve para descifrar y viceversa. Estos criptosistemas deben cumplir además que el conocimiento de la clave pública k_P no permita calcular la clave privada k_p. Ofrecen un abanico superior de posibilidades, pudiendo emplearse para establecer comunicaciones seguras por canales inseguros (puesto que únicamente viaja por el canal la clave pública, que sólo sirve para cifrar), o para llevar a cabo autentificaciones.

En la práctica lo que se emplea es una combinación de estos dos tipos de criptosistemas, puesto que los segundos presentan el inconveniente de ser computacionalmente mucho más costosos que los primeros. Lo que se hace *en el mundo real* es codificar los mensajes (largos) mediante algoritmos simétricos, que suelen ser muy eficientes, y luego emplear la *criptgografía asimétrica* para codificar la clave (que suele ser corta en comparación con los mensajes).

2.3 Esteganografía

La esteganografía (o el empleo de *canales subliminales*) consiste en ocultar en el interior de información aparentemente inocua, otro tipo de información (cifrada o no). Este método ha cobrado bastante importancia últimamente debido a que permite burlar diferentes sistemas de control. Supongamos que un disidente político quiere enviar un mensaje fuera de su

país. Si lo envía codificado, las autoridades difícilmente permitirán que el mensaje salga del país independientemente de que puedan acceder a su contenido, mientras que si ese mismo mensaje viaja camuflado en el interior de una imagen digital para una felicitación navideña, tendrá muchas más posibilidades de llegar a su destino.

Mención especial merece el uso de la esteganografía para exportar información sin violar las leyes restrictivas que, con respecto a la Criptografía *fuerte*, existen en algunos países. El mensaje se envía como texto plano, pero entremezclado con cantidades ingentes de *basura*. El destinatario empleará técnicas esteganográficas para separar la información útil del resto. Esta técnica se conoce como *chaffing and winnowing*, que vendría a traducirse como *llenar de paja y separar el grano de la paja*. En consecuencia, tenemos un mecanismo para transmitir información no cifrada, pero que sólo puede ser reconstruida por el destinatario, con lo que en realidad hemos logrado proteger la información sin usar la Criptografía. Este sistema surgió en marzo de 1998, propuesto por Ronald L. Rivest (uno de los creadores de RSA), como desafío a la política restrictiva del Gobierno de los EE.UU. con respecto a la Criptografía. No deja de ser en cierto modo una curiosidad, debido a lo enormemente grandes que son los mensajes en comparación con la cantidad de texto útil que se puede incluir en ellos.

2.4 Criptoanálisis

El *criptoanálisis* consiste en comprometer la seguridad de un criptosistema. Esto se puede hacer descifrando un mensaje sin conocer la llave, o bien obteniendo la llave empleada para cifrar algún mensaje. No se considera criptoanálisis el descubrimiento de un algoritmo secreto de cifrado; hemos de considerar por el contrario que el algoritmo de cifrado siempre es conocido.

Uno de los tipos de análisis más comunes que se realizan es el de *texto plano escogido*, que parte de que conocemos una serie de pares de textos planos y textos cifrados, codificados con la misma clave. Esta situación se suele dar cuando tenemos acceso al dispositivo de cifrado y éste nos permite efectuar operaciones, pero no nos permite leer su clave (por ejemplo, las tarjetas de los teléfonos móviles GSM). Cuando el sistema es débil, basta con escoger unos cientos de mensajes y codificarlos, y de esta forma tendremos información suficiente para deducir la clave empleada.

También podemos tratar de criptoanalizar un sistema aplicando el algoritmo de descifrado a un mensaje codificado que poseemos y comprobar si lo que nos sale *tiene sentido* como posible texto plano. Este método y todos los que buscan exhaustivamente por el espacio de claves K, se denominan *ataques por la fuerza bruta*, y en muchos casos no suelen considerarse técnicas de criptoanálisis, reservándose este término para aquellos mecanismos que explotan posibles debilidades intrínsecas en el algoritmo de cifrado. Se da por supuesto que el espacio de claves para cualquier criptosistema digno de interés ha de ser suficientemente grande como para que un ataque por la fuerza bruta no sea viable. Hemos de tener en cuenta no obstante que la capacidad de cálculo de las computadoras crece a gran velocidad, por lo que algoritmos que hace unos años eran resistentes frente a ataques por la fuerza bruta hoy por hoy pueden ser inseguros (como es el caso de DES).

Un par de métodos de criptoanálisis que han dado interesantes resultados son el *análisis diferencial* y el *análisis lineal*. El primero de ellos, partiendo de pares de mensajes con diferencias mínimas (usualmente de un bit), estudia las variaciones que existen entre los mensajes cifrados correspondientes, tratando de identificar patrones comunes. El segundo emplea operaciones XOR entre algunos bits del texto plano y algunos bits del texto cifrado, obteniendo finalmente un único bit. Si realizamos esto con muchos pares de texto plano-texto cifrado podemos obtener una probabilidad p en ese bit que calculamos. Si p está suficientemente sesgada (no se aproxima a $\frac{1}{2}$), tendremos la posibilidad de recuperar la clave.

Otro tipo de análisis, esta vez para los algoritmos asimétricos, consistiría en tratar de deducir la llave privada a partir de la pública. Suelen ser técnicas analíticas que básicamente intentan resolver los problemas de elevado coste computacional en los que se apoyan estos criptosistemas: factorización, logaritmos discretos, etc. Mientras estos problemas genéricos permanezcan sin solución, podremos seguir confiando en estos algoritmos.

La Criptografía no sólo se emplea para proteger información, también se utiliza para permitir su autentificación, es decir, para identificar al autor de un mensaje e impedir que nadie suplante su personalidad. En estos casos surge un nuevo tipo de criptoanálisis que está encaminado únicamente a permitir que elementos falsos pasen por buenos. Puede que ni siquiera nos intere descifrar el mensaje original, sino simplemente poder sustituirlo por otro falso y que supere las pruebas de autentificación.

2.5 Compromiso entre Sistema y Criptoanálisis

En la sección 3.5 (pág. 35) veremos que pueden existir sistemas idealmente seguros, capaces de resistir cualquier ataque. También veremos que estos sistemas en la práctica carecen de interés práctico, lo cual nos lleva a tener que adoptar un compromiso entre el coste del sistema (tanto computacional como de almacenamiento, e incluso económico) frente a su resistencia a diferentes ataques criptográficos.

La información posee un tiempo de vida, a partir del cual pierde su valor. Los datos sobre la estrategia de inversiones a largo plazo de una gran empresa, por ejemplo, tienen un mayor periodo de validez que la exclusiva periodística de una sentencia judicial que se va a hacer pública al día siguiente. Será suficiente, pues, tener un sistema que garantice que el tiempo que se puede tardar en comprometer la seguiridad del sistema es mayor que el tiempo de vida de la propia información que éste alberga. Esto no suele ser fácil, sobre todo porque no tardará lo mismo un *oponente* que disponga de una única computadora de capacidad modesta, que otro que emplee una red de supercomputadoras. Por eso también ha de tenerse en cuenta si la información que queremos proteger vale más que el esfuerzo de criptoanálisis que va a necesitar, porque entonces puede que no esté segura. La seguridad de los criptosistemas se suele medir en términos del número de computadoras y del tiempo necesarios para romperlos, y a veces simplemente en función del dinero necesario para llevar a cabo esta tarea con garantías de éxito.

En cualquier caso hoy por hoy existen sistemas que son muy poco costosos (o incluso

gratuitos, como el PGP), y que nos garantizan un nivel de protección tal que toda la potencia de cálculo que actualmente hay en el planeta sería insuficiente para romperlos.

Tampoco conviene depositar excesiva confianza en el algoritmo de cifrado, puesto que en el proceso de protección de la información existen otros puntos débiles que deben ser tratados con un cuidado exquisito. Por ejemplo, no tiene sentido emplear algoritmos con niveles de seguridad extremadamente elevados si luego escogemos *passwords* ridículamente fáciles de adivinar. Una práctica muy extendida por desgracia es la de escoger palabras clave que contengan fechas, nombres de familiares, nombres de personajes o lugares de ficción, etc. Son las primeras que un atacante avispado probaría. Tampoco es una práctica recomendable apuntarlas o decírselas a nadie, puesto que si la clave cae en malas manos, todo nuestro sistema queda comprometido, por buenos que sean los algoritmos empleados.

2.6 Seguridad

El concepto de seguridad en la información es mucho más amplio que la simple protección de los datos a nivel *lógico*. Para proporcionar una seguridad real hemos de tener en cuenta múltiples factores, tanto internos como externos. En primer lugar habría que caracterizar el sistema que va a albergar la información para poder identificar las amenazas, y en este sentido podríamos hacer la siguiente subdivisión:

1. *Sistemas aislados.* Son los que no están conectados a ningún tipo de red. De unos años a esta parte se han convertido en minoría, debido al auge que ha experimentado Internet.
2. *Sistemas interconectados.* Hoy por hoy casi cualquier ordenador pertenece a alguna red, enviando y recogiendo información del exterior casi constantemente. Esto hace que las redes de ordenadores sean cada día más complejas y supongan un peligro potencial que no puede en ningún caso ser ignorado.

En cuanto a las cuestiones de seguridad que hemos de fijar podríamos clasificarlas de la siguiente forma:

1. *Seguridad física.* Englobaremos dentro de esta categoría a todos los asuntos relacionados con la salvaguarda de los soportes físicos de la información, más que de la información propiamente dicha. En este nivel estarían, entre otras, las medidas contra incendios y sobrecargas eléctricas, la prevención de ataques terroristas, las políticas de *backup*, etc. También se suelen tener en cuenta dentro de este punto aspectos relacionados con la restricción de acceso físico a las computadoras únicamente a personas autorizadas.
2. *Seguridad de la información.* En este apartado prestaremos atención a la preservación de la información frente a observadores no autorizados. Para ello podemos emplear tanto criptografía simétrica como asimétrica, estando la primera únicamente indicada en sistemas aislados, ya que si la empleáramos en redes, al tener que transmitir la clave por el canal de comunicación, estaríamos asumiendo un riesgo excesivo.

3. *Seguridad del canal de comunicación.* Los canales de comunicación rara vez se consideran seguros. Debido a que en la mayoría de los casos escapan a nuestro control, ya que pertenecen a terceros, resulta imposible asegurarse de que no están siendo escuchados o intervenidos.

4. *Problemas de autentificación.* Debido a los problemas del canal de comunicación, es necesario asegurarse de que la información que recibimos en la computadora viene de quien realmente creemos que viene. Para esto se suele emplear criptografía asimétrica en conjunción con funciones *resumen* (ver sección 10.1, página 93).

5. *Problemas de suplantación.* En las redes tenemos el problema añadido de que cualquier usuario autorizado puede acceder al sistema desde fuera, por lo que hemos de confiar en sistemas fiables para garantizar que los usuarios no están siendo suplantados por intrusos. Normalmente se emplean mecanismos basados en *password* para conseguir esto.

Parte II

Fundamentos Teóricos de la Criptografía

Capítulo 3

Teoría de la Información

Comenzaremos el estudio de los fundamentos teóricos de la Criptografía dando una serie de nociones básicas sobre Teoría de la Información, introducida por Claude Shannon a finales de los años cuarenta. Esta disciplina nos permitirá efectuar una aproximación teórica al estudio de la seguridad de cualquier algoritmo criptográfico.

3.1 Cantidad de Información

Vamos a introducir este concepto partiendo de su idea intuitiva. Para ello analizaremos el siguiente ejemplo: supongamos que tenemos una bolsa con nueve bolas negras y una blanca. ¿Cuánta información obtenemos si nos dicen que han sacado una bola blanca de la bolsa?. ¿Y cuánta obtenemos si después sacan otra y nos dicen que es negra?

Obviamente, la respuesta a la primera pregunta es que nos aporta bastante información, puesto que estábamos *casi seguros* de que la bola tenía que salir negra. Análogamente si hubiera salido negra diríamos que ese suceso *no nos extraña* (nos aporta poca información). En cuanto a la segunda pregunta, claramente podemos contestar que no nos aporta ninguna información, ya que al no quedar bolas blancas *sabíamos* que iba a salir negra.

Podemos fijarnos en la cantidad de información como una medida de la disminución de incertidumbre acerca de un suceso. Por ejemplo, si nos dicen que el número que ha salido en un dado es menor que dos, nos dan más información que si nos dicen que el número que ha salido es par.

Se puede decir que la cantidad de información que nos aporta conocer un hecho es directamente proporcional al número posible de estados que éste tenía a priori. Si inicialmente teníamos diez posibilidades, conocer el hecho nos proporciona más información que si inicialmente tuviéramos dos. Por ejemplo, supone mayor información conocer la combinación ganadora del próximo sorteo de la Lotería Primitiva, que saber si una moneda lanzada al aire va a salir cara o cruz. Claramente es más fácil acertar en el segundo caso, puesto que el número

de posibilidades a priori (y por tanto la incertidumbre, suponiendo sucesos equiprobables) es menor.

También la cantidad de información es proporcional a la probabilidad de un suceso. En el caso de las bolas tenemos dos sucesos: sacar bola negra, que es más probable, y sacar bola blanca, que es menos probable. Sacar una bola negra aumenta nuesto grado de *certeza* inicial de un 90% a un 100%, proporcionándonos una ganancia del 10%. Sacar una bola blanca aumenta esa misma certeza en un 90%. Podemos considerar la disminución de incertidumbre proporcional al aumento de certeza, por lo cual podemos decir que el primer suceso (sacar bola negra) aporta menos información.

A partir de ahora, con objeto de simplificar la notación, vamos a emplear una variable aleatoria V para representar los posibles sucesos que nos podemos encontrar. Notaremos el suceso i-ésimo como x_i, $P(x_i)$ será la probabilidad asociada a dicho suceso, y n será el número de sucesos.

Supongamos ahora que sabemos con toda seguridad que el único valor que puede tomar V es x_i. Saber el valor de V no nos va a aportar ninguna información (lo conocemos de antemano). Por el contrario, si tenemos una certeza del 99% sobre la posible ocurrencia del valor x_i, obtener un x_j nos aportará bastante información, como ya hemos visto. Este concepto es cuantificable y se puede definir de la siguiente forma:

$$I_i = -\log_2 (P(x_i)) \tag{3.1}$$

siendo $P(x_i)$ la probabilidad del estado x_i. Obsérvese que si la probabilidad de un estado fuera 1 (máxima), la cantidad de información que nos aporta es igual a 0, mientras que si su probabilidad se acercara a 0, tendería a $+\infty$ (esto es lógico, un suceso que no puede suceder nos aportará una cantidad infinita de información si llega a suceder).

3.2 Entropía

Sumando ponderadamente las cantidades de información de todos los posibles estados de una variable aleatoria V, obtenemos:

$$H(V) = -\sum_{i=1}^{n} P(x_i) \log_2 [P(x_i)] = \sum_{i=1}^{n} P(x_i) \log_2 \left[\frac{1}{P(x_i)}\right] \tag{3.2}$$

Esta magnitud $H(V)$ se conoce somo la *entropía* de la variable aleatoria V. Sus propiedades son:

i. $0 \leq H(V) \leq \log_2 N$

ii. $H(V) = 0 \iff \exists i$ tal que $P(x_i) = 1$ y $P(x_j) = 0 \; \forall i \neq j$

iii. $H(x_1, x_2 \ldots x_n) = H(x_1, x_2 \ldots x_n, x_{n+1})$ si $P(x_{n+1}) = 0$

Obsérvese que la entropía es proporcional a la longitud media de los mensajes necesaria para codificar una serie de valores de V de manera óptima dado un alfabeto cualquiera. Esto quiere decir que cuanto más probable sea un valor individual, aportará menos información cuando aparezca, y podremos codificarlo empleando un mensaje más corto. Si $P(x_i) = 1$ no necesitaríamos ningún mensaje, puesto que sabemos de antemano que V va a tomar el valor x_i, mientras que si $P(x_i) = 0.9$ parece más lógico emplear mensajes cortos para representar el suceso x_i y largos para los x_j restantes, ya que el valor que más nos va a aparecer en una secuencia de sucesos es precisamente x_i. Volveremos sobre esto un poco más adelante.

Veamos unos cuantos ejemplos más:

- La entropía de la variable aleatoria asociada a lanzar una moneda es la siguiente:

$$H(M) = -(0.5\log_2(0.5) + 0.5\log_2(0.5)) = 1$$

  Este suceso aporta exactamente una unidad de información.

- Si la moneda está trucada (60% de probabilidades para cara, 40% para cruz), nos sale:

$$H(M) = -(0.6\log_2(0.6) + 0.4\log_2(0.4)) = 0.970$$

- Veamos el ejemplo de las bolas (nueve negras y una blanca):

$$H(M) = -(0.9\log_2(0.9) + 0.1\log_2(0.1)) = 0.468$$

La unidad de cantidad de información que vamos a emplear es la cantidad de información que aporta el suceso más simple (dos posibilidades equiprobables, como el caso de la moneda sin trucar), y a esa unidad se le denomina *bit*. Esta es precisamente la razón por la que empleamos logaritmos base 2, para que la cantidad de información del suceso más simple sea igual a la unidad.

Podemos decir que la entropía de un suceso es el número medio de bits que necesitaremos para codificar cada uno de los estados de la variable (codificamos cada suceso empleando un mensaje escrito en un alfabeto binario). Supongamos que queremos codificar los diez dígitos decimales usando secuencias de bits. Con tres bits no tenemos suficiente, así que necesitaremos más, pero ¿cuántos más? Si usamos cuatro bits para representar todos los dígitos tal vez nos estemos pasando... Veamos cuánta entropía tienen diez sucesos equiprobables:

$$H = -\sum_{i=1}^{10} \frac{1}{10}\log_2\left(\frac{1}{10}\right) = -\log_2\left(\frac{1}{10}\right) = 3.32 bits$$

El valor que acabamos de calcular es el límite teórico, que normalmente no se puede alcanzar. Lo único que podemos decir es que no existe ninguna codificación que emplee longitudes promedio de mensaje inferiores al número que acabamos de calcular. Veamos la siguiente codificación: 000 para 0, 001 para 1, 010 para 2, 011 para 3, 100 para 4, 101 para

5 ,1100 para 6, 1101 para 7, 1110 para 8, y 1111 para 9. Con esta codificación empleamos, como media:

$$\frac{3 \cdot 6 + 4 \cdot 4}{10} = 3.4 bits$$

para codificar cada mensaje.

3.3 Entropía Condicionada

Supongamos que tenemos ahora una variable aleatoria bidimensional (X, Y). Recordemos las distribuciones de probabilidad más usuales que podemos definir sobre dicha variable, teniendo n posibles casos para X y m para Y:

1. Distribución conjunta de (X, Y):
$$P(x_i, y_j)$$
2. Distribuciones marginales de X e Y:
$$P(x_i) = \sum_{j=1}^{m} P(x_i, y_j) \qquad P(y_j) = \sum_{i=1}^{n} P(x_i, y_j)$$
3. Distribuciones condicionales de X sobre Y y viceversa:
$$P(x_i/y_j) = \frac{P(x_i, y_j)}{P(y_j)} \qquad P(y_j/x_i) = \frac{P(x_i, y_j)}{P(x_j)}$$

Definiremos la entropía de las distribuciones que acabamos de referir:

$$H(X, Y) = -\sum_{i=1}^{n} \sum_{j=1}^{m} P(x_i, y_j) \log_2 (P(x_i, y_j))$$

$$H(X/Y = y_j) = -\sum_{i=1}^{n} P(x_i/y_j) \log_2 (P(x_i/y_j))$$

Haciendo la suma ponderada de los $H(X/Y = y_j)$ obtemos la expresión de la *Entropía Condicionada* de X sobre Y:

$$\begin{aligned} H(X/Y) &= -\sum_{i=1}^{n} \sum_{j=1}^{m} P(y_j) P(x_i/y_j) \log_2(P(x_i/y_j)) = \\ &= -\sum_{i=1}^{n} \sum_{j=1}^{m} P(x_i, y_j) \log_2(P(x_i/y_j)) \end{aligned} \tag{3.3}$$

Así como existe una *Ley de la Probabilidad Total*, análogamente se define la *Ley de Entropías Totales*:

$$H(X,Y) = H(X) + H(Y/X) \tag{3.4}$$

cumpliéndose además, si X e Y son variables independientes:

$$H(X,Y) = H(X) + H(Y) \tag{3.5}$$

Teorema de Disminución de la Entropía: La entropía de una variable X condicionada por otra Y es menor o igual a la entropía de X, alcanzándose la igualdad si y sólo si las variables X e Y son independientes.

3.4 Cantidad de Información entre dos Variables

Shannon propuso una medida para la cantidad de información que aporta sobre una variable el conocimiento de otra. Se definirá, pues, la *cantidad de información de Shannon que la variable X contiene sobre Y* como:

$$I(X,Y) = H(Y) - H(Y/X) \tag{3.6}$$

Esto quiere decir que la cantidad de información que nos aporta el hecho de conocer X al medir la incertidumbre sobre Y es igual a la disminución de entropía que este conocimiento conlleva. Sus propiedades son:

i. $I(X,Y) = I(Y,X)$

ii. $I(X,Y) \geq 0$

3.5 Criptosistema Seguro de Shannon

Diremos que un criptosistema es *seguro* si la cantidad de información que nos aporta el hecho de conocer el mensaje cifrado c sobre la entropía del texto plano m vale cero. Es decir:

$$I(C,M) = 0 \tag{3.7}$$

Esto significa sencillamente que la distribución de probabilidad que nos inducen todos los posibles mensajes no cifrados no cambia si conocemos el mensaje cifrado. Para entenderlo mejor supongamos que sí se modifica dicha distribución: El hecho de conocer un mensaje cifrado, al variar la distribución de probabilidad sobre M haría unos mensajes más probables

que otros, y por consiguiente unas claves de cifrado (aquellas que me permitan llegar de los m más probables al c concreto que tenga en cada momento) más probables que otras. Repitiendo esta operación muchas veces con mensajes diferentes, cifrados con la misma clave, podríamos ir modificando la distribución de probabilidad sobre la clave empleada hasta obtener un valor de clave mucho más probable que los otros, permitiéndonos romper el criptosistema.

Si por el contrario el sistema cumpliera la condición 3.7, jamás podríamos romper el sistema, ni siquiera empleando una máquina con capacidad de proceso infinita. Por ello los criptosistemas que cumplen la condición de Shannon se denominan también criptosistemas ideales.

Se puede demostrar también que para que un sistema sea criptoseguro según el criterio de Shannon, la cardinalidad del espacio de claves ha de ser al menos igual que la del espacio de mensajes. En otras palabras, que la clave ha de ser al menos tan larga como el mensaje que queramos cifrar. Esto vuelve inútiles a estos criptosistemas en la práctica.

3.6 Redundancia

Si una persona lee un mensaje en el que faltan algunas letras, normalmente puede reconstruirlo. Esto ocurre porque casi todos los símbolos de un mensaje en lenguajen natural contienen información que se puede extraer de los símbolos de alrededor (información que, en la práctica, se está enviando *dos veces*). En otras palabras, porque el lenguaje natural es *redundante*. Puesto que tenemos mecanismos para definir la cantidad de información que presenta un suceso, podemos intentar medir el exceso de información (redundancia) de un lenguaje. Para ello vamos a dar una serie de definiciones:

- *Indice de un lenguaje.* Definiremos el índice de un lenguaje para mensajes de longitud k como:

$$r_k = \frac{H_k(M)}{k} \tag{3.8}$$

  siendo $H_k(X)$ la entropía de todos los posibles mensajes de longitud k. Estamos midiendo el número de bits de información que nos aporta cada carácter en mensajes de una longitud determinada. Para idiomas como el Inglés, r_k suele valer alrededor de $1.3 bits/letra$ para valores pequeños de k.

- *Indice absoluto de un lenguaje.* Es el máximo número de bits de información que pueden ser codificados en cada carácter, asumiendo que todas las combinaciones de caracteres son igualmente probables. Suponiendo m letras diferentes en nuestro alfabeto (27 en el caso del Español), este índice vale:

$$R = \log_2(m)$$

  En el caso del Español podríamos codificar $4.7 bits$ por letra aproximadamente, luego parece que el nivel de redundancia (asumiendo que su índice r sea parecido al del Inglés) es alto.

- Finalmente, la *redundancia* de un lenguaje se define como la diferencia entre las dos magnitudes anteriores:

$$D = R - r$$

También se define el *índice de redundancia* como el siguiente cociente:

$$I = \frac{D}{R}$$

Desgraciadamente, para medir la auténtica redundancia de un lenguaje, hemos de tener en cuenta secuencias de cualquier número de caracteres, por lo que la expresión 3.8 debería calcularse en realidad como:

$$r_\infty = \lim_{n\to\infty} \frac{H_n(M)}{n} \tag{3.9}$$

Precisamente una de las aplicaciones de la Teoría de la Información es la compresión de datos, que simplemente trata de eliminar la redundancia dentro de un archivo (considerando cada byte como un mensaje elemental, y codificándolo con más o menos bits según su frecuencia de aparición).

Otra de las aplicaciones directas de la Teoría de la Información son los Códigos de Redundancia Cíclica (CRC), que permiten introducir un campo de longitud mínima en el mensaje, tal que éste proporcione la mayor redundancia posible. Así, si el mensaje original resultase alterado, la probabilidad de que el CRC añadido siga siendo correcto es mínima.

Nótese que, conocidos los patrones de redundancia de un lenguaje, es posible dar de forma automática una estimación de si una cadena de símbolos corresponde o no a dicho lenguaje. Esta característica es aprovechada para efectuar ataques por la fuerza bruta, ya que ha de asignarse una probabilidad a cada clave individual en función de las características del mensaje obtenido al decodificar el criptograma con dicha clave. El número de claves suele ser tan elevado que resulta imposible una inspección visual. Una estrategia bastante interesante para protegerse contra este tipo de ataques consistirá en comprimir los mensajes antes de codificarlos. De esa manera eliminamos la redundancia y hacemos más difícil a un atacante apoyarse en las características del mensaje original para recuperar la clave.

3.7 Desinformación y Distancia de Unicidad

Definiremos *desinformación* de un sistema criptográfico como la entropía condicionada de un mensaje m dado un texto cifrado c:

$$H_c(m) = \sum_c P(c) \left(-\sum_m P_c(m) log_2(P_c(m)) \right) \tag{3.10}$$

Esta expresión nos dice la incertidumbre que nos queda sobre m si conocemos c. Si esa incertidumbre fuera la misma que desconociendo c, nos encontraríamos frente a un criptosistema seguro de Shannon. Lo habitual es que $H_c(m) < H(m)$.

Adicionalmente, si el valor de $H_c(m)$ fuera muy pequeño con respecto a $H(m)$, significaría que el hecho de conocer c nos aporta mucha información sobre m, lo cual quiere decir que nuestro criptosistema es inseguro. El peor de los casos sería que $H_c(m) = 0$, puesto que entonces, conociendo c tendríamos albsoluta certeza sobre m.

Esta magnitud se puede medir también en función de la clave k, y entonces nos dirá la incertidumbre que nos queda sobre k conocida c:

$$H_c(k) = \sum_c P(c) \left(- \sum_m P_c(k) log_2(P_c(k)) \right) \tag{3.11}$$

Definiremos finalmente la *distancia de unicidad* de un criptosistema como la longitud mínima de mensaje cifrado que aproxima el valor $H_c(k)$ a cero. En otras palabras, es la cantidad de texto cifrado que necesitamos para poder descubrir la clave. Los criptosistemas seguros de Shannon tienen distancia de unicidad infinita. Nuestro objetivo a la hora de diseñar un sistema criptográfico será que la distancia de unicidad sea lo más grande posible.

3.8 Confusión y Difusión

Según la Teoría de Shannon, las dos técnicas básicas para ocultar la redundancia en un texto plano son la *confusión* y la *difusión*. Estos conceptos, a pesar de su antiguedad, poseen una importancia clave en Criptografía moderna.

- *Confusión.* Trata de ocultar la relación entre el texto plano y el texto cifrado. Recordemos que esa relación existe y se da a partir de la clave k empleada, puesto que si no existiera jamás podríamos descifrar los mensajes. El mecanismo más simple de confusión es la sustitución, que consiste en cambiar cada ocurrencia de un símbolo en el texto plano por otro. La sustitución puede ser tan simple o tan compleja como queramos.

- *Difusión.* Diluye la redundancia del texto plano *repartiéndola* a lo largo de todo el texto cifrado. El mecanismo más elemental para llevar a cabo una difusión es la transposición (que consiste en cambiar de sitio elementos individuales del texto plano).

3.9 Ejercicios Propuestos

1. Calcule la cantidad de información que tiene el hecho de que en un dado no cargado salga un número par.

2. Calcule la entropía que tiene un dado que presenta doble probabilidad para el número tres que para el resto.

3. Demuestre que la entropía está acotada superiormente por $\log_2(N)$.

4. Demuestre la Ley de Entropías Totales.

Capítulo 4

Fundamentos de Aritmética Modular

4.1 Aritmética Modular. Propiedades

Dados tres números $a, b, n \in \mathbb{N}$, decimos que *a es congruente con b módulo n*, y se escribe:

$$a \equiv b (\text{mod } n)$$

si se cumple:

$$a = b + kn \qquad k \in \mathbb{Z}$$

El número natural n inducirá n clases de equivalencia (números congruentes con 0, números congruentes con 1, ..., números congruentes con $n-1$). Podemos definir ahora las operaciones suma y producto en ese conjunto de clases de equivalencia:

- $a + b \equiv c (\text{mod} n) \iff a + b = c + kn \qquad k \in \mathbb{Z}$
- $ab \equiv c (\text{mod} n) \iff ab = c + kn \qquad k \in \mathbb{Z}$

La operación suma en este conjunto cumple las propiedades asociativa y conmutativa y posee elementos neutro y simétrico, por lo que el conjunto tendrá estructura de grupo conmutativo. A partir de ahora llamaremos grupo finito inducido por n a dicho conjunto.

Con la operación producto se cumplen las propiedades asociativa y conmutativa, y tiene elemento neutro, pero no necesariamente simétrico. La estructura del conjunto con las operaciones suma y producto es, pues, de anillo conmutativo. Más adelante veremos bajo qué condiciones existe el elemento simétrico para el producto.

4.1.1 Algoritmo de Euclides

Quizá sea el algoritmo más antiguo que se conoce, y a la vez es uno de los más útiles. Permite calcular el máximo común divisor de dos números.

Sean a y b dos números enteros de los que queremos calcular su máximo común divisor m. El Algoritmo de Euclides explota la siguiente propiedad:

$$m|a \wedge m|b \Longrightarrow m|(a - kb) \text{ con } k \in \mathbb{Z} \Longrightarrow m|(a \bmod b)$$

$a|b$ quiere decir que a *divide* a b, o en otras palabras, que b es múltiplo de a.

Si llamamos c a $(a \bmod b)$, podemos aplicar de nuevo la propiedad y tenemos:

$$m|(b \bmod c)$$

Sabemos, pues, que m tiene que dividir a todos los restos que vayamos obteniendo. Es evidente que el último resto será cero, puesto que los restos siempre son inferiores al divisor. El penúltimo valor obtenido es el mayor número que divide tanto a a como a b. El algoritmo queda entonces como sigue:

```
int euclides(int a, int b)
  { int i;

    g[0]=a;
    g[1]=b;
    i=1;
    while (g[i]!=0)
      { g[i+1]=g[i-1]%g[i];
        i++;
      }
    return(g[i-1]);
  }
```

El invariante del algoritmo es el siguiente:

$$g_{i+1} = g_{i-1} (\bmod\ g_i)$$

4.2 Cálculo de Inversas en Aritmética Modular

4.2.1 Existencia de la Inversa

Hemos comentado en la sección 4.1 que los elementos de un grupo finito no tienen por qué tener inversa (elemento simétrico para el producto). En este apartado veremos qué condiciones han de cumplirse para que exista la inversa de un número dentro de un grupo finito.

Lema: Dados a,$n \in \mathbb{N}$

$$\text{mcd}(a,n) = 1 \Longrightarrow ai \neq aj (\text{mod } n) \qquad \forall\, i \neq j \quad 0 < i,j < n \tag{4.1}$$

Demostración: Supongamos que $\text{mcd}(a,n) = 1$, y que existen $i \neq j$ tales que $ai \equiv aj (\text{mod } n)$. Se cumple, pues:

$$(ai - aj)|n \Longrightarrow a(i-j)|n$$

puesto que a y n son primos entre sí, a no puede dividir a n, luego

$$(i-j)|n \Longrightarrow i \equiv j (\text{mod } n)$$

con lo que hemos alcanzado una contradicción.

Ahora podemos hacer la siguiente reflexión: Si $ai \neq aj$ para cualesquiera $i \neq j$, multiplicar a por todos los elementos del grupo finito módulo n nos producirá una permutación de los elementos del grupo (exceptuando el cero), por lo que forzosamente ha de existir un valor tal que al multiplicarlo por a nos dé 1. Eso nos conduce al siguiente teorema:

Teorema: Si $\text{mcd}(a,n) = 1$, a tiene inversa módulo n.

Corolario: Si n es primo, el grupo finito que genera tiene estructura de cuerpo (todos sus elementos tienen inversa para el producto excepto el cero).

4.2.2 Función de Euler

Llamaremos *conjunto reducido de residuos módulo n* al conjunto de números primos relativos con n. En otras palabras, todos los números que tienen inversa módulo n. Por ejemplo, si n fuera 12, su conjunto reducido de residuos sería:

$$\{1, 5, 7, 11\}$$

El cardinal del conjunto reducido de residuos módulo n es igual a:

$$\Phi(n) = \prod_{i=1}^{n} p_i^{e_i - 1}(p_i - 1) \tag{4.2}$$

siendo p_i los factores primos de n y e_i su multiplicidad. Por ejemplo, si n fuera el producto de dos números primos p y q, $\Phi(n) = (p-1)(q-1)$.

$\Phi(n)$ se conoce como la *función de Euler sobre n.*

Teorema: Si $\mathrm{mcd}(a,n) = 1$:

$$a^{\Phi(n)} \equiv 1 (\mathrm{mod}\ n) \tag{4.3}$$

Demostración: Puesto que a y n son primos entre sí, a multiplicado por cualquier elemento del conjunto reducido de residuos módulo n $\{r_1, \ldots, r_{\Phi(n)}\}$ ha de ser también primo con n, por lo tanto el conjunto $\{ar_1, \ldots, ar_{\Phi(n)}\}$ no es más que una permutación del conjunto anterior, lo cual nos lleva a:

$$\prod_{i=1}^{\Phi(n)} r_i = \prod_{i=1}^{\Phi(n)} ar_i = a^{\Phi(n)} \prod_{i=1}^{\Phi(n)} r_i \Longrightarrow a^{\Phi(n)} \equiv 1 (\mathrm{mod}\ n)$$

Teorema de Fermat: Si p es primo, entonces

$$a^{p-1} \equiv 1 (\mathrm{mod}\ n) \tag{4.4}$$

Podemos emplear, pues, la función de Euler para calcular inversas módulo n, puesto que:

$$a^{\Phi(n)} = aa^{\Phi(n)-1} \equiv 1 (\mathrm{mod}\ n) \Longrightarrow a^{-1} \equiv a^{\Phi(n)-1} (\mathrm{mod}\ n)$$

4.2.3 Algoritmo Extendido de Euclides

El Algoritmo Extendido de Euclides simplemente tiene en cuenta los cocientes en cada paso además de los restos. El invariante que mantiene es el siguiente, suponiendo que se le pasen como parámetros n y a:

$$g_i = nu_i + av_i$$

El último valor de g_i será el máximo común divisor, que será 1 si a y n son primos relativos, por lo que tendremos:

$$1 = nu_i + av_i$$

$(v_i \mod n)$ será entonces la inversa de a módulo n.

Nuestra segunda alternativa para calcular inversas, cuando desconozcamos $\Phi(n)$, será pues el Algoritmo Extendido de Euclides:

```
int euclides_ext(int n, int a)
  { int c,i;

    g[0]=n; g[1]=a;
    u[0]=1; u[1]=0;
    v[0]=0; v[1]=1;
```

```
    i=1;
    while (g[i]!=0)
      { c=g[i-1]/g[i];
        g[i+1]=g[i-1]%g[i];
        u[i+1]=u[i-1]-c*u[i];
        v[i+1]=v[i-1]-c*v[i];
        i++;
      }
    return(v[i-1]%n);
  }
```

4.3 Exponenciación. Logaritmos Discretos

Muchos de los algoritmos de llave pública emplean exponenciaciones dentro de grupos finitos para codificar los mensajes. Tanto las bases como los exponentes en esos casos son números astronómicos, de incluso miles de bits de longitud. Efectuar las exponenciaciones mediante multiplicaciones reiterativas de la base sería inviable. En esta sección veremos mecanismos eficientes para llevar a cabo estas operaciones. También comentaremos brevemente el problema inverso, el cálculo de los logaritmos discretos, puesto que en él basan muchos algoritmos su resistencia.

4.3.1 Algoritmo de Exponenciación Rápida

Supongamos que tenemos dos números naturales a y b, y queremos calcular a^b. El mecanismo más sencillo sería multiplicar a por sí mismo b veces. Sin embargo, para valores muy grandes de b este algoritmo no nos sirve.

Tomemos la representación binaria de b:

$$b = 2^0 b_0 + 2^1 b_1 + 2^2 b_2 + \ldots + 2^n b_n$$

Expresemos la potencia que vamos a calcular en función de dicha representación:

$$a^b = a^{2^0 b_0 + 2^1 b_1 + 2^2 b_2 + \ldots + 2^n b_n} = \prod_{i=0}^{n} a^{2^i b_i}$$

recordemos que los b_i sólo pueden valer 0 ó 1, por tanto para calcular a^b sólo hemos de multiplicar los a^{2^i} correspondientes a los dígitos binarios de b que valgan 1.

Nótese, además, que $a^{2^i} = (a^{2^{i-1}})^2$, por lo que, partiendo de a, podemos calcular el siguiente valor de esta serie elevando al cuadrado el anterior. El Algoritmo de Exponenciación Rápida queda como sigue:

```
int exp_rapida(int a, int b)
    { int z,x,resul;

      z=b;
      x=a;
      resul=1;
      while (z>0)
        { if (z%1==1)
            resul=resul*x;
          x=x*x;
          z=z/2;
        }
      return(resul);
    }
```

La variable `z` se inicializa con el valor de b y se va dividiendo por 2 en cada paso para tener siempre el i-ésimo bit de b en el menos significativo de `z`. En la variable `x` se almacenan los valores de a^{2^i}.

4.3.2 El Problema de los Logaritmos Discretos

El problema inverso la exponenciación es el cálculo de logaritmos discretos. Dados dos números a, b y el módulo n, se define el logaritmo discreto de a en base b módulo n como:

$$c = \log_b(a) \pmod{n} \iff a \equiv b^c \pmod{n} \tag{4.5}$$

En la actualidad no existen algoritmos eficientes que sean capaces de calcular en tiempo razonable logaritmos de esta naturaleza, y muchos esquemas criptográficos basan su resistencia en esta circunstancia. El problema de los logaritmos discretos está íntimamente relacionado con el de la factorización, de hecho está demostrado que si se puede calcular un logaritmo, entonces se puede factorizar fácilmente (el recíproco no se ha podido demostrar).

4.4 Factorización. Tests de Primalidad

Para explotar la dificultad de cálculo de logaritmos discretos, muchos algoritmos criptográficos de llave pública se basan en operaciones de exponenciación en grupos finitos. Dichos conjuntos deben cumplir la propiedad de que su módulo n sea un número muy grande con pocos factores (usualmente dos). Estos algoritmos funcionan si se conoce n y sus factores se mantienen en secreto. Habitualmente para obtener n se calculan primero dos números primos

muy grandes, que posteriormente se multiplican. Necesitaremos pues mecanismos para poder calcular esos números primos *grandes*.

La factorización es el problema inverso a la multiplicación: dado n, se trata de buscar un conjunto de números tales que su producto valga n. Normalmente, y para que la solución sea única, se impone la condición de que los factores de n que obtengamos sean todos primos. Al igual que para el problema de los logaritmos discretos, no existen algoritmos eficientes para efectuar este tipo de cálculos, siempre y cuando los factores (como veremos más adelante) hayan sido escogidos correctamente. Esto nos permite confiar en que los factores de n serán imposibles de calcular aunque se conozca el propio n.

En cuanto al cálculo de primos grandes, bastaría con aplicar un algoritmo de factorización para saber si un número es primo o no. Este mecanismo es inviable, puesto que acabamos de decir que no hay algoritmos eficientes de factorización. Por suerte, sí que existen algoritmos probabilísticos que permiten decir con un grado de certeza bastante elevado si un número cualquiera es primo o compuesto.

Cabría preguntarse, dado que para los algoritmos asimétricos de cifrado necesitaremos generar muchos números primos, si realmente hay *suficientes*. De hecho se puede pensar que, a fuerza de generar números, llegará un momento en el que repitamos un primo generado con anterioridad. Podemos estar tranquilos, porque si a cada átomo del universo le asignáramos mil millones de números primos cada microsegundo desde su origen, harían falta un total de 10^{109} números primos diferentes, mientras que el total estimado de números primos de 512 bits o menos es aproximadamente de 10^{151}.

También podríamos pensar en calcular indiscriminadamente números primos para poder emplearlos en algún algoritmo de factorización rápida. Por desgracia, si en un disco duro que pesara un gramo por cada GByte de capacidad quisiéramos almacenar todos los primos de 512 bits o menos, ese disco duro pesaría tanto que colapsaría en un agujero negro.

En los próximos apartados vamos a ver algunos de los métodos más comunes para generar números primos.

4.4.1 Método de Lehmann

Es uno de los tests más sencillos para saber si un número p es o no primo:

1. Escoger un número aleatorio $a < p$.
2. Calcular $b = a^{(p-1)/2} (\text{mod } p)$.
3. Si $b \neq 1 (\text{mod } p)$ y $b \neq -1 (\text{mod } p)$, p no es primo.
4. Si $b \equiv 1 (\text{mod } p)$ ó $b \equiv -1 (\text{mod } p)$, la probabilidad de que p sea primo es igual o superior al 50%.

Repitiendo el algoritmo n veces, la probabilidad de que p supere el test y sea compuesto (es decir, no primo) será de 1 contra 2^n.

4.4.2 Método de Rabin-Miller

Es el algoritmo más empleado, debido a su facilidad de implementación. Sea p el número que queremos saber si es primo. Se calcula b, siendo b el número de veces que 2 divide a $(p-1)$, es decir, 2^b es la mayor potencia de 2 que divide a $(p-1)$. Calculamos entonces m, tal que $p = 1 + 2^b * m$.

1. Escoger un número aleatorio $a < p$.
2. Sea $j = 0$ y $z = a^m (\text{mod } p)$.
3. Si $z = 1$, o $z = p - 1$, entonces p pasa el test y puede ser primo.
4. Si $j > 0$ y $z = 1$, p no es primo.
5. Sea $j = j + 1$. Si $j = b$ y $z \neq p - 1$, p no es primo.
6. Si $j < b$ y $z \neq p - 1$, $z = z^2 (\text{mod } p)$. Volver al paso (4).
7. Si $j < b$ y $z = p - 1$, entonces p pasa el test y puede ser primo.
8. p no es primo.

La probabilidad de que un número compuesto pase este algoritmo para un número a es del 25%. Esto quiere decir que necesitaremos menos pasos para llegar al mismo nivel de confianza que el obtenido con el algoritmo de Lehmann.

4.4.3 Consideraciones Prácticas

A efectos prácticos el algoritmo que se suele emplear para saber si un número es o no primo es el siguiente:

1. Generar un número aleatorio p de n bits.
2. Poner a uno el bit más significativo (garantizamos que el número es de n bits) y el menos significativo (debe ser impar para poder ser primo).
3. Intentar dividir p por una tabla de primos precalculados (usualmente se usan los primos menores que 2000). Esto elimina gran cantidad de números no primos de una forma muy rápida. Baste decir a título informativo que más del 99.8% de los números impares no primos es divisible por algún número primo menor que 2000.
4. Ejecutar el test de Rabin-Miller como mínimo cinco veces.

4.4.4 Primos *fuertes*

Aunque p y q sean primos grandes, existen algunos casos en los que es relativamente fácil factorizar el número $n = pq$. Se proponen entonces una serie de condiciones para p y q que dificultan la factorización de n. Se dice que p y q son *números primos fuertes* si cumplen:

- El máximo común divisor de $p-1$ y $q-1$ debe ser pequeño.
- $p-1$ y $q-1$ deben tener algún factor primo grande p' y q'.
- Tanto $p'-1$ como $q'-1$ deben tener factores primos grandes.
- Tanto $p'+1$ como $q'+1$ deben tener factores primos grandes.

Las dos primeras condiciones se cumplen si tanto $(p-1)/2$ como $(q-1)/2$ son números primos.

4.5 Ejercicios Propuestos

1. Comprobar las propiedades de la suma en grupos finitos.
2. Comprobar las propiedades del producto en grupos finitos.

Capítulo 5

Aritmética Entera de Múltiple Precisión

En este capítulo daremos una serie de nociones básicas y algoritmos sobre aritmética entera de múltiple precisión, disciplina que ha cobrado un gran interés debido al uso extensivo que hacen de ella sobre todo los algoritmos asimétricos de cifrado y autentificación.

5.1 Representación de enteros largos

Un número *largo* es un número que posee muchos dígitos, normalmente más de los que los tipos de dato convencionales de los lenguajes de programación clásicos pueden soportar. En este apartado vamos a indicar cómo representarlos usando tipos de dato de menor precisión.

Todos conocemos la representación tradicional en base 10 de los números, en la que cada cifra contiene únicamente valores de 0 a 9. Esta representación no es más que un caso particular ($B = 10$) de la siguiente expresión general:

$$n = (-)\sum_{-\infty}^{\infty} a_i B^i$$

donde los términos con índice negativo corresponden a la parte no entera (*decimal*) del n. Sabemos que dicha representación es única, y que significa que n en base B se escribe:

$$(-)a_n a_{n-1} \ldots a_0 . a_{-1} \ldots$$

En cualquier caso, puesto que nuestro objetivo es representar únicamente números enteros positivos, prescindiremos del signo y de los términos con subíndice negativo.

Cualquier número vendrá representado por una serie única de coeficientes a_i (cifras), de las

que importa tanto su valor como su posición dentro del número. Esta estructura corresponde claramente a la de un *array*. Para representar de forma eficiente enteros largos emplearemos una base que sea potencia de dos (normalmente se escoge $B = 2^{16}$ ó $B = 2^{32}$ para que cada *cifra* de nuestro número se pueda almacenar en un dato del tipo `unsigned int` sin desperdiciar ningún bit). Para almacenar los resultados parciales de las operaciones aritméticas emplearemos un tipo de dato de doble precisión (`unsigned long int`, correspondiente a $B = 2^{32}$ ó $B = 2^{64}$) de forma que no se nos desborde al multiplicar dos cifras. Normalmente se escoge una longitud que pueda manejar directamente la ALU de la computadora, para que las operaciones elementales entre cifras sean rápidas.

Por todo esto, para nosotros un número entero largo será un *array* (dinámico o no) de `unsigned int`. En cualquier caso, y a partir de ahora, nuestro objetivo será estudiar algoritmos eficientes para efectuar operaciones aritméticas sobre este tipo de números, independientemente de la base en la que se encuentren representados.

5.2 Operaciones aritméticas sobre enteros largos

Vamos a describir en este apartado cómo realizar operaciones aritméticas (suma, resta, multiplicación y división) de enteros largos.

5.2.1 Suma

La suma de $a = (a_0, a_1 \dots a_{n-1})$ y $b = (b_0, b_1 \dots b_{n-1})$ se puede definir como:

$$(a+b)_i = \begin{cases} (a_i + b_i + c_i) \bmod B & \text{para } i = 0 \dots n-1 \\ c_i & \text{para } i = n \end{cases}$$

siendo

$$c_i = \begin{cases} 0 & \text{para } i = 0 \\ (a_{i-1} + b_{i-1} + c_{i-1}) \text{ div } B & \text{para } i = 1 \dots n \end{cases}$$

c_i es el *acarreo* de la suma de los dígitos inmediatamente anteriores. Tenemos en cuenta el coeficiente n de la suma porque puede haber desbordamiento, en cuyo caso la suma tendría $n+1$ dígitos y su cifra más significativa sería precisamente c_n.

El algoritmo para la suma quedaría, pues, como sigue:

```
suma (unsigned *a, unsigned *b, unsigned *s)
    { unsigned long sum;
      unsigned acarreo;
```

```
    n=max(num. de digitos de a, num. de digitos de b)
    acarreo=0;
    for (i=0;i<n;i++)
      {  sum=acarreo+a[i]+b[i];
         s[i]=sum%BASE;
         acarreo=sum/BASE;
      }
    s[n]=acarreo;
  }
```

El resultado se devuelve en `s`.

5.2.2 Resta

La resta es muy parecida a la suma, salvo que en este caso los acarreos se restan. Suponiendo que $a > b$:

$$(a-b)_i = (a_i - b_i - r_i) \bmod B \qquad \text{para } i = 0 \ldots n-1$$

siendo

$$r_i = \begin{cases} 0 & \text{para } i = 0 \\ 1 - \Big((a_{i-1} - b_{i-1} - r_{i-1}) \text{ div } B \Big) & \text{para } i = 1 \ldots n \end{cases}$$

r_i representa el acarreo de la resta (*borrow*), que puede valer 0 o 1 según la resta parcial salga positiva o negativa. Nótese que, como $a > b$, el último acarreo siempre ha de valer 0.

```
resta (unsigned *a, unsigned *b, unsigned *d)
    { unsigned long dif;
      unsigned acarreo;

      n=max(num. de digitos de a, num. de digitos de b)
      acarreo=0;
      for (i=0;i<n;i++)
        {  dif=a[i]-b[i]-acarreo+BASE;
           d[i]=dif%BASE;
           acarreo=1-dif/BASE;
        }
    }
```

El resultado se devuelve en `d`. La razón por la que sumamos la base a `dif` es para que la resta parcial salga siempre positiva y poder hacer el módulo correctamente. En ese caso, si el valor era positivo, al sumarle B y dividir por B de nuevo nos queda 1. Si fuera negativo, nos saldría 0. Por eso asignamos al nuevo acarreo el valor `1-dif/BASE`.

Nos queda comprobar cuál de los dos números es mayor para poder emplearlo como minuendo. Esta comprobación se puede realizar fácilmente definiendo una función que devuelva el número cuyo dígito más significativo tenga un número de orden mayor. En caso de igualdad iríamos comparando dígito a dígito, empezando por los más significativos hasta que encontremos alguno mayor o lleguemos al último dígito, situación que únicamente ocurrirá si los dos números son iguales.

5.2.3 Multiplicación

Para obtener el algoritmo del producto emplearemos la expresión general de un número entero positivo en base B. Si desarrollamos el producto de dos números cualesquiera a y b de longitud m y n respectivamente nos queda:

$$ab = \sum_{i=0}^{m-1} a_i B^i b$$

A la vista de esto podemos descomponer el producto como m llamadas a una función que multiplica un entero largo por $a_i B^i$ (es decir, un entero largo con un único dígito significativo) y despues sumar todos los resultados parciales.

Para poder implementar esto primero definiremos una función (`summult`) que multiplique b por $a_i B^i$ y el resultado se lo sume al vector s, que no tiene necesariamente que estar a cero:

```
summult(unsigned ai, int i, unsigned *b, int n, unsigned *s)
      { int k;
        unsigned acarreo;
        unsigned long prod,sum;

        acarreo=0;
        for (k=0; k<n; k++)
          { prod=ai*b[k]+s[i+k]+acarreo;
            s[i+k]=prod%BASE;
            acarreo=prod/BASE;
          }
        k=n+i;
        while (acarreo!=0)
          { sum=s[k]+acarreo;
            s[k]=sum%BASE;
```

```
            acarreo=sum/BASE;
          }
      }
```

La segunda parte de la función se encarga de acumular los posibles acarreos en el vector s. A partir de la función que acabamos de definir, el producto queda entonces como sigue:

```
producto(unsigned *a,int m, unsigned *b, int n, unsigned *p)
   { int k;

     for (k=0;k<=m+n;k++)
       p[k]=0;
     for (k=0;k<m;k++)
       summult(a[k],k,b,n,p);
   }
```

El resultado se devuelve en `p`.

Existe otra propiedad de la multiplicación de enteros que nos va a permitir efectuar estas operaciones de manera más eficiente. Tomemos un número entero cualquiera a de k dígitos en base B. Dicho número puede ser representado mediante la de la siguiente expresión:

$$a = a_l B^{\frac{k}{2}} + a_r$$

Es decir, *partimos* a en dos mitades. Por razones de comodidad, llamaremos B_k a $B^{\frac{k}{2}}$. Veamos ahora cómo queda el producto de dos números cualesquiera a y b, en función de sus respectivas *mitades*:

$$ab = a_l b_l B_k^2 + (a_l b_r + a_r b_l) B_k + a_r b_r$$

Hasta ahora no hemos aportado nada nuevo. El *truco* para que este desarrollo nos proporcione un aumento de eficiencia consiste en hacer uso de la siguiente propiedad:

$$a_l b_r + a_r b_l = a_l b_r + a_r b_l + a_l b_l - a_l b_l + a_r b_r - a_r b_r = (a_l + a_r)(b_l + b_r) - a_l b_l - a_r b_r$$

quedando finalmente, lo siguiente:

$$x = a_l b_l \qquad y = (a_l + a_r)(b_l + b_r) \qquad z = a_r b_r$$
$$ab = x B_k^2 + (y - x - z) B_k + z$$

Hemos reducido los cuatro productos y tres sumas del principio a tres productos y seis sumas. Como es lógico, esta técnica debe emplearse dentro de una estrategia *divide y vencerás.*

5.2.4 División

Para calcular cocientes nos basaremos en el algoritmo tradicional de la división. Suponiendo que queremos dividir c por d, obteniendo su cociente (a) y resto (b), iremos calculando los dígitos del cociente uno a uno. Nuestro objetivo será estimar a la baja el valor de cada uno de los dígitos a, e incrementarlos hasta alcanzar el valor correcto. Para que la estimación se quede lo más cerca posible del valor correcto efectuaremos una normalización de los números, de forma que el dígito más significativo d tenga su bit de mayor peso a 1. Esto se consigue multiplicando c y d por 2^k, siendo k el número de ceros a la izquierda del bit más significativo del divisor d. Posteriormente habremos de tener en cuenta lo siguiente:

$$c = ad + b \Longleftrightarrow 2^k c = a(2^k d) + 2^k b$$

luego el cociente será el mismo pero el resto habrá que dividirlo por el factor de normalización. Llamaremos $\bar{c}, \bar{d}$ a los valores de c y d normalizados.

Para hacer la estimación a la baja de los a_i, dividiremos $c_j B + c_{j-1}$ por $\bar{d}_m + 1$ (c_j es el dígito más significativo de c en el paso i, y $\bar{d}_m$ es el dígito más significativo de $\bar{d}$). Luego actualizamos $\bar{c}$ con $\bar{c} - \bar{d} a_i B^i$ y vamos incrementando a_i (y actualizando $\bar{c}$) mientras nos quedemos cortos. Finalmente, habremos calculado el valor del cociente (a) y el valor del resto será

$$b = \frac{\bar{b}}{2^k}$$

El algoritmo podría quedar como sigue:

```
cociente(c,d,a,b)
{
/* Calcular a= c div d
            b= c mod d

   Digito_Mas_Significativo(a) => Devuelve el valor del digito de
                                  mayor peso de a.
   Bits_Significativos(x) => Siendo x un digito, devuelve el numero
                             de bits de los digitos quitando los
                             ceros de la izquierda.
*/

   despl=Num_bits_digito-Bits_significativos(Digito_Mas_Significativo(d));

   factor=2^despl;  /* Desplazamos d hasta que su digito
                       mas significativo tenga su bit de mayor
                       peso a 1 (di>=B/2)            */
   dd=d*factor;
```

```
   cc=c*factor;
   if (Digitos(cc)==Digitos(c))
      Poner_Un_Cero_A_La_Izquierda(cc);  /* Garantizar que cc tiene */
                                         /* exactamente un digito  */
                                         /* mas que c              */

   t=Digito_Mas_Significativo(dd);

   /* Ya hemos normalizado. El cociente que obtengamos seguira siendo
      valido, pero el resto habra luego que dividirlo por factor */

   Poner_a_cero(a);

   for (i=Digitos(c)-Digitos(dd); i>=0; i--)
     {
        /* Subestimar digito del cociente (ai) */

        if (t==B-1)  /* No podemos dividir por t+1 */

           ai=cc[i+Digitos(dd)];  /* La estimacion es el primer
                                     digito significativo de cc */

          else ai=(cc[i+Digitos(dd)]*B+cc[i+Digitos(dd)-1])/(t+1);
                                /* La estimacion es el cociente entre
                                   los dos primeros digitos de cc y t+1 */

        cc=cc-ai*dd*B^i;                /* Restar a cc */


        while (cc[i+Digitos(dd)] ||  /* Si no se ha hecho cero el digito
                                        mas sign. de cc... */
               mayor(cc,dd*B^i))     /* o si cc es mayor o igual que
                                        dd*B^i,            */

           { ai++;            /* Hemos de aumentar la estimacion */
             cc=cc-dd*B^i;
           }
        a[i]=ai;
     }
  b=cc/factor; /* Lo que nos queda en cc es el resto */
               /* dividimos por factor para deshacer */
               /* la normalizacion */
}
```

Aunque a primera vista pueda parecer un algoritmo muy complejo, vamos a ver que no es tan complicado siguiendo su funcionamiento para un ejemplo concreto, con $B = 16$, $c = 3FBA2$, y $d = 47$:

1. Normalización: multiplicamos por 2 y nos queda $\bar{c} = 7F744$, $\bar{d} = 8E$
2. $a_2 = 7F \text{ div } 9 = E$; $\bar{c} = \bar{c} - a_2\bar{d}B^2 = 7F744 - 7C400 = 3344$
   Puesto que $\bar{c} < \bar{d}B^2 = 8E00$, no hay que incrementar a_2.
3. $a_1 = 33 \text{ div } 9 = 5$; $\bar{c} = \bar{c} - a_1\bar{d}B = 3344 - 2C60 = 6E4$
   Puesto que $\bar{c} < \bar{d}B = 8E0$, no hay que incrementar a_1.
4. $a_0 = 6E \text{ div } 9 = C$; $\bar{c} = \bar{c} - a_0\bar{d} = 6E4 - 6A8 = 3C$
   Puesto que $\bar{c} < \bar{d} = 8E$, tampoco hay que incrementar a_0
5. $a = E5C$; $b = \frac{\bar{c}}{2} = 1E$

5.3 Aritmética modular con enteros largos

Los algoritmos criptográficos de llave pública más extendidos se basan en operaciones modulares sobre enteros muy largos. Empleando los algoritmos del apartado 5.2 son inmediatas las operaciones de suma, resta y multiplicación módulo n. La división habremos de tratarla de manera diferente.

- Para sumar dos números basta con efectuar su suma entera y dividir después por el módulo, siendo el resto obtenido el resultado de la suma modular.
- Para restar basta con sumar el módulo al minuendo, restarle el sustraendo, y tomar el resto como en la suma.
- La multiplicación se lleva a cabo multiplicando los factores y tomando el resto de dividir el resultado por el módulo.
- La división habremos de implementarla multiplicando el dividendo por la inversa del divisor. Para calcular la inversa de un número módulo n basta con emplear el Algoritmo Extendido de Euclides, sustituyendo las operaciones elementales por llamadas a las operaciones con enteros largos descritas en la sección 5.2.

5.4 Ejercicios Propuestos

Capítulo 6

Criptografía y Números Aleatorios

Los algoritmos de llave pública, debido a su mayor orden de complejidad, suelen ser empleados en conjunción con algoritmos de llave privada de la siguiente forma (ver capítulo 9): el mensaje primero se codifica empleando un algoritmo simétrico y la llamada *clave de sesión*, que será diferente cada vez. Es la clave de sesión la que se codifica empleando criptografía asimétrica. La única manera de que estas claves sean seguras es que no exista ningún tipo de dependencia entre una clave y la siguiente, esto es, que sean aleatorias. De aquí surge el interés por los números aleatorios que surge en Criptografía.

Seguro que el lector conoce generadores pseudoaleatorios y diferentes tests de aleatoriedad. Los generadores tradicionales (congruenciales lineales) no nos permiten calcular secuencias realmente aleatorias, puesto que sabiendo un número obtenido con el generador podemos determinar cualquiera de los posteriores (recordemos que cada número de la secuencia se emplea como semilla para el siguiente). Si bien las series que producen estos generadores pasan los test estadísticos de aleatoriedad, son siempre las mismas, y esa condición es inadmisible para aplicaciones criptográficas. Un famoso ejemplo de este problema ocurrió en una de las primeras versiones de Netscape, que resultaba insegura debido al uso de un generador pseudoaleatorio demasiado *previsible*.

6.1 Secuencias pseudoaleatorias

En realidad es del todo imposible generar secuencias realmente aleatorias en una computadora, puesto que estas máquinas son totalmente deterministas. Todos los generadores pseudoaleatorios producen secuencias finitas y periódicas de números empleando operaciones aritméticas. Lo más que podremos conseguir es que estas secuencias sean lo más largas posible antes de comenzar a repetirse y que superen los tests estadísticos de aleatoriedad. En este sentido podríamos hablar de:

- *Secuencias estadísticamente aleatorias*: Secuencias que superan los tests estadísticos de aleatoriedad.

6.2 Secuencias criptográficamente aleatorias

El problema de las secuencias estadísticamente aleatorias, y lo que las hace poco útiles en Criptografía, es que son completamente predecibles. Definiremos, por tanto:

- *Secuencias criptográficamente aleatorias*: Para que una secuencia pseudoaleatoria sea *criptográficamente aleatoria*, ha de cumplir la propiedad de ser impredecible. Esto quiere decir que debe ser computacionalmente intratable el problema de averiguar el siguiente número de la secuencia teniendo total conocimiento acerca de todos los números anteriores y del algoritmo de generación empleado.

Lo que se suele hacer en la práctica es confiar en un generador pseudoaleatorio que trabaje con semillas bastante más largas que lo estrictamente necesario para generar una clave. De esta forma sólo empleamos parte de la semilla para construir la clave de sesión. Aunque una clave quedara comprometida, no se podría reconstruir totalmente la semilla que la generó, y por lo tanto un posible atacante será incapaz de predecir la siguiente clave.

Pero seguimos teniendo un problema: la semilla *real* debe ser auténticamente aleatoria, ya que si fuera generada mediante algún mecanismo *reproducible*, un atacante podría obtenerla tratando de imitar dicho proceso de generación. Eso nos lleva al siguiente tipo de secuencias, las auténticas secuencias aleatorias.

6.3 Secuencias totalmente aleatorias

Como ya se ha dicho antes, no existe la aleatoriedad cuando se habla de computadoras. En realidad se puede decir que no existen en el Universo sucesos cien por cien aleatorios. En cualquier caso, y a efectos prácticos, consideraremos un tercer tipo de secuencias pseudoaleatorias:

- *Secuencias aleatorias*: Diremos que una secuencia es totalmente aleatoria (o simplemente aleatoria) si no puede ser reproducida de manera fiable.

En este sentido se buscan fuentes de datos realmente aleatorias para calcular la semilla del generador, como pueden ser movimientos del ratón o pulsaciones de teclas que haga el usuario. Nótese que realmente nuestra secuencia de claves de sesión no corresponderá a una secuencia totalmente aleatoria, pero si protegemos la semilla del generador se comportará como si lo fuese, lo cual es suficiente para nuestros propósitos.

Parte III

Criptografía de Llave Privada

Capítulo 7

Criptografía Clásica

En este capítulo haremos un breve repaso de los mecanismos criptográficos considerados *clásicos*. Podemos llamar así a todos los sistemas de cifrado anteriores a la II Guerra Mundial, o lo que es lo mismo, al nacimiento de las computadoras.

La transición desde la Criptografía clásica a la moderna se da precisamente durante la II Guerra Mundial, cuando el Servicio de Inteligencia aliado *rompe* la máquina de cifrado del ejército alemán, llamada ENIGMA.

Todos los algoritmos criptográficos clásicos son simétricos, ya que hasta mediados de los años setenta no nació la Criptografía asimétrica, y por esa razón este capítulo se engloba dentro del bloque de la asignatura dedicado a los algoritmos de llave privada.

7.1 Algoritmos Clásicos de Cifrado

Estudiaremos en esta sección algunos criptosistemas que en la actualidad han perdido su eficacia, debido a que son fácilmente criptoanalizables empleando cualquier computadora doméstica, pero que fueron empleados con éxito hasta principios del siglo XX. Algunos se remontan incluso, como el algoritmo de César, a la Roma Imperial. Sin embargo mantienen un interés teórico, ya que nos van a permitir explotar algunas de sus propiedades para entender mejor los algoritmos modernos.

7.1.1 Cifrados Monoalfabéticos

Se engloban dentro de este apartado todos los algoritmos criptográficos que, sin desordenar los símbolos del lenguaje, establecen una correspondencia única para todos ellos en todo el mensaje. Es decir, si al símbolo A le corresponde el símbolo D, esta correspondencia se mantiene a lo largo de todo el mensaje.

Algoritmo de César

El algoritmo de *César*, llamado así porque es el que empleaba Julio César para enviar mensajes secretos, es uno de los algoritmos criptográficos más simples. Consiste en sumar 3 al número de orden de cada letra. De esta forma a la A le corresponde la D, a la B la E, y así sucesivamente. Si asignamos a cada letra un número ($A = 0$,$B = 1\ldots$), y consideramos un alfabeto de 26 letras, la transformación criptográfica sería:

$$C = (M + 3) \bmod 26$$

obsérvese que este algoritmo ni siquiera posee clave, la transformación siempre es la misma.

Sustitución Afín

Es el caso general del algoritmo de César. Su transformación sería:

$$E_k(M) = (aM + b) \bmod 26$$

siendo a y b dos números enteros menores que el cardinal N del alfabeto, y cumpliendo que $mcd(a, N) = 1$. La clave k es el par (a, b).

Cifrado Monoalfabético General

Es el caso más general de cifrado monoalfabético. La sustitución ahora es arbitraria, siendo la clave k precisamente la tabla de sustitución de un símbolo por otro.

Criptoanálisis de los Métodos de Cifrado Monoalfabéticos

El cifrado monoalfabético constituye la familia de métodos más simple de criptoanalizar, puesto que las propiedades estadísticas del texto plano se conservan en el criptograma. Supongamos que, por ejemplo, la letra que más aparece en Castellano es la A. Parece lógico que la letra más frecuente en el texto codificado sea aquella que corresponde con la A. Emparejando las frecuencias relativas de aparición de cada símbolo en el mensaje cifrado con el histograma de frecuencias del idioma en el que se supone está el texto plano, podremos averiguar fácilmente la clave.

En el peor de los casos, es decir, cuando tenemos un emparejamiento arbitrario, la distancia de unicidad de Shannon que obtenemos es:

$$N = \frac{H(K)}{D} = \frac{\log_2(n!)}{D} \tag{7.1}$$

donde D es la redundancia del lenguaje empleado en el mensaje original, y n es el número de símbolos de dicho lenguaje. Suponemos que las $n!$ claves diferentes son equiprobables en principio.

En casos más restringidos, como el afín, el criptoanálisis es aún más simple, puesto que el emparejamiento de todos los símbolos debe responder a alguna combinación de coeficientes (a, b).

7.1.2 Cifrados Polialfabéticos

En los cifrados polialfabéticos la sustitución aplicada a cada carácter varía en función de la posición de éste dentro del texto plano. En realidad corresponde a la aplicación cíclica de n cifrados monoalfabéticos.

Cifrado de Vigenere

Es un ejemplo típico de cifrado polialfabético, cuya clave es una secuencia de letras $K = \{k_0, k_1, \ldots k_{d-1}\}$ y que emplea la siguiente función de cifrado:

$$E_k(m_i) = m_i + k_{(i \mod d)} \pmod n$$

siendo m_i el $i-$ésimo símbolo del texto plano y n el cardinal del alfabeto de entrada.

Criptoanálisis

Para criptoanalizar este tipo de claves basta con efectuar d análisis estadísticos independientes agrupando los símbolos según la k_i empleada para codificarlos. Para estimar d, buscaremos la periodicidad de los patrones comunes que puedan aparecer en el texto cifrado. Obviamente, para el criptoanálisis, necesitaremos al menos d veces más cantidad de texto que con los métodos monoalfabéticos.

7.1.3 Cifrados por Sustitución Homofónica

Para paliar la sensibilidad frente a ataques basados en el estudio de las frecuencias de aparición de los símbolos, existe una familia de algoritmos que trata de ocultar las propiedades estadísticas del texto plano empleando un alfabeto de salida con más símbolos que el alfabeto de entrada.

Supongamos que nuestro alfabeto de entrada posee cinco letras, $\{a, b, c, d\}$. Supongamos además que en nuestros textos la letra a aparece con una probabilidad 0.4, y el resto con probabilidad 0.2. Podríamos emplear el siguiente alfabeto de salida $\{\alpha, \beta, \gamma, \delta, \epsilon\}$. Efectuaremos entonces la siguiente asociación:

$$
\begin{aligned}
E(a) &= \begin{cases} \alpha & \text{con probabilidad } 1/2 \\ \beta & \text{con probabilidad } 1/2 \end{cases} \\
E(b) &= \gamma \\
E(c) &= \delta \\
E(d) &= \epsilon
\end{aligned}
$$

En el texto cifrado ahora todos los símbolos aparecen con igual probabilidad, lo que imposibilita un ataque basado en frecuencias. El inconveniente es que necesitamos conocer la distribución estadística de los símbolos en el texto plano y que el alfabeto de salida es mayor que el de entrada.

7.1.4 Cifrados de Transposición

Este tipo de mecanismos de cifrado no sustituye los símbolos por otros, sino que cambia su orden dentro del texto. Un mecanismo de transposición podría consistir en colocar el texto en una tabla de n columnas, y dar como texto cifrado los símbolos de una columna (ordenados de arriba a abajo) concatenados con los de otra, etc. La clave sería el número n y el orden en el que se leen las columnas.

Por ejemplo, supongamos que queremos cifrar el texto *"El perro de San Roque no tiene rabo"*, con $n = 5$ y la permutación $\{3, 2, 5, 1, 4\}$ como clave. Colocamos el texto en una tabla y obtenemos:

1	2	3	4	5
E	L		P	E
R	R	O		D
E		S	A	N
	R	O	Q	U
E		N	O	
T	I	E	N	E
	R	A	B	O

Tendríamos como texto cifrado *" Osonealr r irednu eoere et p aqonb"*.

Criptoanálisis

Este tipo de mecanismos de cifrado se puede criptoanalizar efectuando un estudio estadístico sobre la frecuencia de aparición de pares y tripletas de símbolos en el texto plano.

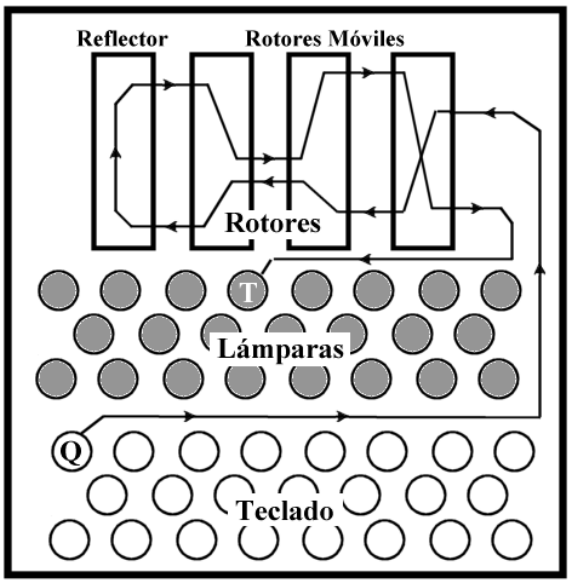

Figura 7.1: Esquema de la máquina Enigma.

7.2 Máquinas de Rotores. La Máquina ENIGMA

En el año 1923, un ingeniero alemán llamado Arthur Scherbius patentó una máquina específicamente diseñada para facilitar las comunicaciones seguras. Se trataba de un instrumento de apariencia simple, parecido a una máquina de escribir. Quien deseara codificar un mensaje sólo tenía que teclearlo y las letras correspondientes al mensaje cifrado se irían iluminando en un panel. El destinatario copiaba dichas letras en su propia máquina y el mensaje original aparecía de nuevo. La *clave* la constituían las posiciones iniciales de tres tambores o *rotores* que el ingenio poseía en su parte frontal.

En la figura 7.1 podemos apreciar un esquema de esta máquina, llamada ENIGMA. Los rotores no son más que tambores con contactos en su superficie y cableados en su interior, de forma que cuando se pulsa una tecla, la posición de éstos determina cuál es la letra que se ha de iluminar. Cada vez que se pulsa una tecla el primer rotor avanza una posición; el segundo avanza cuando el anterior ha dado una vuelta completa y así sucesivamente. El reflector no existía en los primeros modelos, se introdujo posteriormente para permitir que la misma máquina sirviera tanto para cifrar como para descifrar, como veremos posteriormente.

7.2.1 Un poco de Historia

ENIGMA pronto llamó la atención del ejército alemán, que la utilizó de forma intensiva a lo largo de la II Guerra Mundial. Además se le aplicaron varias mejoras, como incluir un pequeño sistema previo de permutación de letras, llamado *Stecker*, hacer que los rotores fueran intercambiables (se podían elegir y colocar en cualquier orden tres de entre cinco disponibles), e incluso se amplió el número de rotores a cuatro.

Aunque ENIGMA parecía virtualmente imposible de romper, presentaba una serie de debilidades, tanto en su diseño como en los mecanismos empleados para utilizarla, que fueron aprovechadas por el ejército aliado. El primero en conseguir avances significativos fue el servicio de inteligencia polaco, ya que en 1928 se recibió accidentalmente en ese país un ejemplar por correo ordinario. Al ser reclamado urgentemente por las autoridades alemanas, despertó el interés del servicio secreto, que dispuso de un fin de semana entero para desmontar, analizar y volver a empaquetar la máquina antes de devolverla el lunes.

El conocimiento preciso de la máquina permitió a un equipo de tres matemáticos (Marian Rejewski, Jerzy Rozycki y Henryk Zygalski) elaborar un mecanismo para aprovechar una debilidad, no en la máquina en sí, sino en el protocolo empleado por el ejército alemán para colocar los rotores al principio de cada mensaje. Dicho protocolo consistía en escoger una posición de un libro de claves, y enviar tres letras cualesquiera *dos veces*, para evitar posibles errores. En realidad se estaba introduciendo una redundancia tal en el mensaje que permitía obtener sin demasiados problemas la clave empleada. Se construyó un aparato que permitía descifrar los mensajes, y se bautizó como *Ciclómetro*.

En 1938 Alemania cambió el protocolo, lo cual obligó a los matemáticos polacos a refinar su sistema, aunque básicamente se seguían enviando tres letras repetidas. No vamos a entrar en detalles, pero el ataque se basaba en buscar ordenaciones de los rotores que llevaran dos letras consecutivas iguales a la misma letra. Estas configuraciones especiales daban una información vital sobre la posición inicial de los rotores para un mensaje concreto. Se construyó entonces una versión mejorada del ciclómetro, llamada *Bomba*, que era capaz de encontrar estas configuraciones de forma automática. Sin embargo, a finales de ese mismo año se introdujeron dos rotores adicionales, lo cual obligaba a emplear sesenta *bombas* simultáneamente para romper el sistema. Polonia simplemente carecía de medios económicos para afrontar su construcción.

Los polacos entonces pusieron en conocimiento de los servicios secretos británico y francés sus progresos, esperando poder establecer una vía de colaboración para seguir descifrando los mensajes germanos, pero la invasión de Polonia era inminente. Tras destruir todas las pruebas que pudieran indicar al ejército alemán el éxito polaco frente a ENIGMA, el equipo de Rejewski huyó precipitadamente, transportando lo que pudieron salvar en varios camiones. Tras pasar por Rumanía e Italia, y tener que quemar todos los camiones por el camino excepto uno, llegaron a París, donde colaboraron con un equipo de siete españoles expertos en criptografía, liderados por un tal Camazón. Cuando al año siguiente Alemania invadió Francia el nuevo equipo tuvo que huir a Africa, y posteriormente instalarse en Montpellier, donde reanudaron sus trabajos. En 1942, la entrada alemana en Vichy forzó a los matemáticos a escapar de nuevo, los polacos a España (donde murió Rozycki), y los españoles a Africa, donde

se perdió definitivamente su pista.

Cuando el equipo de Rejewski llegó por fin a Inglaterra, ya no se le consideró seguro, al haber estado en contacto con el enemigo, y se le confiaron únicamente trabajos menores. Mientras tanto, en Bletchley Park, Alan Turing desarrollaba una segunda *Bomba* basándose en los estudios del polaco, más evolucionada y rápida que su antecesora, en el marco del proyecto *ULTRA* británico, que se encargaba de recoger información acerca de los sistemas de comunicaciones germanos. En su desarrollo participó también Max Von Neuman, y se llegaron a construir varios cientos de ellas. Este nuevo dispositivo aprovechaba una debilidad esencial en ENIGMA: un mensaje no puede codificarse en sí mismo, lo cual implica que ninguna de las letras del texto plano puede coincidir con ninguna del texto cifrado. La Bomba de Turing partía de una *palabra adivinada* (en contra de las normas de uso de ENIGMA, la mayoría de los mensajes que enviaba el ejército alemán comenzaban de igual forma, lo cual facilitó la tarea del equipo aliado enormemente), y buscaba un emparejamiento con el mensaje cifrado tal que el supuesto texto plano y el fragmento de criptograma asociado no coincidieran en ninguna letra. A partir de ahí la Bomba realizaba una búsqueda exhaustiva de la configuración inicial de la máquina para decodificar el mensaje.

Un hecho bastante poco conocido es que Alemania regaló al régimen de Franco casi una veintena de máquinas ENIGMA, que fueron utilizadas para comunicaciones secretas hasta entrados los años cincuenta, suponemos que para regocijo de los servicios de espionaje británico y norteamericano.

7.2.2 Consideraciones Teóricas Sobre la Máquina ENIGMA

Observemos que un rotor no es más que una permutación dentro del alfabeto de entrada. El cableado hace que cada una de las letras se haga corresponder con otra. Todas las letras tienen imagen y no hay dos letras con la misma imagen. Si notamos una permutación como π, podemos escribir que la permutación resultante de combinar todos los rotores en un instante dado es:

$$\pi_{total} = \langle \pi_0, \pi_1, \pi_2, \pi_3, \pi_2^{-1}, \pi_1^{-1}, \pi_0^{-1} \rangle$$

La permutación π_3 corresponde al reflector, y debe cumplir que $\pi_3 = \pi_3^{-1}$, es decir, que aplicada dos veces nos dé lo mismo que teníamos al principio. De esta forma se cumple la propiedad de que, para una misma posición de los rotores, la codificación y la decodificación son simétricas.

La fuerza de la máquina ENIGMA radicaba en que tras codificar cada letra se giran los rotores, lo cual hace que la permutación que se aplica a cada letra sea diferente, y que esa permutación además no se repetirá hasta que los rotores recuperen su posición inicial. Tengamos en cuenta que hay 17576 posiciones iniciales de los rotores, y 60 combinaciones de tres rotores a partir de los cinco de entre los que se puede elegir. Puesto que el *stecker* también presenta un número bastante alto de combinaciones, existe una cantidad enorme de posibles

disposiciones iniciales del la máquina (al menos para aquella época). La potencia del método de criptoanálisis empleado radica en que se podía identificar un emparejamiento plausible entre el criptograma y el texto plano, de forma que sólo bastaba con rastrear dentro del espacio de posibles configuraciones para encontrar aquella que llevara a cabo la transformación esperada. No disponer de dicho emparejamiento hubiera complicado enormemente el criptoanálisis, tal vez hasta el punto de hacerlo fracasar.

7.2.3 Otras Máquinas de Rotores

Además de la máquina alemana ENIGMA, existieron otros dispositivos criptográficos basados en rotores. Estos dispositivos son mucho menos conocidos por diversas razones.

La máquina SIGABA, empleada por el ejército norteamericano, de la cual apenas se conoce nada por razones obvias.

También citaremos las máquinas japonesas empleadas en la II Guerra Mundial, que se denominaron PURPLE y RED, cuyas características son secretas o desconocidas. De hecho los norteamericanos declararon no haber tenido nunca ocasión de capturar nada más que un ejemplar, concretamente en la embajada japonesa tras la toma de Berlín.

Capítulo 8

Algoritmos Simétricos de Cifrado

8.1 Cifrado de producto

La gran mayoría de los algoritmos de cifrado simétricos se apoyan en los conceptos de confusión y difusión inicialmente propuestos por Shannon (ver sección 3.8, en la página 38), que se combinan para dar lugar a los denominados *cifrados de producto*.

Recordemos que la confusión consiste en tratar de ocultar la relación que existe entre el texto plano, el texto cifrado y la clave. Un buen mecanismo de confusión hará demasiado complicado extraer relaciones estadísticas entre las tres cosas. Por su parte la difusión trata de repartir la influencia de cada bit del mensaje original lo más posible entre el mensaje cifrado.

Hemos de hacer notar que la confusión por sí sola sería suficiente, ya que si establecemos una tabla de sustitución completamente diferente para cada clave con todos los textos planos posibles tendremos un sistema extremadamente seguro. Sin embargo, dichas tablas ocuparían cantidades astronómicas de memoria, por lo que en la práctica serían viables. Por ejemplo, un algoritmo que codificara bloques de 128 bits empleando una clave de 80 bits necesitaría una tabla de aproximadamente 10^{63} entradas.

Lo que en realidad se hace para conseguir algoritmos fuertes sin necesidad de almacenar tablas enormes es intercalar la confusión (sustituciones simples, con tablas pequeñas) y la difusión (permutaciones). Esta combinación se conoce como *cifrado de producto*. La mayoría de los algoritmos se basan en diferentes capas de sustituciones y permutaciones, estructura que denominaremos *Red de Sustitición-Permutación*. En muchos casos el criptosistema no es más que un paso simple de sustitución-permutación repetido n veces, como ocurre con DES.

8.1.1 Redes de Feistel

Muchos de los cifrados de producto tienen en común que dividen un bloque de longitud n en dos mitades, L y R. Se define entonces un cifrado de producto iterativo en el que la salida

de cada ronda se usa como entrada para la siguiente según la relación:

$$\left.\begin{array}{rcl} L_i & = & R_{i-1} \\ R_i & = & L_{i-1} \oplus f(R_{i-1}, K_i) \end{array}\right\} \quad \text{si } i < n.$$
$$\begin{array}{rcl} L_n & = & L_{n-1} \oplus f(R_{n-1}, K_n) \\ R_n & = & R_{n-1} \end{array} \tag{8.1}$$

Este tipo de estructura se denomina *Red de Feistel*, y es empleada en multitud de algoritmos, como DES, Lucifer, FEAL, CAST, Blowfish, etcétera. Tiene la interesante propiedad de ser reversible, independientemente de cómo sea la función f. Esto nos va a permitir emplear el mismo algoritmo tanto para cifrar como para descifrar.

8.1.2 Cifrados con Estructura de Grupo

Otra de las cuestiones a tener en cuenta en los cifrados de producto es la posibilidad de que posean estructura de *grupo*. Se dice que un cifrado tiene estructura de grupo si se cumple la siguiente propiedad:

$$\forall\, k_1, k_2 \quad \exists\, k_3 \quad \text{tal que} \quad E_{k_2}(E_{k_1}(M)) = E_{k_3}(M) \tag{8.2}$$

esto es, si hacemos dos cifrados encadenados con k_1 y k_2, existe una clave k_3 que realiza la transformación equivalente. Que un criptosistema tenga estructura de grupo no será, pues, deseable.

8.1.3 S-Cajas

Hemos dicho antes que para poder construir buenos algoritmos de producto, intercalaremos sustituciones sencillas (confusión), con tablas pequeñas, y permutaciones (difusión). Estas tablas pequeñas de sustitución se denominan de forma genérica S-Cajas.

Una S-Caja de m*n bits es una tabla de sustitución que toma como entrada cadenas de m bits y da como salida cadenas de n bits. DES, por ejemplo, emplea ocho S-Cajas de 6*4 bits. La utilización de una S-Caja es sencilla: se divide el bloque original en trozos de m bits y cada uno de ellos se sustituye por otro de n bits, haciendo uso de la S-Caja correspondiente. Normalmente, cuanto más grandes sean las S-Cajas, más resistente será el algoritmo resultante, aunque su elección para que den lugar a un buen algoritmo no es en absoluto trivial.

8.2 El Algoritmo DES

Es el algoritmo simétrico más extendido mundialmente. Data de mediados de los setenta, cuando fue adoptado como estándar para las comunicaciones seguras por el Gobierno de los

EE.UU. En realidad la NSA lo diseñó para ser implementado por hardware, pero al parecer por un malentendido entre ellos y la Oficina Nacional de Estandarización, su especificación se hizo pública con suficiente detalle como para que cualquiera pudiera implementarlo por software. No fue casualidad que el siguiente algoritmo adoptado (*Skipjack*) se considerara secreto.

El algoritmo DES codifica bloques de 64 bits empleando claves de 56 bits. Es una Red de Feistel de 16 rondas con dos permutaciones, una inicial (P_i) y otra final (P_f), tales que $P_i = P_f^{-1}$.

La función f se compone de una permutación de expansión (E), que convierte el bloque de 32 bits correspondiente en uno de 48. Después realiza un *or-exclusivo* con el valor K_i, también de 48 bits, aplica ocho S-Cajas de 6*4 bits, y efectua una nueva permutación P.

Se calcula un total de 16 valores de K_i, uno para cada ronda, efectuando primero una permutación inicial sobre la clave, y luego llevando a cabo desplazamientos a la izquierda de cada una de las dos mitades (de 28 bits) resultantes, y realizando una elección permutada de 48 bits en cada ronda, que será la K_i.

Para descifrar basta con usar el mismo algoritmo (ya que $P_i = P_f^{-1}$) empleando las K_i en orden inverso.

8.3 Variantes de DES

A mediados de julio de 1998, una empresa sin ánimo de lucro, llamada EFF (Electronic Frontier Foundation), logró fabricar una máquina capaz de descifrar un mensaje DES en menos de tres días. Curiosamente, pocas semanas antes, un alto cargo de la NSA había declarado que dicho algoritmo seguía siendo seguro, y que descifrar un mensaje resultaba aún excesivamente costoso, incluso para organizaciones gubernamentales. *DES-Cracker* costó menos de 40 millones de pesetas.

A pesar de su *caída*, DES sigue siendo ampliamente utilizado en multitud de aplicaciones, como por ejemplo las transacciones de los cajeros automáticos. De todas formas, el problema real de DES no radica en su diseño, sino en que emplea una clave demasiado corta (56 bits), lo cual hace que con el avance actual de las computadoras los ataques por la fuerza bruta comiencen a ser opciones realistas. Mucha gente se resiste a abandonar este algoritmo, precisamente porque ha sido capaz de *sobrevivir* durante veinte años sin mostrar ninguna debilidad en su diseño, y prefieren proponer variantes que, de un lado evitarían el riesgo de tener que confiar en algoritmos nuevos, y de otro permitirían aprovechar gran parte de las implementaciones por hardware existentes de DES.

8.3.1 DES Múltiple

Consiste en aplicar varias veces el algoritmo DES con diferentes claves al mensaje original. Se puede hacer ya que DES no presenta estructura de *grupo* (ecuación 8.2). El más común de

todos ellos es el Tripe-DES, que responde a la siguiente estructura:

$$C = E_{k_1}(E_{k_2}^{-1}(E_{k_1}(M)))$$

es decir, codificamos con la subclave k_1, decodificamos con k_2 y volvemos a codificar con k_1. La clave resultante es la concatenación de k_1 y k_2, con una longitud de 112 bits.

8.3.2 DES con Subclaves Independientes

Consiste en emplear subclaves diferentes para cada una de las 16 rondas de DES. Puesto que estas subclaves son de 48 bits, la clave resultante tendría 768 bits en total. No es nuestro objetivo entrar en detalles, pero empleando criptoanálisis diferencial, esta variante podría ser rota con 2^{61} textos planos escogidos, por lo que en la práctica no presenta un avance sustancial sobre DES estándar.

8.3.3 DES Generalizado

Esta variante emplea n trozos de 32 bits en cada ronda en lugar de dos, por lo que aumentamos tanto la longitud de la clave como el tamaño de mensaje que se puede codificar, manteniendo sin embargo el orden de complejidad del algoritmo. Se ha demostrado sin embargo que no sólo se gana poco en seguridad, sino que en muchos casos incluso se pierde.

8.3.4 DES con S-Cajas Alternativas

Consiste en utilizar S-Cajas diferentes a las de la versión original de DES. En la práctica no se han encontrado S-Cajas mejores que propias de DES. De hecho, algunos estudios han revelado que las S-Cajas originales presentan propiedades que las hacen resistentes a técnicas de criptoanálisis que no fueron conocidas fuera de la NSA hasta muchos años después de la aparición del algoritmo.

8.4 El algoritmo IDEA

El algoritmo IDEA (International Data Encryption Algorithm) es bastante más joven que DES, pues data de 1992. Para muchos constituye el mejor y más seguro algoritmo simétrico disponible en la actualidad. Trabaja con bloques de 64 bits de longitud y emplea una clave de 128 bits. Como en el caso de DES, se usa el mismo algoritmo tanto para cifrar como para descifrar.

Como ocurre con todos los algoritmos simétricos de bloques, IDEA se basa en los conceptos de confusión y difusión, haciendo uso de las siguientes operaciones elementales (todas ellas fáciles de implementar):

- XOR.
- Suma módulo 2^{16}.
- Producto módulo $2^{16} + 1$.

El algoritmo IDEA consta de ocho rondas. Dividiremos el bloque X a codificar, de 64 bits, en cuatro partes X_1, X_2, X_3 y X_4 de 16 bits. Denominaremos Z_i a cada una de las 52 subclaves de 16 bits que vamos a necesitar. Las operaciones que llevaremos a cabo en cada ronda son las siguientes:

1. Multiplicar X_1 por Z_1.
2. Sumar X_2 con Z_2.
3. Sumar X_3 con Z_3.
4. Multiplicar X_4 por Z_4.
5. Hacer un XOR entre los resultados del paso 1 y el paso 3.
6. Hacer un XOR entre los resultados del paso 2 y el paso 4.
7. Multiplicar el resultado del paso 5 por Z_5.
8. Sumar los resultados de los pasos 6 y 7.
9. Multiplicar el resultado del paso 8 por Z_6.
10. Sumar los resultados de los pasos 7 y 9.
11. Hacer un XOR entre los resultados de los pasos 1 y 9.
12. Hacer un XOR entre los resultados de los pasos 3 y 9.
13. Hacer un XOR entre los resultados de los pasos 2 y 10.
14. Hacer un XOR entre los resultados de los pasos 4 y 10.

La salida de cada iteración serán los cuatro sub-bloques obtenidos en los pasos 11, 12, 13 y 14, que serán la entrada del siguiente ciclo, en el que emplearemos las siguientes seis subclaves, hasta un total de 48. Al final de todo intercambiaremos los dos bloques centrales (en realidad con eso *deshacemos* el intercambio que llevamos a cabo en los pasos 12 y 13).

Después de la octava iteración, se realiza la siguiente transformación:

1. Multiplicar X_1 por Z_{49}.
2. Sumar X_2 con Z_{50}.

3. Sumar X_3 con Z_{51}.

4. Multiplicar X_4 por Z_{52}.

Las primeras ocho subclaves se calculan dividiendo la clave de entrada en bloques de 16 bits. Las siguientes ocho se calculan rotando la clave de entrada 25 bits a la izquierda y volviendo a dividirla, y así sucesivamente.

Las subclaves necesarias para descifrar se obtienen cambiando de orden las Z_i y calculando sus inversas para la suma o la multiplicación, según la tabla 8.1. A efectos de cálculo, consideraremos que la inversa para el producto de 0 módulo $2^{16}+1$ es 0.

Ronda	Subclaves de Cifrado						Subclaves de Descifrado					
1	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_{49}^{-1}	$-Z_{50}$	$-Z_{51}$	Z_{52}^{-1}	Z_{47}	Z_{48}
2	Z_7	Z_8	Z_9	Z_{10}	Z_{11}	Z_{12}	Z_{43}^{-1}	$-Z_{45}$	$-Z_{44}$	Z_{46}^{-1}	Z_{41}	Z_{42}
3	Z_{13}	Z_{14}	Z_{15}	Z_{16}	Z_{17}	Z_{18}	Z_{37}^{-1}	$-Z_{39}$	$-Z_{38}$	Z_{40}^{-1}	Z_{35}	Z_{36}
4	Z_{19}	Z_{20}	Z_{21}	Z_{22}	Z_{23}	Z_{24}	Z_{31}^{-1}	$-Z_{33}$	$-Z_{32}$	Z_{34}^{-1}	Z_{29}	Z_{30}
5	Z_{25}	Z_{26}	Z_{27}	Z_{28}	Z_{29}	Z_{30}	Z_{25}^{-1}	$-Z_{27}$	$-Z_{26}$	Z_{28}^{-1}	Z_{23}	Z_{24}
6	Z_{31}	Z_{32}	Z_{33}	Z_{34}	Z_{35}	Z_{36}	Z_{19}^{-1}	$-Z_{21}$	$-Z_{20}$	Z_{22}^{-1}	Z_{17}	Z_{18}
7	Z_{37}	Z_{38}	Z_{39}	Z_{40}	Z_{41}	Z_{42}	Z_{13}^{-1}	$-Z_{15}$	$-Z_{14}$	Z_{16}^{-1}	Z_{11}	Z_{12}
8	Z_{43}	Z_{44}	Z_{45}	Z_{46}	Z_{47}	Z_{48}	Z_7^{-1}	$-Z_9$	$-Z_8$	Z_{10}^{-1}	Z_5	Z_6
Final	Z_{49}	Z_{50}	Z_{51}	Z_{52}			Z_1^{-1}	$-Z_2$	$-Z_3$	Z_4^{-1}		

Tabla 8.1: Subclaves empleadas en el algoritmo IDEA

8.5 Modos de Operación para Algoritmos de Cifrado por Bloques

En esta sección comentaremos algunos métodos para aplicar cifrados por bloques a mensajes de gran longitud. En primer lugar, independientemente del método empleado para codificar, hemos de tener en cuenta lo que ocurre cuando la longitud de la cadena que queremos cifrar no es un múltiplo exacto del tamaño de bloque. Entonces tenemos que añadir información al final para que sí lo sea. El mecanismo más sencillo consiste en rellenar con ceros (o algún otro patrón) el último bloque que se codifica. El problema ahora consiste en saber cuando se descifra por dónde hay que cortar. Lo que se suele hacer es añadir como último byte del último bloque el número de bytes que se han añadido (ver figura 8.1). Esto tiene el inconveniente de que si el tamaño original es múltiplo del bloque, hay que alargarlo con otro bloque entero. Por ejemplo, si el tamaño de bloque fuera 64 bits, y sobraran cinco bytes al final, añadiríamos dos ceros y un tres. Si por contra no sobrara nada, tendríamos que añadir siete ceros y un ocho.

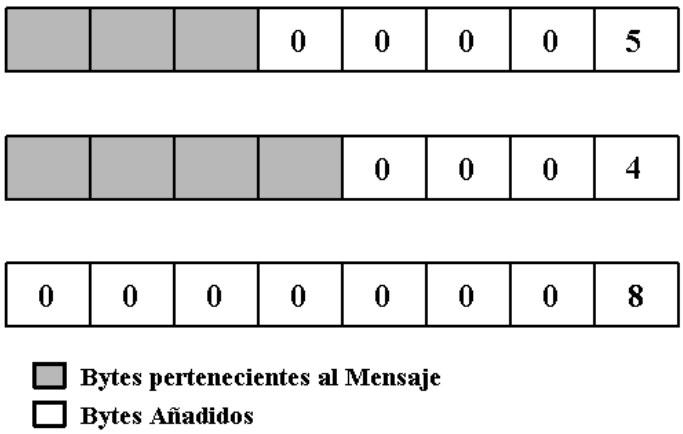

Figura 8.1: Relleno (*padding*) de los bytes del último bloque al emplear un algoritmo de cifrado por bloques.

8.5.1 Modo ECB

El modo ECB (*Electronic Codebook*) es el método más sencillo y obvio de aplicar un algoritmo de cifrado por bloques. Simplemente se subdivide la cadena que se quiere codificar en bloques del tamaño adecuado y se cifran todos ellos empleando la misma clave.

A favor de este método podemos decir que permite codificar los bloques independientemente de su orden, lo cual es adecuado para codificar bases de datos o ficheros en los que se requiera un acceso aleatorio. También es resistente a errores, pues si uno de los bloques sufriera una alteración, el resto quedaría intacto.

Por contra, si el mensaje presenta patrones repetitivos, el texto cifrado también los presentará, y eso es peligroso, sobre todo cuando se codifica información muy redundante (como ficheros de texto), o con patrones comunes al inicio y final (como el correo electrónico). Un contrincante puede en estos casos efectuar un ataque estadístico y extraer bastante información.

Otro riesgo bastante importante que presenta el modo ECB es el de la *sustitución de bloques*. El atacante puede cambiar un bloque sin mayores problemas, y alterar los mensajes incluso desconociendo la clave y el algoritmo empleados. Simplemente se *escucha* una comunicación de la que se conozca el contenido, como por ejemplo una transacción bancaria a nuestra cuenta corriente. Luego se escuchan otras comunicaciones y se sustituyen los bloques correspondientes al número de cuenta del beneficiario de la transacción por la versión codificada de nuestro número (que ni siquiera nos habremos molestado en descifrar). En cuestión de horas nos habremos hecho ricos.

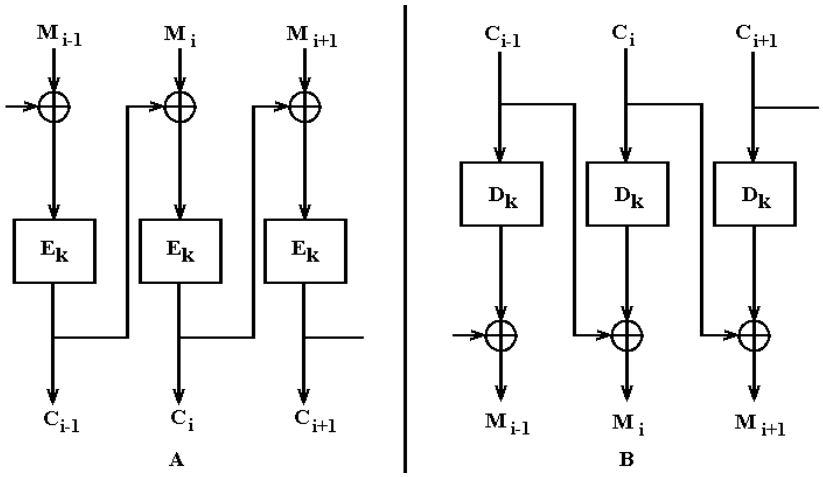

Figura 8.2: Modo de operación CBC. A) codificación, B) decodificación.

8.5.2 Modo CBC

El modo CBC (*Cipher Book Chaining Mode*) incorpora un mecanismo de retroalimentación en el cifrado por bloques. Esto significa que la codificación de bloques anteriores condiciona la codificación del bloque actual, por lo que será imposible sustituir un bloque individual en el mensaje cifrado. Esto se consigue efectuando una operación XOR entre el bloque del mensaje que queremos codificar y el último criptograma obtenido (ver figura 8.2).

En cualquier caso, dos mensajes idénticos se codificarán de la misma forma usando el modo CBC. Más aún, dos mensajes que empiecen igual se codificarán igual hasta llegar a la primera diferencia entre ellos. Para evitar esto se emplea un *vector de inicialización*, que puede ser un bloque aleatorio, como bloque inicial de la transmisión. Este vector será descartado en destino, pero garantiza que siempre los mensajes se codifiquen de manera diferente, aunque tengan partes comunes.

8.5.3 Modo CFB

El modo CBC no empieza a codificar (decodificar) hasta que no se tiene que transmitir un bloque completo. Esto puede ser un inconveniente, por ejemplo en el caso de terminales, que deberían poder transmitir cada carácter que pulsa el usuario individualmente. Tendremos pues que emplear un bloque completo para transmitir cada byte, rellenando el resto con ceros, pero esto hará que tengamos únicamente 256 mensajes diferentes en nuestra transmisión y que un atacante pueda efectuar un análisis estadístico. El modo de operación CFB (*Cipher-Feedback Mode*) permitirá codificar la información en unidades inferiores al tamaño del bloque, manteniendo un nivel de seguridad adecuado.

Sea p el tamaño de bloque del algoritmo simétrico, y sea n el tamaño de los bloques que queremos transmitir (n ha de divisor de p). Sea m_i el $i-$ésimo bloque del texto plano, de tamaño n. Empleamos entonces un registro de desplazamiento R de logitud m y lo cargamos con un vector de inicialización. Codificamos el registro R con el algoritmo simétrico y obtenemos en r sus n bits más a la izquierda. El bloque que deberemos enviar es $c_i = r \oplus m_i$. Desplazamos R n bits a la izquierda e introducimos c_i por la derecha.

Para descifrar basta con cargar el vector de inicialización en R y codificarlo, calculando r. Entonces $m_i = r \oplus c_i$. Desplazamos entonces R e introducimos c_i por la derecha como en el algoritmo de cifrado.

Nótese que si $n = p$, el modo CFB queda reducido al modo CBC.

8.6 Otros Modos

Existen protocolos criptográficos que no se basan en la transmisión de bloques, sino en un mecanismo secuencial de codificación de *streams* de tamaño variable. Estos algoritmos permiten cifrar un mensaje bit a bit de forma continua y enviar cada bit antes que el siguiente sea codificado. Funcionan a partir de lo que se llama un *generador de secuencia de clave (keystream generator)*, un algoritmo que genera una clave continua de longitud infinita (o muy grande) bit a bit. Lo que se hace es aplicar una operación XOR entre cada bit del texto plano y cada bit de la clave. En el destino existe otro generador idéntico sincronizado para llevar a cabo el descifrado. El problema fundamental es mantener ambos generadores sincronizados, para evitar errores si se pierde algún bit de la transmisión.

Los algoritmos de codificación por bloques pueden ser empleados como generadores de secuencia de clave. Existen para ello otros modos de operación de estos algoritmos, como el OFB (*Output-Feedback*), que incorporan mecanismos para mantener la sincronía entre los generadores de secuencia origen y destino.

8.7 Criptoanálisis de Algoritmos Simétricos

Se podría decir que el criptoanálisis se comenzó a estudiar seriamente con la aparición de DES. Mucha gente desconfiaba (y aún desconfía) del algoritmo propuesto por la NSA. Se dice que existen estructuras *extrañas*, que muchos consideran sencillamente *puertas traseras* colocadas por la Agencia para facilitarles la decodificación de los mensajes. Nadie ha podido aún demostrar ni desmentir este punto. Lo único cierto es que el interés por buscar posibles debilidades en él ha llevado a desarrollar técnicas que posteriormente han tenido éxito con otros algoritmos.

Ni que decir tiene que estos métodos no han conseguido doblegar a DES, pero sí representan mecanismos significativamente más eficientes que la fuerza bruta para criptoanalizar un mensaje. Los dos métodos que vamos a comentar parten de que nosotros podemos codificar

cuantos mensajes queramos usando la clave que queremos descubrir.

8.7.1 Criptoanálisis Diferencial

Descubierto por Biham y Shamir en 1990, permite efectuar un ataque de texto plano escogido a DES que resulta más eficiente que la fuerza bruta. Se basa en el estudio de los pares de criptogramas que surgen cuando se codifican dos textos planos con diferencias particulares, analizando la evolución de dichas diferencias a lo largo de las rondas de DES.

Para llevar a cabo un criptoanálisis diferencial se toman dos mensajes cualesquiera (incluso aleatorios) idénticos salvo en un número concreto de bits. Usando las diferencias entre los textos cifrados, se asignan probabilidades a las diferentes claves de cifrado. Conforme tenemos más y más pares, una de las claves aparece como la más probable. Esa será la clave buscada.

8.7.2 Criptoanálisis Lineal

El criptoanálisis lineal, descubierto por Mitsuru Matsui, basa su funcionamiento en tomar algunos bits del texto plano y efectuar una operación XOR entre ellos, tomar algunos del texto cifrado y hacerles lo mismo, y finalmente hacer un XOR de los dos resultados, obteniendo como resultado un único bit. Efectuando esa operación a muchos pares de texto plano y criptograma diferentes podemos ver si salen más ceros o más unos.

Existen combinaciones de bits que, bien escogidas, dan lugar a un sesgo significativo en la medida anteriormente definida, es decir, que el número de ceros (o unos) es significativamente superior. Esta propiedad nos va a permitir poder asignar mayor probabilidad a unas claves sobre otras y de esta forma descubrir la clave que buscamos.

Parte IV

Criptografía de Llave Pública

Capítulo 9

Algoritmos Asimétricos de Cifrado

Los algoritmos de llave pública, o algoritmos asimétricos, han demostrado su interés para ser empleados en redes de comunicación inseguras (Internet). Introducidos por Whitfield Diffie y Martin Hellman a mediados de los años 70, su novedad fundamental con respecto a la criptografía simétrica es que las claves no son únicas, sino que forman pares. Hasta la fecha han aparecido multitud de algoritmos asimétricos, la mayoría de los cuales son inseguros, otros son poco prácticos, bien sea porque el criptograma es considerablemente mayor que el mensaje original, bien sea porque la longitud de la clave es enorme. En la práctica muy pocos algoritmos son útiles, y se basan en general en plantear al atacante problemas matemáticos difíciles de resolver (ver capítulo 4). El algoritmo más popular es RSA, que aún se mantiene a salvo de ataques, si bien necesita una longitud de clave considerable. Otros algoritmos son los de ElGamal y Rabin.

En general, los algoritmos asimétricos emplean longitudes de clave mucho mayores que los algoritmos simétricos. Por ejemplo, mientras que para algoritmos simétricos se consideran seguros 128 bits, para algoritmos asimétricos se recomienda al menos 1024 bits en las claves. Además, la complejidad de cálculo de estos algoritmos los hace considerablemente más lentos que los algoritmos por bloques. En la práctica los algoritmos asimétricos se emplean únicamente para codificar la *clave de sesión* (simétrica) de cada mensaje.

9.1 Aplicaciones de los Algoritmos Asimétricos

Los algoritmos asimétricos poseen dos claves diferentes en lugar de una, K_p y K_P, denominadas *clave privada* y *clave pública*. Una de ellas se emplea para codificar, mientras que la otra se usa para decodificar. Dependiendo de la aplicación que le demos al algoritmo, la clave pública será la de cifrado o viceversa. Para que estos criptosistemas sean seguros también ha de cumplirse que a partir de una de las claves resulte extremadamente difícil calcular la otra.

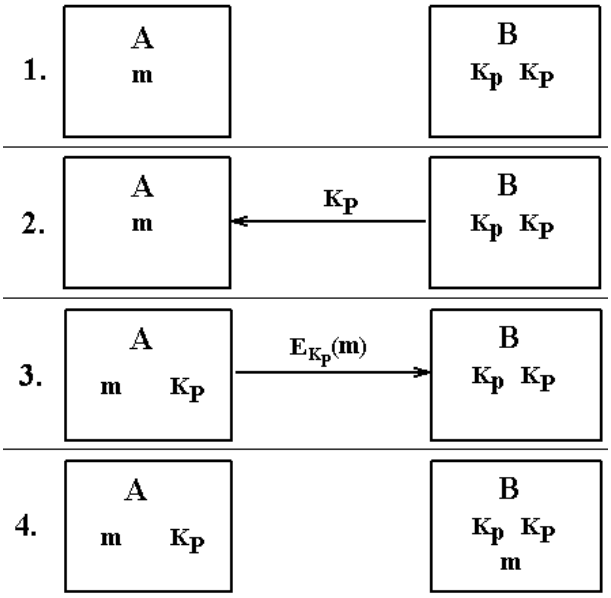

Figura 9.1: Transmisión de información empleando algoritmos asimétricos. 1. **A** tiene el mensaje m y quiere enviárselo a **B**; 2. **B** envía a **A** su clave pública, K_P; 3. **A** codifica el mensaje m y envía a **B** el criptograma $E_{K_P}(m)$; 4. **B** decodifica el criptograma empleando la clave privada K_p.

9.1.1 Protección de la Información

Una de las aplicaciones inmediatas de los algoritmos asimétricos es el cifrado de la información sin tener que transmitir la clave de decodificación, lo cual permite su uso en canales inseguros. Supongamos que A quiere enviar un mensaje a B (figura 9.1). Para ello solicita a B su clave pública K_P. A genera entonces el mensaje cifrado $E_{K_P}(m)$. Una vez hecho esto únicamente quien posea la clave K_p podrá recuperar el mensaje original m.

Nótese que para este tipo de aplicación, la llave que se hace pública es aquella que permite codificar los mensajes, mientras que la llave privada es aquella que permite descifrarlos.

9.1.2 Autentificación

La segunda aplicación de los algoritmos asimétricos es la autentificación de mensajes, con ayuda de funciones *resumen* (ver sección 10.1), que nos permiten obtener una firma a partir de un mensaje. Dicha firma es mucho más pequeña que el mensaje original, y es muy difícil encontrar otro mensaje que tenga la misma firma. Supongamos que A recibe un mensaje m de B y quiere comprobar su autenticidad. Para ello B genera un resumen del mensaje $r(m)$ (ver figura 9.2) y lo codifica empleando la clave de cifrado, que en este caso será privada. La

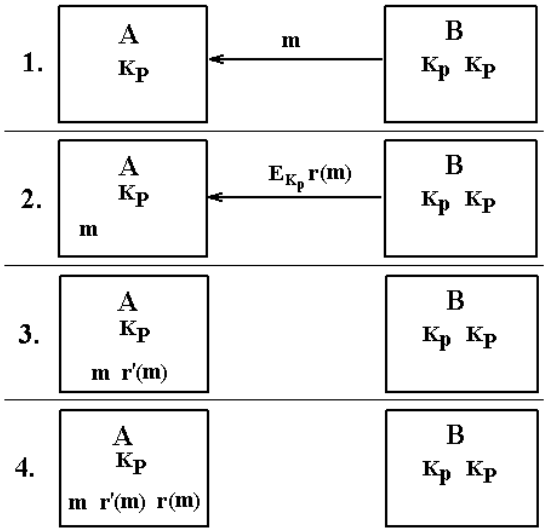

Figura 9.2: Autentificación de información empleando algoritmos asimétricos. 1. **A**, que posee la clave pública K_P de **B**, recibe el mensaje m y quiere autentificarlo; 2. **B** genera el resumen de m envía a **A** el criptograma asociado $E_{K_p}(r(m))$; 3. **A** genera por su cuenta $r'(m)$ y decodifica el criptograma recibido usando la clave K_P; 4. **A** compara $r(m)$ y $r'(m)$ para comprobar la autenticidad del mensaje m.

clave de descifrado se habrá hecho pública previamente, y debe estar en poder de A. B envía entonces a A el criptograma correspondiente a $r(m)$. A puede ahora generar su propia copia de $r(m)$ y compararla con el criptograma enviado por B. Si coinciden, el mensaje será auténtico, puesto que el único que posee la clave para codificar es precisamente B.

Nótese que en este caso la clave que se emplea para cifrar es la clave privada, justo al revés que para la simple codificación de mensajes.

Muchos de los algoritmos asimétricos presentan claves *duales*, esto quiere decir que si empleamos una para codificar, la otra permitirá decodificar y viceversa. Esto ocurre con el algoritmo RSA, por lo que un único par de claves es suficiente para codificar y autentificar.

9.2 El algoritmo RSA

De entre todos los algoritmos asimétricos, quizá RSA sea el más sencillo de comprender e implementar. Sus pares de claves son duales, por lo que sirve tanto para codificar como para autentificar. Su nombre proviene de sus tres inventores: Ron Rivest, Adi Shamir y Leonard Adleman. Desde su nacimiento nadie ha conseguido probar o rebatir su seguridad, pero se le

tiene como uno de los algoritmos asimétricos más seguros.

RSA se basa en la dificultad para factorizar números grandes. Las claves pública y privada se calculan a partir de un número que se obtiene como producto de dos *primos grandes*. El atacante se enfrenatará, si quiere recuperar un texto plano a partir del criptograma y la llave pública, a un problema de factorización (ver sección 4.4).

Para generar un par de llaves (K_P, K_p), en primer lugar se escogen aleatoriamente dos números primos grandes, p y q. Después se calcula el producto $n = pq$.

Escogeremos ahora un número e primo relativo con $(p-1)(q-1)$. (e, n) será la clave pública. Nótese que e debe tener inversa módulo $(p-1)(q-1)$, por lo que exisitrá un número d tal que

$$de \equiv 1 \pmod{(p-1)(q-1)}$$

es decir, que d es la inversa de e módulo $(p-1)(q-1)$. (d, n) será la clave privada. Esta inversa puede calcularse fácilmente empleando el Algoritmo Extendido de Euclides. Nótese que si desconocemos los factores de n, este cálculo resulta prácticamente imposible.

La *codificación* se lleva a cabo según la expresión:

$$c = m^e \pmod{n} \tag{9.1}$$

mientras que la *decodificación* se hará de la siguiente forma:

$$m = c^d \pmod{n} \tag{9.2}$$

ya que

$$c^d = (m^e)^d = m^{ed} = m^{k(p-1)(q-1)+1} = (m^k)^{(p-1)(q-1)} m$$

recordemos que $\Phi(n) = (p-1)(q-1)$, por lo que, según la ecuación (4.3), $(m^k)^{(p-1)(q-1)} = 1$, lo cual nos lleva de nuevo a m.

En la práctica, cogeremos p y q con un número grande de bits, por ejemplo 200, con lo que n tendrá 400 bits. Subdividiremos el mensaje que queramos enviar en bloques de 399 bits (de esta forma garantizamos que el valor de cada bloque sea menor que n) y efectuamos la codificación de cada uno. Obtendremos un mensaje cifrado ligeramente más grande, puesto que estará compuesto por bloques de 400 bits. Para decodificar partiremos el mensaje cifrado en bloques de 400 bits (ya que en este caso sabemos que el valor de cada bloque ha de ser menor que n), y obtendremos bloques de 399 bits.

El atacante, si quiere recuperar la clave privada a partir de la pública, debe conocer los factores p y q de n, y esto representa un problema computacionalmente intratable, siempre que n, p y q sean lo suficientemente grandes.

9.2.1 Seguridad del Algoritmo RSA

Técnicamente no es cierto que el algoritmo RSA deposite su fuerza en el problema de la factorización. En realidad el hecho de tener que factorizar un número para descifrar un mensaje sin la clave privada es una mera *conjetura*. Nadie ha demostrado que no pueda surgir un método en el futuro que permita descifrar un mensaje sin usar la clave privada y sin factorizar el módulo n. De todas formas, este método podría ser empleado como una nueva técnica para factorizar números enteros, por lo que la anterior afirmación se considera en la práctica cierta. De hecho, existen estudios que demuestran que incluso recuperar sólo algunos bits del mensaje original resulta tan difícil como descifrar el mensaje entero.

Aparte de factorizar n, podríamos intentar calcular $\Phi(n)$ directamente, o probar por la fuerza bruta tratando de encontrar la clave privada. Ambos ataques son más costosos computacionalmente que la factorización de n.

Otro punto que cabría preguntarse es qué pasaría si los primos p y q que escogemos realmente fueran compuestos. Recordemos que los algoritmos de prueba de primos que conocemos son probabilísticos, por lo que jamás tendremos la absoluta seguridad de que p y q son realmente primos. Pero obsérvese que si aplicamos, por ejemplo, treinta pasadas del algoritmo de Rabin-Miller (sección 4.4), las probabilidades de que el número escogido pase el test y siga siendo primo son de una contra 2^{60}: resulta más fácil que nos toque la primitiva y que simultáneamente nos parta un rayo (tabla 1.1). Por otra parte, si p o q fueran compuestos, el algoritmo RSA simplemente no funcionaría.

9.2.2 Ataques contra RSA

Aunque el algoritmo RSA es bastante seguro conceptualmente, existen algunos ataques que pueden ser efectivos y que se apoyan en deficiencias en el protocolo de uso del algoritmo. Veremos algunos de esos ataques en esta sección junto con la forma de protegerse frente a ellos.

Ataques de Intermediario

El *ataque de intermediario* puede darse con cualquier algoritmo asimétrico. Supongamos que A quiere establecer una comunicación con B, y que C quiere espiarla. Cuando A le solicite a B su clave pública K_b, C se interpone, obteniendo la clave de B y enviando a A una clave falsa k_c creada por él. Cuando A codifique el mensaje, C lo interceptará de nuevo, decodificándolo con su clave propia y empleando K_b para recodificarlo y enviarlo a B. Ni A ni B son conscientes de que sus mensajes están siendo interceptados.

La única manera de evitar esto consiste en asegurar a A que la clave pública que tiene de B es auténtica. Para ello nada mejor que ésta esté *firmada* por un amigo común, que certifique la autenticidad de la clave. En la actualidad existen los llamados *anillos de confianza*, que permiten certificar la autenticidad de las claves sin necesidad de centralizar el proceso. Por

eso se nos recomienda cuando instalamos paquetes como el PGP que firmemos todas las claves sobre las que tengamos certeza de su autenticidad, y sólamente ésas.

Ataques de Texto Plano Escogido

Existe una familia de ataques a RSA que explotan la posibilidad de que un usuario codifique y firme el mismo mensaje empleando el mismo par de llaves. Para que el ataque surta efecto, l a firma debe hacerse codificando el mensaje completo, no el resultado de una función *resumen* sobre él. Por ello se recomienda que las firmas digitales se lleven a cabo siempre sobre una función resumen del mensaje, nunca sobre el mensaje en sí.

Otro tipo de ataque con texto plano escogido podría ser el siguiente: para falsificar una firma sobre un mensaje m, se pueden calcular dos mensajes individuales m_1 y m_2, aparentemente inofensivos, tales que $m_1 m_2 = m$, y enviárselos a la *víctima* para que los firme. Entonces obtendríamos un m_1^d y m_2^d. Aunque desconozcamos d, si calculamos

$$m_1^d m_2^d = m^d \pmod{n}$$

obtendremos el mensaje m firmado.

Ataques de Módulo Común

Podría pensarse que, una vez generados p y q, será más rápido generar tantos pares de llaves como queramos, en lugar de tener que emplear dos números primos diferentes en cada caso. Sin embargo, si lo hacemos así, un atacante podrá decodificar nuestros mensajes sin necesidad de la llave privada. Sea m el texto plano, que codificamos empleando dos claves de cifrado diferentes e_1 y e_2. Los criptogramas que obtenemos son los siguientes:

$$\begin{aligned} c_1 &= m^{e_1} \pmod{n} \\ c_2 &= m^{e_2} \pmod{n} \end{aligned}$$

El atacante conoce pues n, e_1, e_2, c_1 y c_2. Si e_1 y e_2 son primos relativos, el Algoritmo Extendido de Euclides nos permitirá encontrar r y s tales que

$$re_1 + se_2 = 1$$

Ahora podemos hacer el siguiente cálculo

$$c_1^r c_2^s = m^{e_1 r} m^{e_2 s} = m^{e_1 r + e_2 s} = m^1 \pmod{n}$$

Recordemos que esto sólo se cumple si e_1 y e_2 son números primos relativos, pero eso es generalmente lo que ocurre. Por lo tanto, se deben generar p y q diferentes para cada par de claves.

Ataques de Exponente Bajo

Si el exponente de codificación e es demasiado bajo, existe la posibilidad de que un atacante pueda romper el sistema. Esto se soluciona *rellenando* los m que se codifican con bits aleatorios por la izquierda. Por ejemplo, si n es de 400 bits, una estrategia razonable sería coger bloques de 392 bits (que es un número exacto de bytes) e incluirles siete bits aleatorios por la izquierda. Cuando decodifiquemos simplemente ignoraremos esos siete bits.

Por otra parte, si d es demasiado bajo, también existen mecanismos para romper el sistema, por lo que se recomienda emplear valores altos para d.

Firmar y Codificar

Con el algoritmo RSA nunca se debe firmar un mensaje después de codificarlo. Debe firmarse primero. Existen ataques que aprovechan mensajes primero codificados y luego firmados, aunque se empleen funciones *resumen*.

9.3 Otros Algoritmos Asimétricos

9.3.1 Algoritmo de ElGamal

Fue diseñado en un principio para producir firmas digitales, pero posteriormente se extendió también para codificar mensajes. Se basa en el problema de los logaritmos discretos.

Para generar un par de llaves, se escoge un número primo p y dos números aleatorios g y x menores que p. Se calcula entonces

$$y = g^x \pmod{p}$$

La llave pública es (g, y, p), mientras que la llave privada es x.

Firmas Digitales de ElGamal

Para *firmar* un mensaje m basta con escoger un número k aleatorio, tal que $\text{mcd}(k, p-1) = 1$, y calcular

$$a = g^k \pmod{p} \tag{9.3}$$

Luego se emplea el Algoritmo Extendido de Euclides para resolver la ecuación

$$m = (xa + kb) \pmod{(p-1)}$$

La firma la constituye el par (a, b). En cuanto al valor k, debe mantenerse en secreto y ser diferente cada vez. La firma se verifica comprobando que

$$y^a a^b = g^m \pmod{p} \tag{9.4}$$

Codificación de ElGamal

Para codificar el mensaje m se escoge primero un número aleatorio k primo relativo con $(p-1)$, que también será mantenido en secreto. Calculamos entonces las siguientes expresiones

$$\begin{array}{rcl} a & = & g^k \pmod{p} \\ b & = & y^k m \pmod{p} \end{array} \tag{9.5}$$

El par (a, b) es el texto cifrado, de doble longitud que el texto original. Para decodificar se calcula

$$m = b/a^x \pmod{p} \tag{9.6}$$

9.3.2 Algoritmo de Rabin

El sistema de llave asimétrica de Rabin se basa en el problema de calcular raices cuadradas módulo un número compuesto. Esto es equivalente a factorizar dicho número.

En primer lugar escogemos dos números primos, p y q, ambos congruentes con 3 módulo 4 (los dos últimos bits a 1). Estos primos son la clave privada. La clave pública es su producto, $n = pq$.

Para codificar un mensaje m, simplemente se calcula

$$c = m^2 \pmod{n} \tag{9.7}$$

La decodificación del mensaje se hace calculando lo siguiente:

$$\begin{array}{rcl} m_1 & = & c^{(p+1)/4} \pmod{p} \\ m_2 & = & (p - c^{(p+1)/4}) \pmod{p} \\ m_3 & = & c^{(q+1)/4} \pmod{q} \\ m_4 & = & (q - c^{(q+1)/4}) \pmod{q} \end{array}$$

Luego se escogen a y b tales que $a = q(q^{-1} \pmod{p})$ y $b = p(p^{-1} \pmod{q})$. Los cuatro posibles mensajes originales son

$$\begin{aligned} m_a &= (am_1 + bm_3) \pmod{n} \\ m_b &= (am_1 + bm_4) \pmod{n} \\ m_c &= (am_2 + bm_3) \pmod{n} \\ m_d &= (am_2 + bm_4) \pmod{n} \end{aligned} \tag{9.8}$$

Desgraciadamente, no existe ningún mecanismo para decidir cuál de los cuatro es el auténtico, por lo que el mensaje deberá incluir algún tipo de información para que el receptor pueda distinguirlo de los otros.

Capítulo 10

Métodos de Autentificación

Por autentificación entenderemos cualquier método que nos permita garantizar alguna característica sobre algún objeto. Consideraremos dos grandes tipos de autentificación:

- Autentificación *de mensaje*. Queremos garantizar la procedencia de un mensaje conocido, de forma que podamos asegurarnos de que no es una falsificación. Este mecanismo se conoce habitualmente como *firma digital*.
- Autentificación *de usuario mediante contraseña*. En este caso se trata de garantizar la presencia de un usuario legal en el sistema. El usuario deberá poseer una contraseña secreta que le permita identificarse.
- Autentificación *de dispositivo*. Se trata de garantizar la presencia de un dispositivo válido. Este dispositivo puede estar solo o tratarse de una *llave electrónica* que sustituye a la contraseña para identificar un usuario.

10.1 Firmas Digitales. Funciones *Resumen*

En el capítulo 9 vimos que la criptografía asimétrica permitía autentificar información, es decir, poder asegurar que un mensaje m proviene de un emisor A y no de cualquier otro. Asímismo vimos que la autentificación debía hacerse empleando una *función resumen* y no codificando el mensaje completo. En esta sección estudiaremos dichas funciones resumen, que nos van a permitir crear *firmas digitales*.

Sabemos que un mensaje m puede ser autentificado codificando con la llave privada K_p el resultado de aplicarle una función resumen, $E_{K_p}(r(m))$. Esa información adicional (que denominaremos *firma* o *signatura* del mensaje m) sólo puede ser generada por el poseedor de la clave privada K_p. Cualquiera que posea la llave pública correspondiente estará en condiciones de decodificar y verificar la firma. Para que sea segura, la *función resumen* $r(x)$ debe poseer además ciertas características:

- $r(m)$ es de longitud fija, independientemente de la longitud de m.
- Dado m, es fácil calcular $r(m)$.
- Dado $r(m)$, es computacionalmente intratable recuperar m.
- Dado m, es computacionalmente intratable obtener un m' tal que $r(m) = r(m')$.

10.1.1 Longitud Adecuada para una Firma Digital

Para decidir cuál debe ser la longitud apropiada de una firma digital, contestaremos primero a la siguiente pregunta: ¿Cuál es la cantidad de personas que hay que poner en una habitación para que la probabilidad de que el cumpleaños de una de ellas sea el mismo día que el mío? Debemos calcular n tal que

$$n\frac{1}{365} > 0.5$$

luego $n > 182$. Sin embargo, ¿cuál sería la cantidad de gente necesaria para que dos personas cualesquiera tengan el mismo cumpleaños? Cada pareja tiene una probabilidad 1/365 de compartir el cumpleaños, y en un grupo de n personas hay $n(n-1)/2$ parejas diferentes de personas, luego

$$\frac{n(n-1)}{2} \cdot \frac{1}{365} > 0.5$$

Esto se cumple si $n > 19$, una cantidad *sorprendentemente* mucho menor que 182.

La consecuencia de esta *paradoja* es que aunque resulte muy difícil dado m calcular un m' tal que $r(m) = r(m')$, es mucho menos costoso buscar dos valores aleatorios m y m', tales que $r(m) = r(m')$.

En el caso de una signatura de 64 bits, necesitaríamos 2^{64} mensajes dado un m para obtener el m', pero bastaría con generar aproximadamente 2^{32} mensajes aleatorios para que aparecieran dos con la misma signatura (en general, si la primera cantidad es muy grande, la segunda cantidad es aproximadamente su raíz cuadrada). El primer ataque nos llevaría 600.000 años con una computadora que generara un millón de mensajes por segundo, mientras que el segundo necesitaría apenas una hora.

Hemos de añadir pues a nuestra lista de condiciones sobre las funciones de firma la siguiente:

- Debe ser difícil encontrar dos mensajes aleatorios, m y m', tales que $r(m) = r(m')$.

Actualmente se recomienda emplear signaturas de al menos 128 bits, siendo 160 bits el valor más usado en la actualidad.

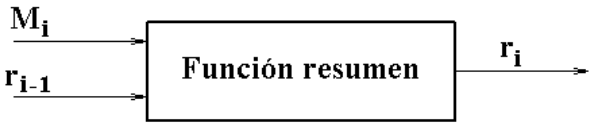

Figura 10.1: Estructura iterativa de una *función resumen*.

10.1.2 Estructura de una Función Resumen

En general, las funciones resumen se basan en la idea de *funciones de compresión*, que dan como resultado bloques de longitud n a partir de bloques de longitud m. Estas funciones se encadenan de forma iterativa, haciendo que la entrada en el paso i sea función del i-ésimo bloque del mensaje y de la salida del paso $i-1$ (ver figura 10.1). En general, se suele incluir en alguno de los bloques del mensaje m (al principio o al final), información sobre la longitud total del mensaje. De esta forma se reducen las probabilidades de que dos mensajes con diferentes longitudes den el mismo valor en su resumen.

En esta sección veremos dos algoritmos de generación de firmas digitales: MD5 y DSA.

10.1.3 Algoritmo MD5

Se trata de uno de los más populares algoritmos de generación de signaturas, debido a su inclusión en las primeras versiones de PGP. Resultado de una serie de mejoras sobre el algoritmo MD4, diseñado por Ron Rivest, procesa los mensajes de entrada en bloques de 512 bits, y produce una salida de 128 bits.

En primer lugar, el mensaje se alarga hasta que su longitud es exactamente 64 bits inferior a un múltiplo de 512 bits. El alargamiento se lleva a cabo añadiendo un 1 seguido de tantos ceros como sea necesario. En segundo lugar, se añaden 64 bits que representan la longitud del mensaje original, sin contar los bits añadidos en el proceso anterior. De esta forma tenemos el mensaje como un número entero de bloques de 512 bits, y le hemos añadido información sobre la longitud del mensaje.

Seguidamente, se inicializan cuatro registros de 32 bits con los siguientes valores (hexadecimales):

$$\begin{aligned} A &= 01234567 \\ B &= 89ABCDEF \\ C &= FEDCBA98 \\ D &= 76543210 \end{aligned}$$

Posteriormente comienza el *lazo principal* del algoritmo, que se repetirá para cada bloque

de 512 bits del mensaje. En primer lugar copiaremos los valores de A,B,C y D en otras cuatro variables, a,b,c y d. Luego definiremos las siguientes cuatro funciones:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\ G(X,Y,Z) &= (X \wedge Z) \vee ((Y \wedge (\neg Z)) \\ H(X,Y,Z) &= X \oplus Y \oplus Z \\ I(X,Y,Z) &= Y \oplus (X \wedge (\neg Z)) \end{aligned}$$

Ahora representaremos por m_j el j-ésimo bloque de 32 bits del mensaje m (de 0 a 15), y definiremos otras cuatro funciones:

$$\begin{aligned} FF(a,b,c,d,m_j,s,t_i) &\text{ representa } a = b + ((a + F(b,c,d) + m_j + t_i) \lhd s) \\ GG(a,b,c,d,m_j,s,t_i) &\text{ representa } a = b + ((a + G(b,c,d) + m_j + t_i) \lhd s) \\ HH(a,b,c,d,m_j,s,t_i) &\text{ representa } a = b + ((a + H(b,c,d) + m_j + t_i) \lhd s) \\ II(a,b,c,d,m_j,s,t_i) &\text{ representa } a = b + ((a + I(b,c,d) + m_j + t_i) \lhd s) \end{aligned}$$

donde la función $a \lhd s$ representa desplazar circularmente el valor a s bits a la izquieda.

Las 64 operaciones que se realizan en total quedan agrupadas en cuatro rondas.

- Primera Ronda:
  $FF(a,b,c,d,m_0,7,D76AA478)$
  $FF(d,a,b,c,m_1,12,E8C7B756)$
  $FF(c,d,a,b,m_2,17,242070DB)$
  $FF(b,c,d,a,m_3,22,C1BDCEEE)$
  $FF(a,b,c,d,m_4,7,F57C0FAF)$
  $FF(d,a,b,c,m_5,12,4787C62A)$
  $FF(c,d,a,b,m_6,17,A8304613)$
  $FF(b,c,d,a,m_7,22,FD469501)$
  $FF(a,b,c,d,m_8,7,698098D8)$
  $FF(d,a,b,c,m_9,12,8B44F7AF)$
  $FF(c,d,a,b,m_{10},17,FFFF5BB1)$
  $FF(b,c,d,a,m_{11},22,895CD7BE)$
  $FF(a,b,c,d,m_{12},7,6B901122)$
  $FF(d,a,b,c,m_{13},12,FD987193)$
  $FF(c,d,a,b,m_{14},17,A679438E)$
  $FF(b,c,d,a,m_{15},22,49B40821)$

- Segunda Ronda:
  $GG(a,b,c,d,m_1,5,F61E2562)$
  $GG(d,a,b,c,m_6,9,C040B340)$
  $GG(c,d,a,b,m_{11},14,265E5A51)$
  $GG(b,c,d,a,m_0,20,E9B6C7AA)$
  $GG(a,b,c,d,m_5,5,D62F105D)$

$GG(d, a, b, c, m_{10}, 9, 02441453)$
$GG(c, d, a, b, m_{15}, 14, D8A1E681)$
$GG(b, c, d, a, m_4, 20, E7D3FBC8)$
$GG(a, b, c, d, m_9, 5, 21E1CDE6)$
$GG(d, a, b, c, m_{14}, 9, C33707D6)$
$GG(c, d, a, b, m_3, 14, F4D50D87)$
$GG(b, c, d, a, m_8, 20, 455A14ED)$
$GG(a, b, c, d, m_{13}, 5, A9E3E905)$
$GG(d, a, b, c, m_2, 9, FCEFA3F8)$
$GG(c, d, a, b, m_7, 14, 676F02D9)$
$GG(b, c, d, a, m_{12}, 20, 8D2A4C8A)$

- Tercera Ronda:
  $HH(a, b, c, d, m_5, 4, FFFA3942)$
  $HH(d, a, b, c, m_8, 11, 8771F681)$
  $HH(c, d, a, b, m_{11}, 16, 6D9D6122)$
  $HH(b, c, d, a, m_{14}, 23, FDE5380C)$
  $HH(a, b, c, d, m_1, 4, A4BEEA44)$
  $HH(d, a, b, c, m_4, 11, 4BDECFA9)$
  $HH(c, d, a, b, m_7, 16, F6BB4B60)$
  $HH(b, c, d, a, m_{10}, 23, BEBFBC70)$
  $HH(a, b, c, d, m_{13}, 4, 289B7EC6)$
  $HH(d, a, b, c, m_0, 11, EAA127FA)$
  $HH(c, d, a, b, m_3, 16, D4EF3085)$
  $HH(b, c, d, a, m_6, 23, 04881D05)$
  $HH(a, b, c, d, m_9, 4, D9D4D039)$
  $HH(d, a, b, c, m_{12}, 11, E6DB99E5)$
  $HH(c, d, a, b, m_{15}, 16, 1FA27CF8)$
  $HH(b, c, d, a, m_2, 23, C4AC5665)$

- Cuarta Ronda:
  $II(a, b, c, d, m_0, 6, F4292244)$
  $II(d, a, b, c, m_7, 10, 432AFF97)$
  $II(c, d, a, b, m_{14}, 15, AB9423A7)$
  $II(b, c, d, a, m_5, 21, FC93A039)$
  $II(a, b, c, d, m_{12}, 6, 655B59C3)$
  $II(d, a, b, c, m_3, 10, 8F0CCC92)$
  $II(c, d, a, b, m_{10}, 15, FFEFF47D)$
  $II(b, c, d, a, m_1, 21, 85845DD1)$
  $II(a, b, c, d, m_8, 6, 6FA87E4F)$
  $II(d, a, b, c, m_{15}, 10, FE2CE6E0)$
  $II(c, d, a, b, m_6, 15, A3014314)$
  $II(b, c, d, a, m_{13}, 21, 4E0811A1)$
  $II(a, b, c, d, m_4, 6, F7537E82)$

$II(d, a, b, c, m_{11}, 10, BD3AF235)$
$II(c, d, a, b, m_2, 15, 2AD7D2BB)$
$II(b, c, d, a, m_9, 21, EB86D391)$

Finalmente, los valores resultantes de a,b,c y d son sumados con A,B,C y D, se procesa el siguiente bloque de datos. El resultado final del algoritmo es la concatenación de A,B,C y D.

A modo de curiosidad, diremos que las constantes t_i empleadas en cada paso son la parte entera del resultado de la operación $2^{32} \cdot \text{abs}(\sin(i))$, estando i representado en radianes.

En los últimos tiempos el algoritmo MD5 ha mostrado ciertas *debilidades*, aunque sin implicaciones prácticas reales, por lo que se considera en la actualidad un algoritmo seguro.

10.1.4 El Algoritmo SHA

El algoritmo SHA fue desarrollado por la NSA, para ser incluido en el estándar DSS (*Digital Signature Standard*). Al contrario que los algoritmos de cifrado propuestos por esta organización, SHA se considera seguro y libre de *puertas traseras*, ya que a la propoa NSA le interesa que el algoritmo sea totalmente seguro. Produce firmas de 160 bits, a partir de bloques de 512 bits del mensaje original.

El algoritmo es similar a MD5, y se inicializa igual que éste, solo que con cinco registros de 32 bits en lugar de cuatro:

$$\begin{aligned} A &= 67452301 \\ B &= EFCDAB89 \\ C &= 98BADCFE \\ D &= 10325476 \\ E &= C3D2E1F0 \end{aligned}$$

Una vez que los cinco valores están inicializados, se copian en cinco variables, a, b, c, d y e. El lazo principal tiene cuatro rondas con 20 operaciones cada una:

$$\begin{aligned} F(X,Y,Z) &= (X \wedge Y) \vee ((\neg X) \wedge Z) \\ G(X,Y,Z) &= X \oplus Y \oplus Z \\ H(X,Y,Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) \end{aligned}$$

La operación F se emplea en la primera ronda, la G en la segunda y en la cuarta, y la H en la tercera. Además se emplean cuatro constantes, una para cada ronda:

$$\begin{aligned} K_0 &= 5A827999 \\ K_1 &= 6ED9EBA1 \\ K_2 &= 8F1BBCDC \\ K_3 &= CA62C1D6 \end{aligned}$$

El bloque de mensaje m se trocea en 16 partes de 32 bits m_0 a m_{15} y se convierte en 80 trozos de 32 bits w_0 a w_{79} usando el siguiente algoritmo:

$$\begin{array}{lcll} w_t & = & m_t & \text{para } t = 0 \ldots 15 \\ w_t & = & (w_{t-3} \oplus w_{t-8} \oplus w_{t-14} \oplus w_{t-16}) \lhd 1 & \text{para } t = 16 \ldots 79 \end{array}$$

Como curiosidad, diremos que la NSA introdujo el desplazamiento a la izquierda para corregir una debilidad del algoritmo, pero no ha dado explicaciones sobre la naturaleza de esta debilidad.

El lazo principal del algoritmo es entonces el siguiente:

$$\begin{array}{ll} \text{FOR} & t = 0 \text{ TO } 79 \\ & i = t \text{ div } 20 \\ & Tmp = (a \lhd 5) + A(b,c,d) + e + w_t + K_i \\ & e = d \\ & d = c \\ & c = b \lhd 30 \\ & b = a \\ & a = Tmp \end{array}$$

siendo A la función F, G o H según el valor de t. Después los valores de a a e son sumados a los registros A a E y el algoritmo continua con el siguiente bloque de datos.

10.2 Autentificación de Dispositivos

Los algoritmos simétricos pueden ser empleados para autentificar dispositivos, siempre que éstos permitan hacer operaciones de cifrado/descifrado a la vez que impidan acceder físicamente a la clave que llevan almacenada. Un ejemplo de este mecanismo de autentificación lo tenemos en las tarjetas que emplean los teléfonos GSM. Dichas tarjetas llevan implementado un algoritmo simétrico de cifrado, y usan una clave k almacenada en un lugar de la memoria que no se puede leer desde el *exterior*. En cada tarjeta se graba una única clave, de la que se guarda una copia en lugar seguro. Si la compañía quiere identificar una tarjeta simplemente genera un bloque de bits aleatorio X y calcula su criptograma $E_k(X)$ asociado. Posteriormente se envía X a la tarjeta para que lo codifique. Si ambos mensajes codificados coinciden, la tarjeta será auténtica. Esta técnica se conoce como *autentificación por desafío* (fig. 10.2). Nótese que la clave k en ningún momento queda comprometida.

10.3 Autentificación de Usuario Mediante Contraseña

El sistema de autentificación basa su funcionamiento en una información secreta conocida únicamente por el usuario, que le permite identificarse positivamente frente al sistema. Supon-

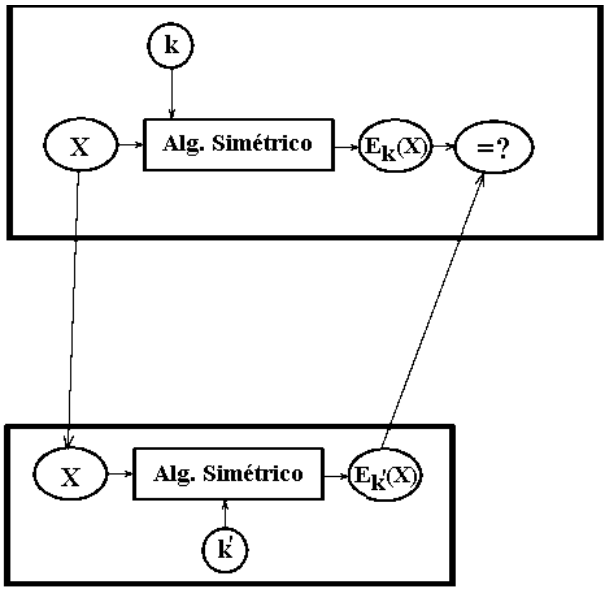

Figura 10.2: Esquema de autentificación por desafío.

dremos que el usuario se encuentra en un terminal *seguro*, es decir, libre de posibles ataques del exterior. Distinguiremos entonces dos casos claramente diferenciados:

a) El sistema se comunica con el usuario, pero éste no puede *entrar* en él. Piénsese en un cajero automático. El usuario carece de a los archivos del sistema y no tiene posibilidad ejecutar aplicaciones en él, únicamente puede llevar a cabo una serie de opearciones muy restringidas.

b) El sistema permite al usuario *entrar*. Este es el caso de los sistemas operativos como UNIX, que permiten a los usuarios operar con el sistema desde terminales remotos. Normalmente el usuario tiene acceso más o menos restringido a los archivos del sistema y puede ejecutar programas.

El primer caso es el más simple y sencillo de resolver. Basta con que el sistema mantenga la lista de usuarios y sus contraseñas asociadas en un archivo. Como este archivo no puede ser accedido desde el exterior, es imposible averiguar la clave de un usuario. Para protegerse de los ataques por la fuerza bruta, será suficiente con limitar el número de intentos desde un terminal concreto e introducir retardos cuando la contraseña introducida sea errónea.

El caso b) es considerablemente más complejo. Por un lado deberemos tomar las mismas medidas que en el caso anterior para protegernos de los ataques por la fuerza bruta, y por otro hemos de tener en cuenta que cualquier usuario puede acceder a algunos ficheros. En versiones *antiguas* de sistemas operativos UNIX el fichero con las contraseñas podía ser descargado por

cualquier usuario anónimo (que tiene un nombre concreto y una contraseña genérica), por lo que las palabras clave no pueden ser almacenadas como texto plano en dicho fichero. El mecanismo que surge entonces de manera inmediata consiste en almacenar en el fichero de claves la signatura de cada contraseña. De esta forma será difícil *adivinar* una contraseña que se ajuste a una signatura concreta. Los sistemas operativos modernos impiden además leer el fichero de claves de forma directa, pero eso no evita que en algunos casos éste pueda quedar comprometido.

10.3.1 Ataques Mediante Diccionario

El principal problema de las palabras clave son las elecciones *poco afortunadas* por parte de los usuarios. Desgraciadamente todavía hay personas que emplean su fecha de nacimiento, el nombre de algún familiar o la matrícula del coche como contraseña. Un atacante avispado podría tratar de generar millones de claves y construir un *diccionario*. El siguiente paso sería precalcular las signaturas de todas las claves de su diccionario. Si de alguna manera ha obtenido el fichero con las signaturas de las claves, bastaría con compararlas con las de su diccionario para obtener en pocos segundos una contraseña que le permita entrar en el sistema.

Para protegerse frente a este tipo de ataques se introduce en el cálculo de la signatura de la contraseña la denominada *sal*, que no es ni más ni menos que un conjunto de bits aleatorios que se añaden a la contraseña antes de calcular su firma. En el fichero de claves se almacenará, junto con la signatura, la sal necesaria para su obtención. Esto obligará al atacante a recalcular todas las signaturas de su diccionario antes de poder compararlas con cada una de las entradas del fichero de claves del sistema.

Además de estar bien salvaguardadas, las palabras clave han de cumplir una serie de condiciones para que puedan considerarse *seguras*:

1. Deben ser memorizadas. Una contraseña jamás debe ser escrita en un papel, por razones obvias.
2. Suficientemente complejas. Una buena contraseña debe constar de al menos ocho letras. Pensemos que si empleamos únicamente seis caracteres alfanuméricos (números y letras), tenemos *únicamente* unos dos mil millones de posibilidades. Teniendo en cuenta que hay programas para PC capaces de probar más de cuarenta mil claves en un segundo, una clave de estas características podría ser descubierta en menos de quince horas.
3. Fáciles de recordar. Puesto que una palabra clave ha de ser memorizada, no tiene sentido emplear contraseñas difíciles de recordar. En este sentido podemos seguir reglas como que la palabra se pueda pronunciar en voz alta, o que responda a algún acrónimo más o menos complejo. En este punto no debemos olvidar que hay que evitar a toda costa palabras que *signifiquen algo*.
4. Deben ser modificadas con frecuencia. Hemos de partir de la premisa de que toda palabra clave caerá tarde o temprano, por lo que será muy recomendable que nuestras

contraseñas sean cambiadas periódicamente. La frecuencia con la que se produzca el cambio dependerá de la complejidad de las claves y del nivel de seguridad que se desee alcanzar. Y lo más importante: ante cualquier sospecha, cambiar todas las claves.

10.4 Dinero Electrónico

El dinero *físico* es algo bastante engorroso. Incómodo de transportar, se desgasta con facilidad y suele ser susceptible de falsificación. Además debe ser cambiado periódicamente debido a la renovación de las monedas. Para sustituirlo están las tarjetas de crédito y los cheques. El problema que éstos presentan es que rompen el anonimato de quien los emplea, por lo que la privacidad queda comprometida. Existen sin embargo protocolos que permiten el intercambio de capital de una forma segura y anónima. Es lo que denominaremos *dinero electrónico*.

Las ventajas que reportará su extensión en un futuro próximo son evidentes. Facilitará el comercio electrónico, las compras por Internet, y además garantizará el anonimato en las transacciones comerciales. El problema es que los protocolos que permiten garantizar todas estas buenas propiedades son muchos y bastante complejos, por lo que haremos únicamente una breve introducción a título ilustrativo.

Supongamos que queremos enviar un cheque anónimo. Para ello creamos cien cheques por la misma cantidad, los metemos cada uno en un sobre y los enviamos al banco. El banco abre noventa y nueve al azar y se asegura de que todos llevan la misma cantidad. Al que queda le pone su sello sin abrirlo y nos lo devuelve, restando la cantidad de nuestra cuenta corriente. Tenemos ahora un cheque validado por el banco, pero del que el banco no sabe nada (la probabilidad de que tenga una cantidad diferente de la que el banco supone es únicamente del uno por ciento). Cuando entreguemos ese cheque y alguien quiera cobrarlo, bastará con que lo lleve al banco, que verificará su sello y abonará su importe, sin conocer su procedencia. Este protocolo se puede implementar mediante criptografía asimétrica de la siguiente forma: se construyen cien órdenes de pago anónimas y se envían al banco. Este las comprueba y firma digitalmente una, restando además la cantidad correspondiente en nuestra cuenta. El destinatario podrá cobrar la orden de pago cuando quiera.

El problema que surge con el protocolo anterior es que una orden se puede cobrar varias veces. Para evitar esto basta con incluir una cadena aleatoria en cada orden de pago, de forma que sea muy difícil tener dos órdenes con la misma cadena. Cada una de las cien órdenes tendrá pues una cadena diferente. El banco, cuando pague la cantidad, únicamente tendrá que comprobar la cadena de la orden para asegurarse de que no la ha pagado ya.

Ahora cada orden de pago es única, por lo que el banco puede detectar una orden duplicada, pero no sabe quién de los dos ha cometido fraude: el que paga o el que cobra. Existen mecanismos que permiten saber, cuando la orden de pago aparece duplicada, quién de los dos ha intentado engañar. Si lo ha hecho el cobrador, puede ser localizado sin problemas, pero ¿y si quien envía dos veces la misma orden es el pagador?. El protocolo completo de dinero electrónico hace que la identidad del pagador quede comprometida si envía dos veces la misma

orden de pago, por lo que podrá ser capturado.

10.5 Esteganografía

La esteganografía consiste en almacenar información *camuflada* dentro de otra información. Supongamos que queremos enviar un mensaje secreto a un interlocutor que se encuentra en un lugar donde la Criptografía está prohibida. Podríamos, por ejemplo, enviarle una imagen de mapa de bits y utilizar el bit menos significativo del color de cada píxel para guardar cada bit del mensaje secreto. La imagen será válida y un observador externo nunca sospechará que en realidad esconde un mensaje secreto.

Existen infinidad de métodos de esteganografía, sólo limitados por la imaginación, pero, ¿por qué incluir esta técnica dentro del capítulo dedicado a autentificación? La respuesta es sencilla: en general la esteganografía consiste en mezclar información *útil* con información de alguna otra naturaleza, que sólo sirve para despistar. Podríamos definirla entonces como el mecanismo que nos permite entresacar la información útil, reduciendo este problema a una simple autentificación.

Volvamos al ejemplo de la imagen de mapa de bits. Si el observador externo conociera el algoritmo que hemos empleado para camuflar nuestro mensaje dentro de la imagen, el sistema quedaría automáticamente comprometido. Alguien podría proponer entonces emplear la Criptografía y almacenar en los bits menos significativos de cada píxel la versión codificada del mensaje. Existe sin embargo una forma más elegante de proteger la información sin emplear ningún algoritmo criptográfico.

Podríamos generar una gran cantidad de información irrelevante y subdividirla junto con el mensaje original en pequeños *paquetes*, a los que añadiríamos un código de identificación (signatura), de forma que sólo los paquetes que corresponden al mensaje contengan una signatura correcta. Si enviamos una secuencia de paquetes en la que aparece el mensaje original entremezclado con la *basura*, sólo quien disponga del mecanismo de autentificación correcto (que podría depender de una *clave*) estará en condiciones de recuperar el mensaje original. Sin embargo, el mensaje ha sido enviado como texto plano, sin ser codificado en ningún momento. El ejemplo del mapa de bits no sería más que un caso particular de este esquema, en el que el algoritmo de autentificación simplemente considera válidos los bits menos significativos de cada píxel y descarta todos los demás.

Capítulo 11

PGP

El nombre PGP responde a las siglas *pretty good privacy* (privacidad bastante buena), y se trata de un proyecto iniciado a principios de los 90 por Phill Zimmerman. La total ausencia por aquel entonces de herramientas sencillas, potentes y baratas que acercaran la criptografía *seria* al usuario movió a su autor a desarrollar una aplicación que llenara este hueco.

Con el paso de los años, PGP se ha convertido en uno de los mecanismos más populares y seguros para mantener la seguridad y privacidad en las comunicaciones, especialmente a través del correo electrónico, tanto para pequeños usuarios como para grandes empresas. Hasta la fecha la política de distribución de PGP ha consistido en permitir su uso gratuito para usos no comerciales y en publicar el código fuente en su integridad, con el objetivo de satisfacer a los desconfiados y a los curiosos.

Actualmente PGP se ha convertido en un estándar internacional (*RFC 2440*), lo cual está dando lugar a la aparición de múltiples productos PGP, que permiten desde cifrar correo electrónico hasta a codificar particiones enteras del disco duro.

11.1 Fundamentos e Historia de PGP

PGP trabaja con criptografía asimétrica, y por ello tal vez su punto más fuerte sea precisamente la gran facilidad que ofrece al usuario a la hora de gestionar sus claves públicas y privadas. Si uno emplea algoritmos asimétricos, debe poseer las claves públicas de todos sus interlocutores, además de la clave privada propia. Con PGP surge el concepto de *anillo de claves* (o llavero), que no es ni más ni menos que el *lugar* que este programa proporciona para que el usuario guarde todas las claves que posee. El anillo de claves es un único fichero en el que se pueden efectuar operaciones de extracción e inserción de claves de manera sencilla, y que además proporciona un mecanismo de identificación y autentificación de llaves completo y simple de utilizar.

La historia de PGP se remonta a comienzos de los años 90. La primera versión era

completamente diferente a los PGP posteriores, además de ser incompatible. La familia de versiones 2.x.x fue la que alcanzó una mayor popularidad, e incluso es utilizada por mucha gente en la actualidad. Los PGP 2.x.x emplean únicamente los algoritmos Idea, RSA y MD5.

En algún momento una versión de PGP atravesó las fronteras de EE.UU. y nació la primera versión internacional de PGP, lo que le valió a Phill Zimmermann una investigación de más de tres años por parte del FBI, ya que supuestamente se habían violado las restrictivas leyes de exportación de material criptográfico que poseen los Estados Unidos. Para la versión 5 de PGP se subsanó este problema exportando una versión impresa del código fuente, que luego era reconstruida y compilada en Europa (más información en `http://www.pgpi.com`).

En la actualidad la versión internacional de PGP va por su versión 6.0.2i, mientras que la versión comercial ha alcanzado la versión 6.5.

11.2 Estructura de PGP

11.2.1 Codificación de Mensajes

Como el lector ya sabe, los algoritmos simétricos de cifrado son considerablemente más rápidos que los asimétricos. Por esta razón PGP cifra primero el mensaje empleando un algoritmo simétrico (ver figura 11.1) con una clave generada aleatoriamente (*clave de sesión*) y posteriormente codifica la clave haciendo uso de la llave pública del destinatario. Dicha clave es extraida convenientemente del anillo de claves públicas a partir del identificador suministrado por el usuario, todo ello de forma transparente, por lo que únicamente debemos preocuparnos de indicar el mensaje a codificar y la lista de identificadores de los destinatarios.

Cuando se trata de decodificar el mensaje, PGP simplemente busca en la cabecera las claves públicas con las que está codificado y nos pide una contraseña. La contraseña servirá para que PGP abra nuestro anillo de claves privadas y compruebe si tenemos una clave que permita decodificar el mensaje. En caso afirmativo, PGP descifrará el mensaje. Nótese que siempre que queramos hacer uso de una clave privada, habremos de suministrar a PGP la contraseña correspondiente, por lo que si el anillo de claves privadas quedara comprometido, un atacante aún tendría que averiguar nuestra contraseña para descifrar nuestros mensajes. No obstante, si el anillo de claves privadas quedara comprometido, lo mejor será revocar todas las claves que tuviera almacenadas y generar otras nuevas.

Como puede comprenderse, gran parte de la seguridad de PGP reside en la calidad del generador aleatorio que se emplea para generar las claves de sesión, puesto que si alguien logra predecir la secuencia de claves que estamos usando, podrá descifrar todos nuestros mensajes independientemente de los destinatarios a los que vayan dirigidos. Afortunadamente, PGP utiliza un método de generación de números pseudoaleatorios muy seguro (una secuencia aleatoria pura es imposible de conseguir, como se dijo en el capítulo 6), y protege criptográficamente la semilla aleatoria que necesita. No obstante, consideraremos *sensible* al fichero que contiene dicha semilla (normalmente `RANDSEED.BIN`), y por lo tanto habremos de evitar que quede expuesto.

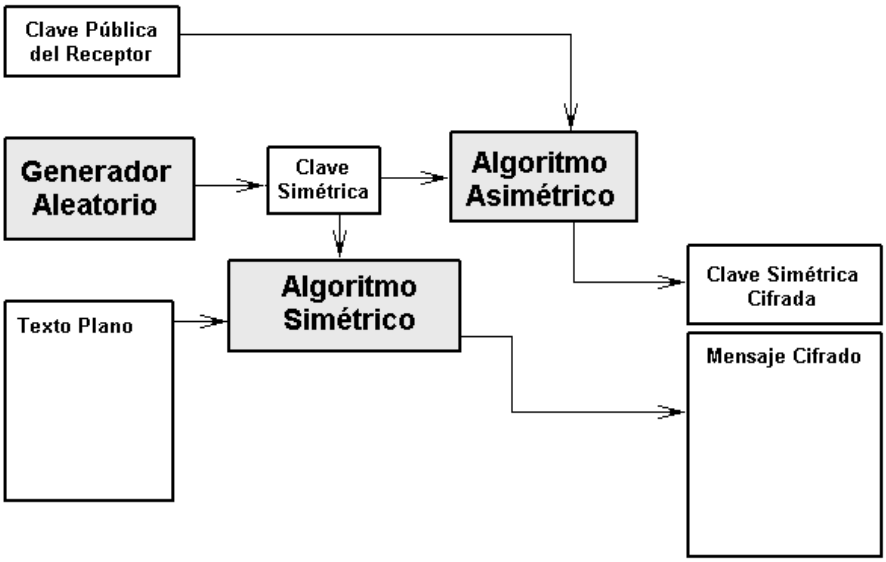

Figura 11.1: Codificación de un mensaje PGP

11.2.2 Firma Digital

En lo que se refiere a la firma digital, las primeras versiones de PGP obtienen en primer lugar la signatura MD5 (ver sección 10.1.3), que posteriormente se codifica empleando la clave privada RSA correspondiente. Las versiones actuales implementan el algoritmo DSA (sección 10.1.4).

La firma digital o signatura puede ser añadida al fichero u obtenida en otro fichero aparte. Esta opción es muy útil si queremos *firmar* un fichero ejecutable, por ejemplo.

11.2.3 Armaduras ASCII

Una de las funcionalidades más útiles de PGP consiste en la posibilidad de generar una *armadura ASCII* para cualquiera de sus salidas. Obviamente, todas las salidas de PGP (mensajes codificados, claves públicas extraidas de algún anillo, firmas digitales, etc.) consisten en secuencias binarias, que pueden ser almacenadas en archivos. Sin embargo, en la mayoría de los casos puede interesarnos enviar la información mediante correo electrónico, o almacenarla en archivos de texto.

Recordemos que el código ASCII es de 7 bits, eso quiere decir que los caracteres situados por encima del valor ASCII 127 no están definidos, y de hecho diferentes computadoras y sistemas operativos los interpretan de manera distinta. También hay que tener en cuenta que entre esos 128 caracteres se encuentran muchos que representan códigos de control, como el retorno de

carro, el fin de fichero, el tabulador, etc. La idea es elegir 64 caracteres *imprimibles* (que no sean de control) dentro de esos 128 caracteres ASCII. Con esos 64 caracteres podremos representar exactamente 6 bits, por lo que una secuencia de tres bytes (24 bits) podrá representarse mediante cuatro de estos caracteres. Esta cadena de caracteres resultante se *trocea* colocando en cada línea un número razonable de símbolos, por ejemplo 72. El resultado es una secuencia de caracteres que pueden ser tratados como texto estándar y, por tanto, manipulados en cualquier editor de texto y, por supuesto, enviados por correo electrónico.

Como ejemplo incluyo mi clave pública PGP en formato ASCII:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: PGPfreeware 5.5.3i for non-commercial use

mQGiBDRkk6kRBADKYHrNnFeXlggr14IVGy6FudLG2Cd1wb3yKOaNnodyjZa0a5oi
Ls9jDfDfEdq8K+W6QBLv06w7oVFPNMYsU+ufb0pa/bHWq6IrHxKkTVH4o4PUYTmH
W0jfGjoXEtAUZ0vp9wYR0Yqi7wXO3L/N5KuVNjLj7rXOT7rOmHsOjmY1cQCg//2w
OcyAnkaDCODFNif/VdowntcD/j5midszzU6M7BWmeDJoqEEGzSuxfmRSNyNZe6/6
5k8TFXIVpB0vnxwsZSh0POSlNgz1cmX6VbEmmUXoYsMRfq7iXHSAZ3DLB333yR2b
QUbkrH5WZF75G2vvTO7rKS5KtmROJ8E+vX/py6PGz1f3tBZJ94KwM787g6j43F4X
IYTAA/9L5GZzClHOGt01BtZkioH5YoHnDGHKC8mMXcykXA5KdJvl+9jGz3InUHiG
04StaMxMcDcWLzL5FVLz3LBzlOXGs7jikgH3BYBI3p7dIExfRADucDHyKL/CpIl5
zqHBI+5bxY3Tysu3UlA1UkQloJMsSInlkkjQhwihNYsj8Avr9LQsTWFudWVsIEx1
Y2VuYSBMb3BleiA8bWx1Y2VuYUBhcG9sby51amFlbi5lcz6JAEsEEBECAAsFAjRk
k6kECwMBAgAKCRBIslFhaatXhPnAAJsEyj9JCmn043BtdBprjIWcJuXtrQCg9bF7
21L0JLqEU6M6zzmPT+hmkVyJAD8DBRA0cv9B4JPOvVv6kF8RAkTnAKDPxk/I+DSb
iHaVpmtXQJpTT9gXxQCfe9BiiiWXte5XLLi6BluP60pcA5S0Jk1hbnVlbCBMdWNl
bmEgTG9wZXogPG1sdWNlbmFAdWphZW4uZXM+iQBLBBARAgALBQI02uBaBAsDAQIA
CgkQSLJRYWmrV4TvcgCfbyNUFooSCjFGuIN0vLdxGWfUl8YAmgI35x8fCmsspZul
1447AuoiSgZmuQINBDRklF8QCADD+IZtNb05pPJ9XhE1l2DUdEMYTJqxl0Q8Iz2K
E+R+9DIrjicfXc6x5qKQhijKgD8EWJZ5M1we9v/go64IrYac7+n/qiEVwA4rlqcT
KAauFRS21TDyRhnStQEUgme6u0NzHBrC6Bs8W/eoD6yfvnqrtdTGOdO2mi76VbKP
pWQ38S+tkN/0TuKQ9SpG4iwfMnk9TDKla1doqDYd8KUqRmVgWrN08MKyytc5+NkP
HaroZJ6TpKCPk+HdfiimhcjHoJ+angQtia92acO7ih7oDqX9chDw02GP4bF206BO
rxAoNFaXZ/ozelCA+djMEegMiqev+J7yMrix4N7NE+0vX9OVAAICB/4lfDTdDWik
sEBo79ZmVyzZKITOq/6AKQVb2+TZsltJjg9AfCItzQfBw8QOLuMi71DEl9lweRrH
xeBeQK98UkUWaqCpqsg7Nen8GCJjKZroZUCZdwjqUj1lKDM++9o0UZ7T/tHpiuh5
7ucn/1HLEXz1Ij77m1lj79uku4oBudf1IDc8SdRaIcPZqHdwLdVc2u6zeWwoDswS
Her3GegX/xnu/CV2spTItl1oOpIzuKzIjkV7Q9af+a1bj7v+uNuhpjs8YkICq2fx
wcO8HJFgJnyQtGWhtAfMvxc3V+x18G/pYM15L0ioEJdw8wTtHbXDK2yLZYwOdgvu
tBZT8KAuoXHYiQA/AwUYNGSUX0iyUWFpq1eEEQL3JACfTfvh6A70A9N2SbnRBmkt
uRBp9NsAn2ZQbpg0eaeVRuzejA2QM7ldrz53
=NZ+7
-----END PGP PUBLIC KEY BLOCK-----
```

Como puede verse, los únicos símbolos empleados son las letras mayúsculas y minúsculas,

los números, y los signos '/'y '+'; el resto de símbolos y caracteres de control simplemente será ignorado. Cualquiera podría copiar esta clave pública a mano (¡!) o emplear un OCR para introducirla en su anillo de claves correspondiente, aunque es mejor descargarla a través de Internet.

11.2.4 Gestión de Claves

PGP, como ya se ha dicho, almacena las claves en unas estructuras denominadas *anillos*. Un anillo no es más que una colección de claves, almacenadas en un fichero. Cada usuario tendrá dos anillos, uno para las claves públicas (`PUBRING.PKR`) y otro para las privadas (`SECRING.SKR`).

Cada una de las claves, además de la secuencia binaria correspondiente para el algoritmo concreto donde se emplee, posee una serie de datos, como son el identificador del usuario que la emitió, la fecha de expiración, la versión de PGP con que fue generada, y la denominada huella digital (*fingerprint*). Este último campo es bastante útil, pues se trata de una secuencia hexadecimal lo suficientemente larga como para que sea única, y lo suficientemente corta como para que pueda ser escrita en un papel, o leida de viva voz. La huella digital se emplea para asegurar la autenticidad de una clave. Por ejemplo, la huella digital de la clave pública anterior es:

```
9E2B 9D14 CBCE FE12 16A8  C103 48B2 5161 69AB 5784
```

Si alguien quisiera asegurarse de la autenticidad de la clave anterior, bastaría con que llamara por teléfono al autor, y le pidiera que le leyera la su huella digital.

11.2.5 Distribución de Claves y Redes de Confianza

PGP, como cualquier sistema basado en clave pública, es susceptible a ataques de intermediario (sección 9.2.2). Esto nos obliga a establecer mecanismos para asegurarnos de que una clave procede realmente de quien nosotros creemos. Uno de los mecanismos que permite esto es la huella digital, pero no el único.

PGP permite a un usuario firmar claves, y de esta forma podremos confiar en la autenticidad de una clave siempre que ésta venga firmada por una persona de confianza. Hay que distinguir entonces dos tipos de confianza: aquella que nos permite creer en la validez de una clave, y aquella que nos permite fiarnos de una persona como certificador de claves. La primera se puede calcular automáticamente, en función de que las firmas que contenga una clave pertenezcan a personas de confianza, pero la segunda ha de ser establecida manualmente. No olvidemos que el hecho de que una clave sea auténtica no nos dice nada acerca de la persona que la emitió. Por ejemplo, yo puedo tener la seguridad de que una clave pertenece a una persona, pero esa persona puede dedicarse a firmar todas las claves que le llegan, sin asegurarse de su autenticidad, por lo que en ningún caso merecerá nuestra confianza.

11.2.6 *Otros* PGP

La rápida popularización de PGP entre ciertos sectores de la comunidad de Internet, y la disponibilidad del código fuente, han hecho posible la proliferación de variantes más o menos complejas de PGP. Muchas de ellas simplemente implementaban claves de mayor longitud (como PGPg), y otras corresponden a proyectos tan ambiciosos como GNU-PGP, que se basa únicamente en algoritmos de libre distribución.

11.3 Vulnerabilidades de PGP

Según todo lo dicho hasta ahora, parece claro que PGP proporciona un nivel de seguridad que nada tiene que envidiar a cualquier otro sistema criptográfico jamás desarrollado. ¿Qué sentido tiene, pues, hablar de sus *vulnerabilidades*, si éstas parecen no existir?

Como cualquier herramienta, PGP proporcionará un gran rendimiento si se emplea correctamente, pero su uso inadecuado podría convertirlo en una protección totalmente inútil. Es por ello que parece interesante llevar a cabo una pequeña recapitulación acerca de las *buenas costumbres* que harán de PGP nuestro mejor aliado.

- *Escoger contraseñas adecuadas.* Todo lo comentado en la sección 10.3 es válido para PGP.
- *Proteger adecuadamente los archivos sensibles.* Estos archivos serán, lógicamente, nuestros llaveros (anillos de claves) y el fichero que alberga la semilla aleatoria. Esta protección debe llevarse a cabo tanto frente al acceso de posibles curiosos, como frente a una posible pérdida de los datos (¡recuerde que si pierde el archivo con su clave privada no podrá descifrar jamás ningún mensaje!).
- *Emitir revocaciones de nuestras claves al generarlas* y guardarlas en lugar seguro. Serán el único mecanismo válido para revocar una clave en caso de pérdida del anillo privado. Afortunadamente, la versión 6 de PGP permite nombrar *revocadores* para nuestras claves, de forma que éstos podrán invalidarla en cualquier momento sin necesidad de nuestra clave privada.
- *Firmar sólo las claves de cuya autenticidad estemos seguros.* Es la única manera de que las redes de confianza puedan funcionar, ya que si todos firmáramos las claves alegremente, podríamos estar certificando claves falsas.

Parte V

Seguridad en Redes de Computadores

Capítulo 12

Seguridad en Redes

La rápida expansión y popularización de Internet ha convertido a la seguridad en redes en uno de los tópicos más importantes dentro de la Informática moderna. Con tal nivel de interconexión, los virus y los *hackers* campan a sus anchas, aprovechando las deficientes medidas de seguridad tomadas por administradores y usuarios a los que esta nueva revolución ha cogido por sorpresa.

Las ventajas de las redes en Informática son evidentes, pero muchas veces se minusvaloran ciertos riesgos, circunstancia que a menudo pone en peligro la seguridad de los sistemas. En unos pocos años la inmensa mayoría de las empresas operarán a través de la Red, y esto sólo será posible si los profesionales de la Informática saben aportar soluciones que garanticen la seguridad de la información.

12.1 Importancia de las Redes

La Informática es la ciencia del tratamiento automático de la información, pero tanto o más importante que su procesamiento y almacenamiento es la capacidad para poder transmitirla de forma eficiente. La información tiene un tiempo de vida cada vez menor y la rapidez con la que pueda viajar es algo crucial. Los últimos avances en compresión y transmisión de datos digitales permiten hoy por hoy transferir cantidades enormes de información a velocidades que hace tan solo unos años eran impensables. En este sentido las redes de computadoras desempeñan un papel fundamental en la Informática moderna.

Pero hemos de tener en cuenta que la complejidad de las grandes redes y su carácter público convierte la protección física de los canales de comunicación en algo tremendamente difícil. Hemos de depositar nuestra confianza en la Criptografía para garantizar la confidencialidad en las comunicaciones.

Uno de los mayores obstáculos que han tenido que superarse para que las redes pudieran desarrollarse, ha sido encontrar *lenguajes* comunes para que computadoras de diferentes tipos

pudieran *entenderse*. En este sentido el protocolo TCP/IP se ha erigido como estándar *de facto* en la industria de la Informática. En general todas las redes de computadoras se construyen conceptualmente sobre diferentes capas de abstracción, que desarrollan tareas distintas y proporcionan un protocolo unificado a las capas superiores. La Criptografía podrá entonces ser empleada en diferentes niveles de abstracción. Por ejemplo, podemos codificar un fichero antes de transmitirlo por la red, lo cual correspondería al nivel de abstracción mayor, o podemos enviarlo sin codificar, pero a través de un protocolo de bajo nivel que cifre cada uno de los *paquetes* de información en los que se va a subdividir el fichero en el momento de transmitirlo.

En función del tipo de red con el que trabajemos nos enfrentaremos a diferentes tipos de riesgos, lo cual nos conducirá inevitablemente a medidas de diferente naturaleza para garantizar la seguridad en las comunicaciones. En este capítulo haremos una breve reflexión sobre algunos de los casos que pueden darse, sin tratar de ser exhaustivos (sería imposible, dada la inmensa cantidad de posibilidades). Nuestro objetivo se centrará en aportar una serie de directrices que nos permitan analizar cada situación y establecer una correcta política de protección de la información.

Ya que no existe una solución universal para proteger una red, en la mayoría de los casos la mejor estrategia suele consistir en tratar de *colarnos* nosotros mismos para poner de manifiesto y corregir posteriormente los *agujeros de seguridad* que siempre encontraremos. Esta estrategia se emplea cada vez con mayor frecuencia, y en algunos casos hasta se contrata a *hackers* para que impartan cursillos de seguridad a los responsables de las redes de las empresas.

12.2 Redes Internas

El caso más sencillo de red que nos podemos encontrar es local (LAN), con todos los computadores interconectados a través de unos cables de los que también se es propietario. Esta última circunstancia nos va a permitir ejercer un control total sobre el canal de comunicaciones, pudiendo protegerlo físicamente, lo cual evita prácticamente cualquier riesgo de falta de privacidad en la información.

Uno de los riesgos dignos de mención en estos casos son las posibles pérdidas de información debidas a fallos físicos, que pueden ser minimizados llevando a cabo una adecuada política de *copias de respaldo*, que deberán ser confeccionadas periódicamente, almacenadas en un lugar diferente de aquel donde se encuentra la red, y protegidas adecuadamente contra incendios y accesos no deseados.

Otro riesgo que se da en las redes locales, a menudo infravalorado, es el que viene del uso inadecuado del sistema por parte de los propios usuarios. Ya sea por mala fe o descuido, un usuario con demasiados privilegios puede destruir información, por lo que estos permisos deben ser asignados con mucho cuidado por parte de los administradores. Esta circunstancia es muy importante, ya que, sobre todo en pequeñas empresas, el dueño muchas veces cree que debe conocer la clave del administrador, y luego es incapaz de resistir la tentación de *jugar con ella*, poniendo en serio peligro la integridad del sistema y entorpeciendo el trabajo del superusuario.

Existen redes internas en las que un control exhaustivo sobre el medio físico de transmisión de datos es en la práctica imposible. Piénsese en un edificio corporativo con un acceso no muy restringido, por ejemplo un aulario de una universidad, que posee conexiones *ethernet* en todas sus dependencias. En principio, nada impediría a una persona conectar un ordenador portátil a una de esas conexiones para llevar a cabo un análisis del tráfico de la red sin ser descubierta, o *suplantar* a cualquier otro computador. En estos casos será conveniente llevar a cabo algún tipo de control, como la deshabilitación dinámica de las conexiones de red no utilizadas en cada momento, la verificación del identificador único de la tarjeta de red concreta que debe estar conectada en cada punto, o la adopción de protocolos de autentificación de las computadoras dentro de la red, como por ejemplo *kerberos* (que forma parte del proyecto *Athena*, en el MIT).

12.3 Redes Externas

Consideraremos red externa a aquella que en todo o en parte se apoye en un canal físico de comunicación ajeno. Existirán redes externas de muy diferentes tipos, pero todas ellas tienen en común la característica de que que en algún momento la información viaja por canales sobre los que no se tiene ningún tipo de control. Todas las técnicas que nos va a permitir llevar a cabo protecciones efectivas de los datos deberán hacer uso necesariamente de la Criptografía.

Para identificar los posibles riesgos que presentará una red externa, hemos de fijarnos en cuestiones tan dispares como el sistema operativo que corre sobre los ordenadores o el tipo de acceso que los usuarios *legales* del sistema pueden llevar a cabo.

Una de las configuraciones más comunes consiste en el uso de una red local conectada al exterior mediante un *cortafuegos* (computadora que filtra el tráfico entre la red interna y el exterior). Los cortafuegos son herramientas muy poderosas si se emplean adecuadamente, pero pueden entrañar ciertos riesgos si se usan mal. Por ejemplo, existen muchos lugares donde el cortafuegos está conectado a la red local y ésta a su vez a la red externa (ver figura 12.1, caso A). Esta configuración es la más sencilla y barata, puesto que sólo necesitamos una tarjeta de red en el cortafuegos, pero no impediría a un computador situado en el exterior acceder directamente a los de la red local. La configuración correcta se puede apreciar en el caso B de la figura 12.1, donde la red externa (y todos sus peligros) está separada físcamente de la red local.

Podemos distinguir dos grandes tipos de peligros potenciales que pueden comprometer nuestra información desde una red externa:

- Ataques indiscriminados. Suelen ser los más frecuentes, y también los menos dañinos. Dentro de esta categoría podemos incluir los troyanos y los virus, programas diseñados normalmente para *colarse* en cualquier sistema y producir efectos de lo más variopinto. Precisamente por su carácter general, existen programas específicos que nos protegen de ellos, como los antivirus. Conviene disponer de un buen antivirus y actualizarlo periódicamente.
- Ataques *a medida*. Mucho menos comunes que los anteriores, y también más peligrosos,

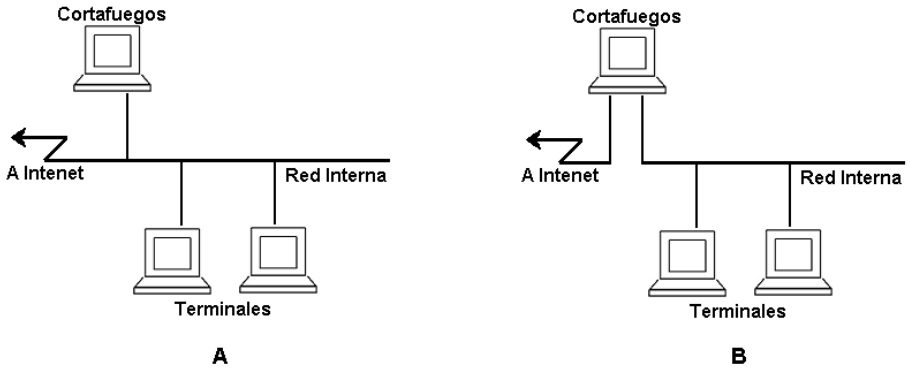

Figura 12.1: **A**: Configuración incorrecta, el cortafuegos tiene una única tarjeta de red, y los terminales están conectados físicamente a la red externa. **B**: Configuración correcta, el cortafuegos dispone de dos tarjetas de red y el resto de las computadoras está aislado físicamente de la red externa.

son los ataques que generalmente llevan a cabo los *hackers*. En estos casos las víctimas son casi siempre grandes corporaciones, y muchas veces la información ni siquiera es destruida o comprometida, puesto que los *hackers* sólo persiguen enfrentarse al reto que supone para ellos entrar en un sistema *grande*. El problema es que para borrar sus huellas y dificultar el rastreo de sus acciones, suelen atacar en primer lugar sistemas pequeños para desde ellos cometer sus *travesuras*, lo cual convierte a cualquier sistema en potencial víctima de estos personajes. Lo que ocurre en la mayoría de los casos es que su necesidad de emplear sistemas pequeños como plataforma les obliga a no dañarlos, para no dejar ningún tipo de rastro que permita localizarlos posteriormente. Por desgracia, no existe otro sistema para protegerse de los *hackers*, más que la vigilancia constante.

En cuanto a la protección de las comunicaciones en sí, baste decir que existen protocolos de comunicación segura de *bajo nivel*, como el SSL (Secure Sockets Layer), que permite establecer comunicaciones seguras a través de Internet, haciendo uso de algoritmos simétricos y asimétricos simultáneamente. Este protocolo es transparente y puede correr bajo otros protocolos ampliamente conocidos, como POP3, TELNET, FTP, HTTP, etc. De hecho, gran cantidad de aplicaciones emplean protocolos de este tipo en sus comunicaciones. Desgraciadamente, las restrictivas leyes norteamericanas en cuanto a la exportación de material criptográfico hacen que la gran mayoría de las aplicaciones *seguras* que se venden fuera de los EE.UU. y Canadá estén en realidad debilitadas, por lo que hemos de informarnos muy bien antes de depositar nuestra confianza en ellas.

12.3.1 Intranets

El término *intranet* se ha popularizado recientemente y hace alusión a redes externas que se comportan de cara a los usuarios como redes privadas internas. Obviamente, este tipo de redes ha de ser implementado haciendo uso de protocolos criptográficos de autentificación y codificación de las transmisiones, puesto que el tráfico que nosotros vemos como *interno* a nuestra red, en realidad viaja por Internet.

12.4 Conclusiones

Después de todo lo dicho parece una locura conectarse a una red externa, y ciertamente lo es si no se toman las precauciones adecuadas. La cantidad de posibles riesgos es enorme, y con toda seguridad en el futuro aparecerán nuevos peligros, pero no olvidemos que ante todo debemos ser racionales. Si bien puede ocurrir que un equipo de *hackers* trate de entrar en nuestro sistema, esta posibilidad suele ser remota en la mayoría de los casos, debido precisamente al escaso interés va a despertar en los *hackers* penetrar en una red pequeña. Por otro lado hay que tener en cuenta que cierto tipo de ataques requiere fuertes inversiones, por lo que si nuestra información no resulta realmente valiosa para el atacante, podemos considerarnos a salvo. No olvidemos que el coste de la protección en ningún caso puede superar el valor de la propia información que se desea proteger. Por lo demás, parece claro que las ventajas que nos proporcionará estar en la Red son claramente mayores que los inconvenientes, pero nunca se debe bajar la guardia.

En general, conviene estar preparado para el peor de los casos, que suele ser la pérdida de la información, casi siempre debida a fallos físicos o a la presencia de virus. En cuanto al resto de posibilidades, será suficiente con la adopción de protocolos seguros, además de llevar un registro de todas las operaciones que tienen lugar dentro del sistema, registro que deberá ser controlado periódicamente para detectar posibles anomalías. Otra práctica bastante recomendable consiste en mantenerse al día sobre los fallos de seguridad detectados en los programas y sistemas operativos que empleemos, así como de los sucesivos *parches* que las empresas de software suelen distribuir periódicamente.

Bibliografía

[1] Bruce Schneier. *Applied Cryptography. Second Edition.* John Wiley & sons, 1996

[2] Seberry, J., Pieprzyk, J. *Cryptography. An Introduction to Computer Security.* Prentice Hall. Australia, 1989

[3] Juan Manuel Velázquez y Arturo Quirantes. *Manual de PGP 5.53i.* 1998.

[4] *RFC 2440: Open PGP Message Format.* http://www.ietf.org/rfc/rfc2440.txt

[5] http://www.kriptopolis.com

[6] http://www.pgpi.com

[7] http://www.replay.com