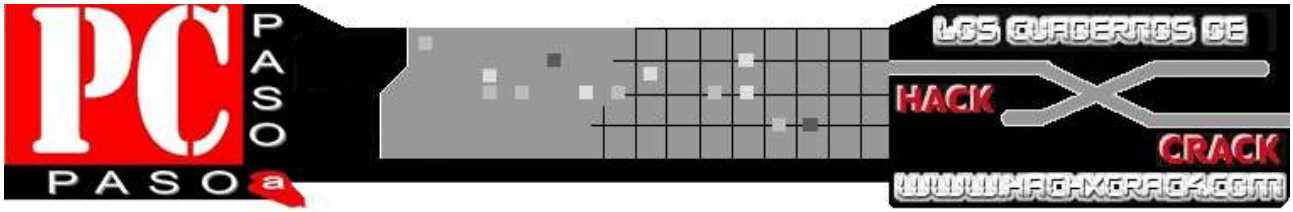


presenta...

Criptosistemas Informáticos

Autor:

Death Master

Índice de contenidos

Introducción

-¿Cuánto tiempo quieres que sean secretos esos mensajes? -le preguntó Randy en el último mensaje antes de abandonar San Francisco-. ¿Cinco años? ¿Diez años? ¿Veinticinco años?

Después de llegar al hotel esa tarde, Randy descifró y leyó la respuesta de Avi. Todavía la tiene colgada frente a los ojos, como la imagen remanente de un flash.

Quiero que sigan siendo secretos mientras los hombres sean capaces del mal.

Neal Stephenson, "Criptonomicón"

Desde los tiempos del Antiguo Egipto hasta hoy en día ha pasado mucho tiempo, pero hay algo que no ha cambiado: el anhelo del ser humano por esconder sus secretos. *Criptografía*.

Personajes como Cleopatra o El César ya aprendieron a valorar la importancia de esconder de ojos indiscretos sus mensajes. La Scilata que los espartanos usaban allá por el año 400 a.C., o el propio código César (un simple desplazamiento alfabético) fueron los inicios. Con el desarrollo de las ciencias, y más concretamente de las matemáticas, la criptografía creció como un hermano menor, de la mano. En la Edad Media comenzó a adquirir una gran importancia cuando un siervo del Papa Clemente VII escribió el primer manual sobre el tema de la historia en el viejo continente. En 1466 León Battista Alberti ideó el sistema polialfabético basado en la rotación de unos rodillos. Un siglo más tarde, Giovan Battista Belaso inventó la clave criptográfica basada en una palabra o texto que se transcribía letra a letra sobre el mensaje original...

Pero si un punto ha marcado un antes y un después en la criptografía, ese ha sido la Segunda Guerra Mundial. Ya a principios del pasado siglo *XX* se idearon los denominados "traductores mecánicos", basados en el concepto ideado en el siglo *XV* por Alberti de las ruedas concéntricas, y el sistema comenzó a tornarse sumamente complejo, ya no bastaba con analizarlo concienzudamente para comprenderlo. Y eso nos llevó a la Segunda Guerra Mundial, a los años 30 y 40 y a la máquina Enigma. Aunque mucha gente no lo sepa, la criptografía fue uno de los principales motivos para que los aliados ganaran la guerra, pues los alemanes creían el código de su máquina Enigma inviolable, y de hecho sí que era extremadamente complejo. Pero al igual que antes comentaba que el anhelo humano de esconder secretos es eterno, también lo es su anhelo por desenterrar secretos ajenos. Un equipo de criptoanalistas, matemáticos y demás mentes privilegiadas (entre ellos Alan Turing, uno de los padres de la informática) logró en 1942 lo que parecía imposible: romper el cifrado de Enigma. Para ello diseñaron las llamadas "bombas navales" (las bombes), aparatos de cálculo mecánicos que se encargaban de romper el cifrado alemán y entregar todos los secretos a los aliados. Igualmente, el Purple (versión japonesa de Enigma) fue roto en Midway por un equipo dirigido por el comandante Joseph J. Rochefort. El hecho de conocer los secretos, de tener todas las llaves (nunca mejor dicho), dio la vuelta a la guerra y cambió el curso de la historia. Y el mundo nunca volvió a ser el mismo.

El nacimiento de la informática y de los criptosistemas informático supuso un cambio radical del concepto de criptografía, y también del criptoanálisis. Los criptosistemas y los algoritmos aumentaron repentinamente y de forma descomunal su complejidad. Desde el DES hace ya mucho, hasta los criptosistemas asimétricos de curvas elípticas actuales todo ha cambiado también mucho, pero uno de esos cambios ha supuesto el segundo gran punto de inflexión de la criptografía: PGP. Hasta entonces el privilegio de guardar secretos estaba exclusivamente en manos de los gobiernos o los poderosos, y hoy día, seguramente gracias a PGP, es un derecho de cualquier ciudadano por humilde que sea (aunque en algunos países no sea así). Arriesgando mucho, Philip Zimmermann nos abrió las puertas a la criptografía, y a la libertad de comunicarnos de forma segura.

Por supuesto hay otros nombres que son capitales en el desarrollo de los criptosistemas informáticos, aparte de Zimmermann: Rivest, Shamir, Adleman, Diffie, Hellman, ElGamal, Rijmen, Daemen, Massey, Miller, Goldwaser... Todos ellos son padres de lo que hoy se llama criptografía. Gracias a todos.

Death Master

Conceptos básicos

Criptología

Criptografía, el arte de ocultar. La palabra tiene su origen en el griego: ***kryptos*** (oculto, escondido) y ***graphein*** (escribir). El arte de ocultar un mensaje mediante signos convencionales es muy antiguo, casi tanto como la escritura. Y efectivamente siempre ha sido considerado un arte hasta hace relativamente poco, cuando **Claude E. Shannon** publicó en dos años dos documentos que supusieron la fundación de la moderna **Teoría de la Información**. Esos documentos son:

- **[SHA48] C. E. Shannon, "*A mathematical theory of communication*",** Bell System Tech. J. 27 (1948), 379-423 y 623-656.
- **[SHA49] C. E. Shannon, "*Communication theory of secrecy systems*",** Bell System Tech. J. 28 (1949), 656-715.

A partir de entonces, y unido al desarrollo de la computación moderna, el desarrollo de la criptografía alcanzó nuevos horizontes.

Antes de entrar en más detalles, hay que comprender que la criptografía se complementa con otra rama de estudio, el **criptoanálisis**, que estudia el camino inverso de la criptografía, dedicando sus esfuerzos a desentrañar los secretos que la criptografía se empeña en mantener ocultos. Ambas ramas conforman lo que entendemos como **criptología**.

> ***criptografía***
> *(Del gr. κρυπτός, oculto, y -grafía).*
> 1. f. Arte de escribir con clave secreta o de un modo enigmático.
> **** FUENTE: Real Academia Española** (http://www.rae.es/)*

> ***criptoanálisis***
> *(Del gr. κρυπτός, oculto, y análisis).*
> 1. m. Arte de descifrar criptogramas.
> **** FUENTE: Real Academia Española** (http://www.rae.es/)*

Texto en claro y criptogramas

En el ámbito criptográfico, entendemos por **Texto en claro** cualquier información que resulta legible y comprensible *per se*. Un texto en claro sería cualquier información antes de ser encriptada o después de ser desencriptada. Se considera que cualquier información es vulnerable si se encuentra en este estado.

Así mismo, denominamos **criptograma** a cualquier información que se encuentre convenientemente cifrada y no resulte legible ni comprensible más que para el destinatario legítimo de la misma.

El mecanismo de transformar un texto en claro en un criptograma lo denominamos **encriptación o cifrado**, y al proceso de recuperación de la información a partir de un criptograma lo denominamos **desencriptación o descifrado**.

Es muy importante no confundir estos términos con codificación o decodificación, pues codificación es el acto de representar la información de distintas formas, pero no necesariamente encriptadas. Por ejemplo, un número decimal puede codificarse como hexadecimal, y no por ello se convierte en un criptograma.

Flujo de información

En un criptosistema la información sigue un flujo siempre fijo:

El emisor cifra el texto en claro para obtener el criptograma, que viaja por un **canal ruidoso**. El receptor descifra el criptograma y obtiene de nuevo el texto en claro que el emisor le envió. Durante toda la transmisión, el mensaje es ilegible.

Criptosistemas

Antes de nada, es conveniente definir qué entendemos -matemáticamente- por **criptosistema**. Un criptosistema es una cuaterna de elementos formada por:

- Un conjunto finito llamado **alfabeto**, que según unas normas sintácticas y semánticas, permite emitir un mensaje en claro así como su correspondiente criptograma.
- Un conjunto finito denominado **espacio de claves** formado por todas las posibles claves, tanto de encriptación como de desencriptación, del criptosistema.
- Una familia de aplicaciones del alfabeto en sí mismo que denominamos **transformaciones de cifrado**.
- Una familia de aplicaciones del alfabeto en sí mismo que denominamos **transformaciones de descifrado**.

Ya sabemos qué es un criptosistema, pero ¿qué es un criptosistema informático? Un criptosistema informático se definiría por los siguientes cuatro elementos:

- Un conjunto finito denominado **alfabeto**, que permite representar tanto el texto en claro como el criptograma. A bajo nivel hablaríamos de **bits**, y a más alto nivel podríamos hablar de caracteres **ASCII** o **MIME**.
- Un conjunto finito denominado **espacio de claves**. Estaría constituido por la totalidad de las claves posibles del criptosistema.
- Una familia de **transformaciones aritmético-lógicas** que denominamos **transformaciones de cifrado**.
- Una familia de **transformaciones aritmético-lógicas** que denominamos **transformaciones de descifrado**.

Se trata simplemente de un criptosistema adaptado a las posibilidades y limitaciones de una computadora. El alfabeto o espacio de caracteres suele ser un estándar de representación de información (típicamente MIME o UNICODE por motivos de compatibilidad) y a más bajo nivel, por bits. Las transformaciones de cifrado y descifrado se ciñen a las normas de computación de los ordenadores actuales. En realidad, a efectos prácticos no existe mucha diferencia entre criptosistema matemático e informático, pues los matemáticos suelen diseñarse pensando en representaciones computacionales (pues solamente los ordenadores tienen la potencia necesaria para soportar los complejos algoritmos), y los informáticos se desarrollan siempre con una base matemática.

Algoritmo criptográfico

Comenzaremos definiendo el término algoritmo por si algún despistado no sabe qué es.

> ***algoritmo***
> *(Quizá del lat. tardío *algobarismus, y este abrev. del ár. clás. hisābu lgubār, cálculo mediante cifras arábigas).*
> 1. m. Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
> 2. m. Método y notación en las distintas formas del cálculo.
> **** FUENTE: Real Academia Española** (http://www.rae.es/)*

Ahora veamos la definición de algoritmo que da un manual de programación algorítmica:

> ***algoritmo***
> Descripción precisa de una sucesión de instrucciones que permiten llevar a cabo un trabajo en un número finito de pasos.
> **** FUENTE:* J. Castro, F. Cucker, X. Messeguer, A. Rublo, L. Solano, B. Valles*, "Curso de Programación",** *McGraw Hill. (1993), 2. ISBN 84-481-1959-2*

Así pues, un algoritmo debe describir de forma unívoca y sin dar lugar a interpretaciones, la solución a un problema en un numero de pasos concreto. Ya podemos deducir qué es un algoritmo de encriptación: una descripción unívoca y concreta de cómo funciona un criptosistema determinado.

Clave criptográfica

El concepto de clave criptográfica surge con el propio concepto de la criptografía, y es el alma de un algoritmo de encriptación.

Obviamente un algoritmo tiene que poseer la capacidad de ser usado muchas veces sin que su mecanismo sea idéntico, pues de lo contrario cada persona debería tener su propio algoritmo de encriptación. Para implementar esta funcionalidad, se usan las claves. La clave es un dato que interviene de forma activa en la ejecución del algoritmo y lo personaliza.

Atendiendo únicamente al tipo de clave, podemos distinguir dos criptosistemas:

Sistemas de clave única o criptosistemas simétricos: Son aquellos en los que los procesos de cifrado y descifrado son llevados a cabo por una única clave.

Sistemas de clave pública o criptosistemas asimétricos: Son aquellos en los que los procesos de cifrado y descifrado son llevados a cabo por dos claves distintas y complementarias.

Ejemplo:

*Si usamos un sistema de criptografía simétrica por sustitución, con una **clave x=3**...*

A	***B***	***C***	***D***	***E***	***F***	***G***	***H***	***I***	***J***	***K***	***L***	***M***	***N***	***Ñ***	***O***	***P***	***Q***	***R***	***S***	***T***	***U***	***V***	***W***	***X***	***Y***	***Z***	
D	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*	*Ñ*	*O*	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*		*A*	*B*	*C*

De esta forma la frase:

all your base are belong to us

Pasaría a ser:

dññcarxucedvhcduhcehñrpjcwrcxv

Longitud de clave

A día de hoy, gracias a los ordenadores y los superordenadores (típicamente clusters) es posible realizar complejos cómputos matemáticos en un espacio de tiempo relativamente corto. Así pues, el campo del criptoanálisis está íntimamente ligado a esta potencia de cálculo.

Para que un algoritmo se considere seguro, su criptoanálisis sin la clave necesaria debe ser **computacionalmente imposible** de resolver. Consideramos "imposible" de resolver un sistema cuya vulneración necesite mayores recursos (económicos o en tiempo) que el beneficio reportado. Por ejemplo, ser consideraría seguro un criptosistema que requisiera miles de años para ser descifrado.

Para lograr semejante complejidad, se tratan números o conjuntos de números enormes. Es obvio que a mayor tamaño de estos números, existe un mayor número de claves posibles, y la posibilidad de éxito del criptoanálisis es menor.

Hoy en día este tamaño se denomina **longitud de clave**, y se mide típicamente en los **bits** que ocupa la clave. Así, una clave de un número de 1024 bits sería un número cualquiera desde el 0 hasta el 1,8 * 10^308 (2^1024). Al representar las longitudes de clave como potencias de dos, es importante darse cuenta de la relación existente entre las longitudes de clave. Una clave de 1025 bits es el doble de larga que una de 1024 (2^1025 frente a 2^1024).

A día de hoy se usan claves que oscilan entre los 512 bits y los 4096 bits de longitud.

Algoritmos simétricos

Son los criptosistemas más sencillos. Se trata de algoritmos que trabajan con una única clave de doble función (cifrado y descifrado). Dentro de los sistemas simétricos distinguimos dos tipos de algoritmos: los de **cifrado de bloque**, que dividen el texto en claro en bloques de tamaño prefijado (por ejemplo 64 bits) y los cifran bloque a bloque; y los de **cifrado de flujo**, que cifran bit a bit o byte a byte.

A continuación vemos algunos de los principales algoritmos criptográficos simétricos:

DES:

Data Encryption Standard (1976)

RFC's relacionados

#1829 *- "The ESP DES-CBC Transform" - ftp://ftp.rfc-editor.org/in-notes/rfc1829.txt*
#2952 *- "Telnet Encryption: DES 64 bit Cipher Feedback" - ftp://ftp.rfc-editor.org/in-notes/rfc2952.txt*
#2953 *- "Telnet Encryption: DES 64 bit Output Feedback" - ftp://ftp.rfc-editor.org/in-notes/rfc2953.txt*

Descripción

Algoritmo simétrico de cifrado en bloques de 64 bits basado en **LUCIFER** (criptosistema interno de IBM). Fue ideado por **IBM** y aceptado por el **NIST (National Institute of Standars and Technology)** en 1976. Se trata de un algoritmo de **64 bits** de clave de los cuales 56 bits componen la clave de cifrado propiamente dicha, mientras los 8 restantes son de paridad y se usan para corrección de errores.

DES actualmente ya no es estándar criptográfico y **fue roto en Enero de 1999** con un sistema de cómputo que analizaba 250.000.000.000 claves por segundo.

Su principal ventaja es la rapidez de cálculo y la sencillez de su implementación.

Sus principales defectos son la poca longitud de clave que maneja, unido a la incapacidad de manejar claves de longitud variable; y su debilidad en un uso continuado de la misma clave, pues si se disponen de suficientes criptogramas, mediante criptoanálisis diferencial es posible romper la clave en 2^47 iteraciones.

Triple-DES:

Triple - Data Encryption Standard (1995)

RFC's relacionados

#3217 - "Triple-DES and RC2 Key Wrapping" - ftp://ftp.rfc-editor.org/in-notes/rfc3217.txt
#3537 - "Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key" - ftp://ftp.rfc-editor.org/in-notes/rfc3537.txt

Descripción

Dada la capacidad de cómputo actual y la relativa facilidad que supone romper el algoritmo DES, se desarrolló un sistema de **triple aplicación al algoritmo DES**, con tres claves diferentes para aplicar sucesivamente (en realidad se usa una clave externa dividida para aplicación intermedia dado que DES matemáticamente no es grupo, y su aplicación repetida ocasionaría un aumento efectivo de tamaño).

Mediante este sistema se obtiene un cifrado de **192 bits** (168 efectivos y 24 de paridad) con tres claves que resulta mucho más complejo de vulnerar.

AES (Rijndael):

Advanced Encryption Standard (2000)

RFC's relacionados

#3268 - "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)" - ftp://ftp.rfc-editor.org/in-notes/rfc3268.txt
#3394 - "Advanced Encryption Standard (AES) Key Wrap Algorithm" - ftp://ftp.rfc-editor.org/in-notes/rfc3394.txt
#3565 - "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)" - ftp://ftp.rfc-editor.org/in-notes/rfc3565.txt
#3686 - "Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)" - ftp://ftp.rfc-editor.org/in-notes/rfc3686.txt

Descripción

En 1977 el NIST organizó un concurso a nivel mundial para crear un sistema que cumpliera los requisitos de seguridad necesarios para convertirse en estándar a nivel mundial. Este estándar se denomina **AES (Advanced Encryption Standard)** y el algoritmo **AEA (Advanced Encryption Algorithm)**.

Muchos expertos presentaron sus propuestas, pero el ganador fue el conocido como **algoritmo Rijndael**, creado por los belgas **Vincent Rijmen** y **Joan Daemen**. Se trata de un algoritmo simétrico que puede funcionar mediante cifrado en bloques de longitud variable o en flujo y que se sirve de claves de longitud variable (**128, 192 ó 256 bits**).

Su algoritmo puede resumirse de la siguiente forma:

A	Toma el texto en claro y lo cifra en bloques para obtener el punto intermedio llamado **Estado**, que se representa como una matriz de bytes de cuatro filas.
B	A partir del **Estado**, se realizan las siguientes operaciones en forma de bucle durante un número determinado de iteraciones.
b1	Sustitución de bytes **no lineal**, operando independientemente sobre cada uno de los bytes del **Estado**.
b2	Desplazamiento de las filas del Estado cíclicamente con **offsets diferentes**.
b3	Mezcla de columnas, que se realiza multiplicando las columnas del Estado módulo ***x*4+1***, consideradas como polinomios en ***GF(28)***, por un polinomio fijo ***c(x)***.
b4	Adición de la clave de vuelta, en la que se aplica al Estado por medio de un simple **XOR**. La clave de cada vuelta se deriva de la clave de cifrado mediante el esquema de clave.
C	El esquema de clave consiste en dos operaciones, expansión de clave y selección de clave de vuelta de cifrado, y el proceso de cifrado consta de tres pasos: una adición inicial de la clave de vuelta, n-1 vueltas de cifrado y una vuelta final.

**** FUENTE: HTML Web, Seguridad, Criptografía** (http://www.htmlweb.net/)*

Se puede encontrar información mucho más detallada de la descripción algorítmica de AES en el siguiente enlace: http://home.ecn.ab.ca/~jsavard/crypto/co040401.htm.

Hoy en día AES es, efectivamente, el estándar en cifrado simétrico, siendo usado por sistemas tan populares como **Pretty Good Privacy (PGP)**.

IDEA:

International Data Encryption Algorithm (1990)

RFC's relacionados

#3058 - "Use of the IDEA Encryption Algorithm in CMS" - ftp://ftp.rfc-editor.org/in-notes/rfc3058.txt

Descripción

IDEA fue creado en 1990 por **Xuejia Lai y L. Massey**. Se trata de un algoritmo simétrico de cifrado en bloques de 64 bits. Su funcionamiento se basa en operaciones sencillas como multiplicaciones de enteros, sumas y XOR. IDEA trabaja con claves de **128 bits** de longitud.

Es importante reseñar que IDEA es un algoritmo que si bien es de uso libre para fines no comerciales, sí que está cubierto por patentes, concretamente:

- **USA y Canadá** - Patente **5.214.703** - Expira el 25 de Mayo de 2010.
- **Europa**(*) - Patente **0482154** - Expira el 16 de Mayo de 2011.
  - (*): *Austria, Francia, Alemania, Italia, Holanda, España, Suecia, Suiza, Reino Unido.*
- **Japón** - Patente **JP3225440B2** - No hay información de la fecha de expiración.

Precisamente a causa de esta patente, el sistema gnuPG no lo incluye como algoritmo de cifrado simétrico (a pesar de que PGP sí lo hace), siendo necesario añadirlo mediante un parche al software.

RC6:

Rivest Cypher 6 (1998)

RFC's relacionados

#2040 - "The RC5, RC5-CBC, RC5-CBC-Pad, and RC5-CTS Algorithms" - ftp://ftp.rfc-editor.org/in-notes/rfc2040.txt

Descripción

RC6 fue uno de los finalistas del concurso AES del NIST. Fue ideado por el matemático Ron Rivest (autor de muchos y famosos algoritmos criptográficos) en 1998. El algoritmo es propiedad de la empresa ***RSA Security***.

RC6 supone la evolución de los algoritmos RC, RC2, RC4 y RC5. Concretamente, se trata de una adaptación del sistema RC5 para cumplir los requisitos del concurso AES. Se trata de un algoritmo simétrico de cifrado de flujo con claves de longitud variable **entre 40 y 2040 bits**, siendo por defecto 128.

Su algoritmo se basa en una mezcla de sumas, restas, multiplicaciones, XOR y rotaciones, y se puede encontrar una amplia descripción, así como un ejemplo de su implementación en el siguiente enlace: http://www.codeproject.com/cpp/hexenc.asp?print=true.

Twofish:

Twofish (1998)

Descripción

Twofish es un algoritmo simétrico de cifrado en bloques de 64 bits. Supone la evolución natural de su predecesor, Blowfish, y fue otro de los cinco finalistas del concurso AES del NIST. El algoritmo es **totalmente libre**, tanto de patentes como de copyright.

Twofish maneja claves de **128, 192 ó 256 bits** y es muy potente a la par que extremadamente sencillo de implementar y rápido (18 ciclos por byte en una arquitectura Pentium I, más rápido que RC5, IDEA, DES, Triple DES, Serpent, Square, Cast-128, Feal-32, etc). Estas características le han llevado a ser incluido en el estándar SSH (Secure Shell).

Una descripción precisa del algoritmo y su implementación puede ser encontrada en el siguiente enlace: http://home.ecn.ab.ca/~jsavard/crypto/co040402.htm.

MARS:

MARS (1998)

Descripción

MARS es otro de los finalistas del concurso del NIST. Fue ideado por IBM como un algoritmo simétrico de cifrado en bloques de 128 bits. Su longitud de clave es variable, trabajando con claves de **128 a 448 bits**.

Comparado con el anterior gran algoritmo de IBM, el DES-3, la complejidad y potencia es mucho mayor, a la vez que es mucho más rápido de ejecutar (65 Mbit/s en un Pentium-Pro).

Después de no haber sido elegido como ganador del concurso AES, IBM liberó el algoritmo MARS para uso público, aunque no es muy usado en la actualidad.

En el libro "MARS - a candidate cipher for AES", publicado por IBM Corporation en Septiembre de 1999, podemos encontrar una descripción completa del algoritmo y sus diversas implementaciones: http://www.research.ibm.com/security/mars.pdf.

CAST-256:

Carlisle Adams - Stafford Tavares - 256 (1997)

RFC's relacionados

#2612 - "The CAST-256 Encryption Algorithm" - ftp://ftp.rfc-editor.org/in-notes/rfc2612.txt

Descripción

CAST-256 fue desarrollado a partir del algoritmo CAST-128. Se trata de un algoritmo simétrico de cifrado en bloques de 128 bits que maneja claves de **128, 160, 192, 224, ó 256 bits**. Su funcionamiento es muy complejo, combinando el sistema de permutación-rotación similar al utilizado por DES con otros sistemas más complejos, como combinaciones de XOR o uso de **f-funciones** (de tres formas distintas y excluyentes).

En el RFC citado arriba se puede encontrar una descripción detallada de su funcionamiento e implementación.

Algoritmos hash

Los criptosistemas de resumen, conocidos familiarmente como funciones o algoritmos hash, constituyen un tipo especial de criptosistemas. Muchos manuales de criptografía los sitúan como un subgrupo de los criptosistemas simétricos, pero a mí me gusta considerarlos como un grupo independiente debido a sus características especiales.

Para empezar, en los algoritmos hash no existe el concepto de clave criptográfica, ni tampoco el concepto de descifrado; el concepto de algoritmo criptográfico se mantiene; y surge un nuevo concepto denominado **fingerprint**, **huella digital**, **resumen** ó **hash**.

Así pues, un algoritmo tipo hash acepta como entrada un mensaje de longitud arbitraria, y tras efectuar sobre él los cálculos determinados por el algoritmo, devuelve una cadena de caracteres que representa el hash del mensaje al que aplicamos el algoritmo. Este hash **NO** puede ser denominado criptograma dado que no es posible el proceso de descifrado que nos devolvería el mensaje original.

Para que quede más claro, vamos a ver las características que definen a los criptosistemas de tipo hash:

- **Unidireccional**: Conocido un hash, es computacionalmente imposible la reconstrucción del mensaje original.
- **Compresión**: A partir de un mensaje de cualquier longitud se obtiene un hash de un tamaño fijo, normalmente menor que el del mensaje original.
- **Difusión**: El resumen es una función compleja de todos los bits del mensaje.
- **Colisión simple**: Se conoce como resistencia débil a las colisiones el hecho de que dado un mensaje cualquiera, es computacionalmente imposible encontrar otro mensaje cuyo hash sea igual.
- **Colisión fuerte**: Se conoce como resistencia fuerte a las colisiones el hecho de que sea computacionalmente difícil encontrar dos mensajes cuyo hash sea idéntico.

Estas características hacen de los criptosistemas hash el medio perfecto para autentificación de todo tipo de información, con usos que van desde la autentificación de ficheros descargados a través de Internet, hasta *checksum* de paquetes TCP/IP. Es tan sencillo como conocer el hash de la información y una vez obtenida realizar de nuevo la función hash para comparar las cadenas de salida.

A continuación vemos algunos de los principales algoritmos criptográficos de tipo hash:

MD5:

Message Digest 5 (1992)

RFC's relacionados

#1321 - "The MD5 Message-Digest Algorithm" - ftp://ftp.rfc-editor.org/in-notes/rfc1321.txt
#1810 - "Report on MD5 Performance" - ftp://ftp.rfc-editor.org/in-notes/rfc1810.txt
#1828 - "IP Authentication using Keyed MD5" - ftp://ftp.rfc-editor.org/in-notes/rfc1828.txt
#1864 - "The Content-MD5 Header Field" - ftp://ftp.rfc-editor.org/in-notes/rfc1864.txt

Descripción

MD5 fue ideado por el matemático Ron Rivest, y supone la evolución de los algoritmos MD2 y MD4. Se trata de una función criptográfica de tipo hash que acepta como **entrada un mensaje de cualquier longitud** y devuelve como **salida una cadena de 128 bits** (usualmente una cadena de 32 caracteres hexadecimales).

Su fácil implementación y su gran popularidad le hacen uno de los principales algoritmos hash de la red, usado principalmente en comprobación de ficheros en Internet.

SHA-1:

Secure Hash Algorithm - 1 (1994)

RFC's relacionados

#2841 - "IP Authentication using Keyed SHA1 with Interleaved Padding (IP-MAC)" - ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc2841.txt.pdf
#3174 - "US Secure Hash Algorithm 1 (SHA1)" - ftp://ftp.rfc-editor.org/in-notes/pdfrfc/rfc3174.txt.pdf

Descripción

SHA-1 fue ideado por el NIST en 1994 como ampliación al algoritmo SHA. Se trata de una función criptográfica de tipo hash que acepta una **entrada de 2^64 bits como máximo** (2048 Terabytes) y devuelve como **salida una cadena de 160 bits**.

SHA-1 es ligeramente más lento que MD5, pero también es computacionalmente más complejo y su salida es de mayor longitud, por lo que se considera de forma global más seguro.

RIPEMD-160:

RACE Integrity Primitives Evaluation Message Digest - 160 (1996)

RFC's relacionados

#2286 - "Test Cases for HMAC-RIPEMD160 and HMAC-RIPEMD128" - ftp://ftp.rfc-editor.org/in-notes/rfc2286.txt
#2857 - "The Use of HMAC-RIPEMD-160-96 within ESP and AH" - ftp://ftp.rfc-editor.org/in-notes/rfc2857.txt

Descripción

RIPEMD-160 fue ideado por **Hans Dobbertin**, **Antoon Bosselaers**, y **Bart Preneel** como ampliación del algoritmo RIPEMD. Se trata de una función criptográfica de tipo hash que acepta una **entrada un mensaje de cualquier longitud** y devuelve como **salida una cadena de 160 bits**.

A pesar de haberse desarrollado mucho más libre que SHA-1, no es muy popular y tampoco ha sido muy estudiado por criptólogos. No obstante existen dos extensiones de este algoritmo (que son menos usadas aún) denominadas RIPEMD-256 y RIPEMD-320. Las longitudes de sus salidas son respectivamente 256 y 320 bits, con lo que se reduce significativamente las colisiones débiles y fuertes.

Algoritmos asimétricos

Se trata de criptosistemas más modernos y complejos que los simétricos, a la vez que mucho más seguros. Se fundamentan en la existencia de un **par de claves complementarias** que denominamos **clave pública y clave privada** respectivamente (aunque ambas pueden actuar como pública o privada, su función la da el usuario mediante su utilización). Un criptograma generado por una de las claves puede ser descifrado únicamente por la otra clave, y viceversa.

A pesar de ser computacionalmente mucho más complejos, son el estándar hoy en día, usados en sistemas que combinan cifrado asimétrico y simétrico (**Pretty Good Privacy** y **Secure Socket Layer** entre ellos).

Estos sistemas duales aunan las ventajas de ambos sistemas, pues un cifrado continuado en clave asimétrica requiere mucho esfuerzo computacional, y un sistema de clave simétrica no es seguro *per se*, pues necesita un canal seguro de traspaso de información para la clave. El cifrado asimétrico proporciona ese canal seguro de traspaso de información, y el uso de claves únicas en cifrado simétrico garantiza la seguridad a la vez que se reduce sensiblemente el nivel de recursos necesarios.

Existen principalmente tres familias de algoritmos asimétricos, según el principio matemático en el que basan su potencia y seguridad:

- **Problema de la factorización entera:**
  Estos algoritmos basan su seguridad en una debilidad de las máquinas de cómputo actuales. Para un computador es relativamente trivial el cálculo de enormes productos o potencias, pero el proceso contrario resulta muy costoso.
  Así pues, la potencia de estos algoritmos reside en que no existe un método eficiente para factorizar números enteros muy grandes.
  A esta familia pertenecen **RSA** (Rivest-Shamir-Adleman) y **RW** (Rabin-Williams) entre otros.
- **Problema del logaritmo discreto del grupo multiplicativo de un campo finito:**
  La potencia de los algoritmos basados en el problema del logaritmo discreto se basan en que no existe un método eficiente de calcular ***x*** a partir de la expresión $y = a^x \pmod p$ donde ***p*** es un número primo.
  Matemáticamente se trata de un método bastante más complejo, pero computacionalmente es igual de complicado que el problema de la factorización entera.
  A esta familia pertenecen **DH** (Diffie-Hellman), **DSA** (Digital Signature Algorithm), **ElGamal** y **Nyberg-Rueppel** entre otros.
- **Problema del logaritmo discreto sobre el grupo de puntos racionales de una curva elíptica sobre un campo finito:**
  Este grupo es un derivado del problema del logaritmo discreto, solo que en lugar de estudiarlo en un grupo multiplicativo, lo estudia en curvas elípticas (no obstante este grupo es conocido también como "método de curvas elípticas"). Este método es relativamente moderno (surgió en 1986 de la mano de Miller).
  Estos algoritmos basan su potencia en que el cálculo de logaritmos sobre un sistema de curvas elípticas es computacionalmente aún más costoso que su cálculo sobre cuerpos finitos.
  No hay muchos algoritmos de curvas elípticas definidos, pero no obstante existen **versiones elípticas** de los algoritmos **DH** (Diffie-Hellman) y **ElGamal**.

Existe un cuarto grupo de criptografía asimétrica, los **criptosistemas probabilísticos**, que fundamentan su potencia en el hecho de que un mismo texto en claro puede dar lugar a un gran número de criptogramas distintos. Estos sistemas evitan la fuga de información que supone el cifrado asimétrico tradicional (PKCS) ($C = E_k(M)$), pero no han sido muy estudiados aún, y mucho menos trasladados a criptosistemas informáticos.

El concepto de **firma digital**, derivado de la combinación de criptografía asimétrica y algoritmos hash es también de capital importancia, pero lo trataremos en el apartado de PGP.

A continuación vemos algunos de los principales algoritmos criptográficos asimétricos:

RSA:

Rivest - Shamir - Adleman (1978)

RFC's relacionados

#2792 - "DSA and RSA Key and Signature Encoding for the KeyNote Trust Management System" - ftp://ftp.rfc-editor.org/in-notes/rfc2792.txt
#3110 - "RSA/SHA-1 SIGs and RSA KEYs in the Domain Name System (DNS)" - ftp://ftp.rfc-editor.org/in-notes/rfc3110.txt
#3447 - "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1" - ftp://ftp.rfc-editor.org/in-notes/rfc3447.txt

Documentos relacionados

- **[COCK73] Clifford Cocks, *"A Note on 'Non-Secret Encryption'"*,** CESG Research Report, 20 Noviembre 1973, <http://www.cesg.gov.uk/publications/media/nsecret/notense.pdf>.
- **[RIVE78] R. Rivest, A. Shamir and L. Adleman, *"A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM"*,** 21 (2), pp. 120-126, Febrero 1978.
- **[RSA78] R.L. Rivest, A. Shamir, and L.M. Adleman, *"Communications of the ACM (2) 21"*,** (1978), 120-126.
- **[KALI93] Burton Kalinski, *"Some Examples of the PKCS Standards"*,** RSA Laboratories, Noviembre 1993, <ftp://ftp.rsasecurity.com/pub/pkcs/ascii/examples.asc>.
- **[PKCS1] RSA Laboratories, *"PKCS #1 v2.1: RSA Encryption Standard"*,** Junio 2002, <http://www.rsasecurity.com/rsalabs/pkcs/pkcs-1/index.html>.

Descripción

El algoritmo RSA nació en 1978 de la mano de **Ron Rivest**, **Adi Shamir** y **Leonard Adleman [RSA78]**. Se trata de un algoritmo de cifrado asimétrico basado en el problema de la factorización entera, y aunque la descripción de este algoritmo fue propuesta en 1973 por **Clifford Cocks [COCK73]**, fue secreta hasta 1978 cuando se publicó RSA. Aunque el algoritmo fue patentado, la patente expiró en el año 2000 y actualmente se trata de un algoritmo libre.

Echemos un vistazo al algoritmo RSA:

1	Escoger dos números primos muy grandes ***p*** y ***q*** (secretos) y calcular el número ***n*** (público) correspondiente a su producto, ***n = p * q***
2	Escoger la clave de descifrado constituida por un gran número entero ***d*** (secreto), que es primo con el número ***Φ (n)*** (secreto) obtenido mediante: ***Φ (n) = (p-1) * (q-1)***
3	Calcular el entero ***e*** (público) tal que ***1 ≤ e ≤ Φ (n)*** , mediante la fórmula: ***e * d = 1 (mod Φ (n))***
4	Hacer pública la clave de cifado ***(e, n)***
5	Para cifrar texto, es necesario previamente codificar el texto en un sistema numérico en base ***b*** dividiéndolo en bloques de tamaño ***j-1*** de forma que $b^{j-1} < n < b^{j}$
6	Cifrar cada bloque M_i transformándolo en un nuevo bloque de tamaño ***j*** C_i de acuerdo con la expresión $C_i \equiv M_i^{e} \pmod n$
7	Para descifrar el bloque C_i , se usa la clave privada d según la expresión: $M_i \equiv C_i^{d} \pmod n$

**** FUENTE: Pino Caballero Gil, "Introducción a la Criptografía" 2ª Edición actualizada, Ra-Ma. (2002), 55. ISBN 84-7897-520-9***

Veamos un ejemplo eminentemente práctico de cómo generar un criptosistema RSA:

1. Seleccionamos dos números primos ***p = 11*** y ***q = 3***.
2. Calculamos ***N = p * q*** y ***Φ = (p-1) * (q-1) = 10 * 2 = 20***
3. Elegimos el exponente ***e = 3*** comprobando ***mcd (e, p-1) = mcd (3, 10) = 1*** y...
   mcd (e, q-1) = mcd (3, 2) = 1 , lo que implica que...
   mcd (e, Φ) = mcd (e, (p-1) (q-1)) = mcd (3, 10, 2) = 1
4. Calcular ***d*** tal que ***e * d = 1 (mod Φ(n))*** , por ejemplo ***d = 7*** (comprobamos ***e * d-1 = 3 * 7-1 = 20*** que es divisible por ***Φ***)
5. ***Clave Pública = (n, e) = (33, 3)***
   Clave Privada = (n, d) = (33, 7)

Ahora podemos ver un ejemplo de cómo generar un mensaje cifrado con RSA:

1. Queremos encriptar el mensaje ***m = 7***
2. Calculamos el cifrado $c = m^e \bmod n = 7^3 \bmod 33 = 343 \bmod 33 = 13$
3. Nuestro texto cifrado es ***c = 13***
4. Calculamos el descifrado $m' = c^d \bmod n = 13^7 \bmod 33 = 7$
5. El mensaje descifrado es ***m' = 7***

Hoy en día RSA es el algoritmo asimétrico de cifrado más usado, tanto en conexiones de Internet y protocolos seguros, como en cifrado de datos (por ejemplo en el sistema PGP). Las longitudes de clave usadas hoy en día varían desde los 512 hasta los 4096 bits, aunque se suelen tomar de forma habitual claves de 1024 puesto que las de 512 no se consideran suficientemente seguras. Este tamaño puede parecer pequeño, pero permite la generación de claves de longitudes de hasta 1233 cifras con 4096 bits (concretamente hasta 1,044388881413152506691752710716 6 * 10^1233).

No obstante, RSA no es infalible, y como ya dijimos la enorme complejidad computacional del problema de la factorización entera se debe a una limitación de los computadores actuales. Sin embargo, en el momento en el que se puedan construir computadores cuánticos (que trabajen con lógica ternaria en lugar de lógica binaria) suficientemente potentes, mediante la debida implementación del **algoritmo de Shor** (ideado por **Peter Shor** en 1994) este trabajo sería trivial y permitiría resolver criptosistemas basados en el problema de factorización entera en un tiempo polinomial. De hecho en el año 2001, **IBM demostró el algoritmo de Shor** mediante su implementación en un computador cuántico de 7 qubits, factorizando 15 en 3 y 5.

Podemos encontrar más información sobre el algoritmo de Shor, así como el propio algoritmo en: http://en.wikipedia.org/wiki/Shor's_algorithm.

DH/DSS:

Diffie - Hellman / Digital Standard Signature (1976 / 1991)

RFC's relacionados

#2539 - "Storage of Diffie-Hellman Keys in the Domain Name System (DNS)" - ftp://ftp.rfc-editor.org/in-notes/rfc2539.txt
#2631 - "Diffie-Hellman Key Agreement Method" - ftp://ftp.rfc-editor.org/in-notes/rfc2631.txt
#2785 - "Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME" - ftp://ftp.rfc-editor.org/in-notes/rfc2785.txt
#2875 - "Diffie-Hellman Proof-of-Possession Algorithms" - ftp://ftp.rfc-editor.org/in-notes/rfc2875.txt
#3526 - "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)" - ftp://ftp.rfc-editor.org/in-notes/rfc3526.txt

Documentos relacionados

- **[DIH76] W. Diffie and M.E. Hellman, *"IEEE Transactions on Information Theory 22"*,** (1976), 644-654.
- **[ELG85] T. ElGamal, *"IEEE Transactions on Information Theory 31"*,** (1985), 469-472.
- **[DVW92] W. Diffie, P.C. van Oorschot, and M.J. Wiener, *"Designs, Codes and Cryptography 2"*,** (1992), 107-125.
- **[PKCS3] RSA Laboratories, *"PKCS #3 1.4: Diffie-Hellman Key-Agreement Standard"*,** Noviembre 1993, <ftp://ftp.rsasecurity.com/pub/pkcs/ascii/pkcs-3.asc>.
- **[MAU94] U. Maurer, *"Advances in Cryptology"*,** Crypto '94, Springer-Verlag (1994), 271-281.

Descripción

La historia del nacimiento del algoritmo **DH/DSS** es bastante compleja. En 1976, **Dres. W. Diffie y M.E. Hellman** publicaron el documento **[DIH76]** en el que nació el sistema Diffie-Hellman como algoritmo de intercambio de claves. En aquella época, Diffie y Hellman creían que su algoritmo no podía constituir un criptosistema asimétrico completo para cifrado de clave pública. Tuvieron que pasar muchos años hasta que en 1985 **ElGamal** publicara **[ELG85]** y demostrara que se podía desarrollar un criptosistema asimétrico completo a partir del sistema Diffie-Hellman, lo que supuso el nacimiento del algoritmo **DH**.

Pero aún había un problema, pues el sistema de firma ideado por ElGamal ocasionaba que la firma de un mensaje ocupara el doble que el mensaje original, lo cual representaba un problema grave de eficiencia. Este problema fue corregido en 1991, cuando el NIST propuso un sistema de firma nuevo para el sistema Diffie-Hellman, que denominaron **DSS (Digital Standard Signature)**. Así, en 1991 nació el algoritmo completo que hoy conocemos como **DH/DSS**.

DH/DSS es un algoritmo de cifrado asimétrico basado en el problema del logaritmo discreto del grupo multiplicativo de un campo finito. Aunque matemáticamente es mucho más complejo con diferencia que el problema de la factorización entera, se ha demostrado que ambos son computacionalmente de una complejidad muy similar. Hay que decir, eso sí, que DH/DSS y los demás algoritmos basados en el problema del logaritmo discreto no tiene sobre su cabeza la espada de Damocles que supone el algoritmo de Shor para los criptosistemas basados en el problema de la factorización entera.

Ahora echemos un vistazo al algoritmo DH/DSS:

1	Serán de información pública un número primo grande ***p*** y ***a***, una raíz primitiva de ***1 módulo p*** (es decir, tal que $a^{p-1} \equiv (mod\ p)$ y $a^{d-1} \neq (mod\ p)$ *para todo d tal que* $1 < d < p$)
2	La clave privada del usuario B es un entero k_B escogido dentro del intervalo ***[1 , p-1]***
3	La clave pública de B es el entero $K_B \equiv a^{k_B} (mod\ p)$
4	Se supone que el emisor A quiere enviar un mensaje ***M*** $(1 \leq M \leq p-1)$ al receptor B
Proceso de Cifrado (e)	
1e	Escoger aleatoriamente un entero k_A tal que $1 \leq k_A \leq p-1$, que constitye su clave secreta
2e	Calcular la clave de cifrado a partir de su propia clave privada y la clave pública de B, $Q \equiv K_B^{k_A} (mod\ p)$
3e	Cifrar el mensaje M según la expresión $C \equiv Q * M (mod\ p)$
Proceso de Descifrado (d)	
1d	Obtener Q gracias a la clave pública de A, K_A , y a su propia clave secreta mediante la fórmula $Q \equiv K_A^{k_B} (mod\ p)$
2d	Recuperar M a partir de $M \equiv (Q^{-1})_p * C (mod\ p)$ donde $(Q^{-1})_p$ denota el inverso de Q en módulo p

**** FUENTE: Pino Caballero Gil, "Introducción a la Criptografía" 2ª Edición actualizada, Ra-Ma. (2002), 65. ISBN 84-7897-520-9***

Es importante señalar que aunque el sistema Diffie-Hellman siempre ha sido libre, la patente que lo cubría (US 4.200.770) expiró el 6 de Septiembre de 1997.

RW:

Rabin - Williams (1980)

Documentos relacionados

- **[RAB79] M.O. Rabin, *"Digitalized signatures and public-key functions as intractable as factorization"*,** Technical Report MIT/LCS/TR-212, MIT, 1979.
- **[WEL88] D. Welsh, *"Codes and Cryptography"***, Claredon Press, 1988.

Descripción

En 1979 **M.O. Rabin** ideó un sistema a partir de las bases sentadas por RSA. Sin embargo, hay una sutil diferencia, pues en el sistema RSA, resolver el problema general de la factorización implica romperlo, pero no de forma inversa. En el sistema de Rabin, ambos preceptos se cumplen. Este defecto de base fue corregido en 1980 por **H.C. Williams**, dando lugar al algoritmo **RW**.

RW es un algoritmo de cifrado asimétrico basado en el problema de la factorización entera (al igual que RSA). No obstante, este algoritmo no ha sido prácticamente estudiado ni usado por la comunidad criptográfica moderna.

Echemos un vistazo al algoritmo RW:

1	Escoger dos grandes primos ***p*** y ***q*** (secretos) y definir ***n*** (público) mediante su producto, ***n = p * q*** (público)
2	Escoger un entero ***B < n*** que será parte de la clave pública ***(B,n)***
3	Para cifrar un texto, es necesario previamente codificar el texto en un sistema de base ***b*** dividiéndola previamente en bloques de tamaño ***j-1*** de forma que cumpla $b^{j-1} < n < b^{j}$
4	Cifrar el bloque M_i según la expresión $C_i \equiv M_i (M_i + B) \pmod n$
5	Para descifrar el criptograma C_i , hay que construir el mensaje $M_i = a * u + b * v$ *a* partir de las soluciones de las ecuaciones $u^2 + B * u \equiv C_i \pmod p$ y $v^2 + B * v \equiv C_i \pmod q$, y los enteros ***a*** y ***b*** tales que $a \equiv 1 \pmod p$, $a \equiv 0 \pmod q$, $b \equiv 0 \pmod p$ y $b \equiv 1 \pmod q$

**** FUENTE: Pino Caballero Gil, "Introducción a la Criptografía" 2ª Edición actualizada, Ra-Ma. (2002), 62. ISBN 84-7897-520-9***

Por descontado, al estar basado este algoritmo en el mismo problema que RSA, posee igualmente sus mismas debilidades, y es también extremadamente vulnerable al criptoanálisis cuántico.

Otros sistemas asimétricos:

El último gran grupo de criptografía asimétrica es el método de las curvas elípticas. Éste surgió en 1986 de la mano de **Miller** con la publicación del documento:

- **[MIL86] V.S. Miller,** ***"Use of Elliptic Curves in Cryptography",*** *Advances in Cryptology-CRYPTO'85, Lecture Notes in Computer Science,* Springer-Verlag, 1986.

El sistema de curvas elípticas se inspira en el sistema de ElGamal, con el grupo aditivo de puntos de la curva representando el papel del grupo multiplicativo.

Se trata de un sistema mucho más complejo, tanto matemática como computacionalmente, que los métodos tradicionales (factorización y logaritmo único), pero proporciona un nivel de seguridad mucho mayor. Así, se ha evaluado que una clave de un sistema de curvas elípticas de 210 bits es igual de segura que una clave RSA de 2048 bits.

Aunque existen versiones elípticas de sistemas como DH ó ElGamal, y a pesar de ser uno de los campos más estudiados por la criptografía moderna, no han sido implementados de una manera efectiva aún, y no gozan de mucha importancia. No obstante, se trata de un campo de estudio relativamente moderno y por desarrollar.

Existen otros dos sistemas importantes no mencionados anteriormente:

- **Problema de la mochila:**
  Se trata de un sistema ideado en 1978 por **Merkle** y **Hellman** en el algoritmo **Merkle-Hellman [MEH78]** y que toma su nombre de una metáfora del algoritmo, puesto que se presenta como una mochila de tamaño ***N*** donde queremos meter ***n*** objetos pero rellenando el espacio sobrante hasta completar ***N***.
  Aunque el problema general de la mochila es muy complejo y constituye un problema NP-completo, todas las versiones de este sistema han sido rotas y no es usado salvo con fines didácticos
- **Teoría de la codificación algebraica:**
  Este sistema fue ideado por **McEliece** en 1978 y dio lugar al algoritmo **McEliece [MCE78]**. Basa su potencia en el hecho de que la decodificación de un código lineal general es un problema NP-completo. En este algoritmo juega un papel crucial la teoría de la codificación.
  Un punto importante a la hora de entender la complejidad computacional de este algoritmo es el hecho de que no trabaje con cifras enteras, sino con matrices. Además, se introduce un gran factor de expansión de datos, en función de las palabras del código de Goppa, y se produce un desorden intencionado mediante la adición de ruido.
  Por desgracia su aplicación de momento no pasa del ámbito teórico, y no se han desarrollado criptosistemas de clave pública sobre este algoritmo, aunque está siendo investigado hoy en día.

El último grupo del que hablaremos brevemente es el de los **criptosistemas probabilísticos**. Este sistema se basa en las nociones que **Micali** y **Goldwaser** publicaron en el documento **[GOM82]**:

- **[GOM82] S. Goldwasser, S. Micali,** ***"Probabilistic Encryption and How to Play Mental Poker Keeping Secret all Partial Information",*** *Proceedings of the 14th ACM Symposium on Theory of Computing,* 365, 1982.

Basa su potencia en que no existe un cálculo factible para obtener información del texto en claro a partir del criptograma. Esto se logra mediante la virtual existencia de un gran número de criptogramas distintos para un mismo texto en claro de partida.

Se puede considerar, de hecho, un "Sistema de Secreto Perfecto" de Shannon, pero con la ventaja añadida del uso personalizado de claves criptográficas. Por desgracia aún no se han podido desarrollar criptosistemas informáticos completos a partir de este sistema.

El sistema PGP

Corría el año 1991 y en Estados Unidos comenzaba a tomar forma el rumor de que el Gobierno pretendía prohibir el uso de la criptografía en las líneas de comunicación. La criptografía había alcanzado un nivel de desarrollo que había escapado al control de gobiernos y servicios de inteligencia, y por ello se comenzó a pensar en controlarla. Mucha gente comenzó a preocuparse por el incipiente rumor, y un programador llamado **Philip Zimmermann** decidió poner remedio, había que hacer que no fuera posible prohibir la criptografía.

Zimmermann escogió los mejores algoritmos del momento de cifado simétrico (**IDEA**), asimétrico (**RSA**) y hash (**MD5**) y creó el programa llamado **Pretty Good Privacy (PGP)**, un software de cifrado al nivel más alto y totalmente libre de puertas traseras o control alguno. Comenzó a distribuirlo por BBSs de forma gratuita (freeware) y la gente respondió distribuyéndolo por todo el país. De repente no era posible prohibir la criptografía porque estaba demasiado extendida y se había convertido en el estándar de cifrado y firma de correo electrónico.

Los problemas no tardaron en llegar. En ese mismo año (1991) alguien distribuyó el programa en **USENET** y éste comenzó a distribuirse y usarse fuera de Estados Unidos. No tardaron en acusar a Zimmermann de violar la **legislación ITAR** del Departamento de Estado Americano sobre exportación de armas. Sí, en Estados Unidos la criptografía se considera un arma equiparable al armamento pesado, los carros de combate o las armas químicas. Además, **RSA Data** le acusó de violar el copyright del algoritmo RSA (que por aquel entonces aún estaba patentado). Así, se formaron dos frentes de problemas legales contra Philip.

Para intentar paliar el problema con RSA Data, se llegó a la determinación de desarrollar dos versiones paralelas de PGP: el propio PGP en Estados Unidos y Canadá, usando la librería **RSAREF**; y la versión PGPi (de International) que se desarrollo para el resto del mundo en Europa, usando la librería **MPILIB** desarrollada por el propio Zimmermann. Aún así, el proceso con RSA Data continuó.

Ambos problemas se terminaron solucionando en 1996. Respecto a la supuesta violación de la legislación ITAR, el 11 de Enero de 1996 el abogado de Zimmermann recibió una escueta carta con el siguiente texto:

> *"La Oficina del Fiscal del Distrito Norte de California ha decidido que su cliente, Philip Zimmermann, no será juzgado por el envío a USENET en junio de 1991 del programa de encriptación Pretty Good Privacy. El caso queda cerrado".*

También fue publicada un no menos escueto comunicado a la prensa:

> *"Michael J. Yamaguchi, Fiscal del Distrito Norte de California, ha anunciado que su Oficina ha decidido abandonar la causa contra cualquier individuo supuestamente implicado en el envío a USENET en junio de 1991 del programa de encriptación Pretty Good Privacy. El caso queda cerrado, y la Oficina no hará más declaraciones".*

Y de ese problema no volvió a saberse nada, ni se hicieron más declaraciones.

El proceso que RSA Data inició contra Zimmermann también terminó en 1996, siendo declarado inocente y absuelto. Ese mismo año creó su propia compañía, **Pretty Good Privacy Inc.**, y comenzó a desarrollar software (por supuesto entre ese software se encontraba PGP) dentro de Estados Unidos. Las versiones internacionales de PGP siguieron desarrollándose en Europa.

El año siguiente, 1997, supuso un cambio radical para PGP. Pretty Good Privacy Inc. fue absorbida por Network Associates, y Zimmermann quedó relegado a un puesto de asesor. Así mismo, se publicó la primera versión para Windows, la 5, que cambiaba la línea de comandos DOS por un entorno gráfico. A partir de esta versión comenzaron a incorporarse nuevos algoritmos.

Aunque durante un tiempo PGP perdió mucho de su espíritu original, hoy en día por fortuna vuelve a ser lo que era. Aún así mucha gente sigue prefiriendo la última versión de DOS que se publicó antes del cambio de manos de PGP, la 2.6.3 bajo línea de comandos, con soporte único para **RSA-IDEA-MD5**.

La versión actual de PGP es la 8.03. Existen dos versiones de PGP: una de pago, y otra freeware (con ciertas limitaciones) que puede descargarse desde http://www.pgp.com/products/freeware.html. Aunque PGP **NO** es software libre, su código fuente vuelve a estar disponible para ser descargado y consultado por el público general en http://www.pgp.com/products/sourcecode.html.

Hoy en día PGP trabaja con los siguientes algoritmos:

- **Encriptación simétrica:** AES (Rijndael), CAST, TripleDES, IDEA, Twofish.
- **Encriptación asimétrica:** RSA, RSA Legacy, DH/DSS.
- **Hash:** MD5, SHA-1, RipeMD (sólo para claves RSA).

Es importante hablar también de **openPGP**. A partir de la idea de PGP, se desarrolló en Noviembre de 1998 el estándar openPGP, gracias al cual cualquiera puede integrar en una aplicación compatibilidad con el sistema PGP a través de una implementación de openPGP (ciertos clientes de correo electrónico incluyen esta funcionalidad). Es conveniente leer los siguientes RFC's referidos a openPGP:

#2440 - "OpenPGP Message Format" - ftp://ftp.rfc-editor.org/in-notes/rfc2440.txt
#3156 - "MIME Security with OpenPGP" - ftp://ftp.rfc-editor.org/in-notes/rfc3156.txt

Por último (pero no menos importante) hay que hablar de la implementación openPGP más famosa y usada: **GNU Privacy Guard (GnuPG, GPG)**. GnuPG es un reemplazo libre y completo del sistema PGP, cumpliendo los estándares de openPGP. Trabaja con los mismos algoritmos que las versiones actuales de PGP **excepto IDEA**, pues se encuentra patentado aún y no es libre en el territorio de Estados Unidos y Canadá. No obstante se puede añadir el algoritmo IDEA a GnuPG mediante un parche, pero sólo debemos hacerlo en caso de residir en países donde IDEA esté libre de patentes (España por ejemplo).

GPG es completamente software libre, bajo los términos de la licencia **GNU General Public License (GPL)** y hoy día está muy extendido, contando incluso con apoyo económico del Ministerio de Economía y Tecnología de la República Federal de Alemania para su desarrollo.

La primera versión de GnuPG (1.0.0) fue liberada el 7 de Septiembre de 1999, y la última versión disponible a día de hoy es la 1.2.4 que podemos descargar desde su sitio web http://www.gnupg.org/. GnuPG tiene actualmente soporte para gran cantidad de sistemas, entre ellos: **GNU/Linux**, **GNU/Hurd**, **FreeBSD**, **OpenBSD**, **NetBSD**, **Microsoft Windows**, **PocketConsole**, **MacOS X**, **AIX**, **BSDI**, **HPUX**, **IRIX**, **MP-RAS**, **OSF1**, **OS/2**, **SCO UnixWare**, **SunOS**, **Solaris**, **USL UnixWare**.

GnuPG es una aplicación de línea de comandos, pero existen muchas Interfaces Gráficas de Usuario (GUI, Graphic User Interface) para integrarlo en un escritorio Unix e incluso Windows. Las más famosas son **GPA**, **GnomePGP**, **KGpg** y **TkPGP**. No obstante, la más recomendada (yo la uso personalmente) es **GPA (GNU Privacy Assistant)**, que además es mantenido por el propio equipo de GnuPG. GPA es una aplicación gráfica que utiliza **GTK (El Gimp Tool Kit)**. La última versión de GPA es la 0.70, que puede ser encontrada en http://www.gnupg.org/(es)/download/index.html#gpa.

Ya conocemos los algoritmos criptográficos, conocemos la historia de PGP y sus implementaciones. Es el momento de empezar a comprender el funcionamiento de PGP. Vamos a ver algunas de las principales funciones de los sistemas openPGP, explicando la teoría de su funcionamiento, y cómo realizarlas en los sistemas openPGP más usados: PGP (versión 8.03), GnuPG (versión 1.2.4) y GPA (versión 0.70).

Es importante recomendar también la lectura de los manuales de PGP (*Help > Contents and Index*) ó GnuPG (*man gpg*) detalladamente para poder aprovechar todo el potencial que estos programas nos brindan.

El anillo de claves

¿Qué?

Ya sabemos lo que es una clave criptográfica. Antes de ver cómo generar nuestra propia clave, es importante saber cómo se manejan y almacenan en nuestro PC. En el sistema openPGP existe un concepto muy importante, el concepto de **armadura**. Una armadura es un elemento openPGP (clave, criptograma, firma, anillo de claves...) representado en caracteres ASCII y con unas cadenas de inicio y final predefinidas. Veamos algunos ejemplos:

-----BEGIN PGP SIGNED MESSAGE-----
Hash: *{algoritmo de hash}*

{mensaje firmado}

-----BEGIN PGP SIGNATURE-----
Version: *{versión}*

{firma}
-----END PGP SIGNATURE-----

****

-----BEGIN PGP MESSAGE-----
Version: *{versión}*

{mensaje cifrado}
-----END PGP MESSAGE-----

****

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: *{versión}*

{clave pública}
-----END PGP PUBLIC KEY BLOCK-----

Estas estructuras son lo que denominamos **armaduras ASCII** o **ASCII armor**. Mediante ese sistema, y en texto plano (que normalmente se representa en ficheros *.asc) podemos transportar fácilmente cualquier elemento openPGP. Hay que saber también que existe otro formato denominado **MIME/PGP**.

A partir de estas armaduras (que pueden combinarse para obtener, por ejemplo, mensajes encriptados y firmados) se constituye el **anillo de claves** o **keyring**. Existen dos anillos, el de claves públicas (**public keyring**) que contiene, además de nuestras propias claves públicas, todas las claves públicas de otras personas que importemos para usar; y el de claves privadas (**secret keyring**) que contendrá nuestras claves privadas. En el anillo podemos ver la clave, el nombre, el correo, el ID de clave (KeyID)...

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*

GPG

1. Anillo de claves públicas: *gpg --list-keys*
2. Anillo de claves privadas: *gpg --list-secret-keys*

GPA

1. Abrir el programa *GPA (Editor del anillo)*

Generar un par de claves

¿Qué?

Es el momento de generar un **par de claves pública/privada (public/private key pair)**. Es importante entender que se trata de un par de claves complementarias a las que se les asigna una utilidad en el criptosistema. Una vez generado nuestro par de claves, pasará a formar parte de nuestro anillo de claves y podremos usarlas, exportarlas y modificarlas.

Hay que mencionar el concepto de **passphrase**. Al igual que un password es una palabra que protege el acceso a un sistema (contraseña), un passphrase es una frase que protege el uso de la clave privada. También la llamamos contraseña en castellano. Se almacena en formato de hash criptográfico dentro de la propia clave.

A la hora de crear un par de claves, debemos elegir si queremos usar el algoritmo RSA o DH/DSS. Esa es una elección personal y ya conocemos los detalles de cada algoritmo. Yo personalmente elegí RSA, pero la mayoría de la gente prefiere DH/DSS (también es cierto que se trata del sistema por defecto y poca gente se molesta en personalizar el proceso de generación).

También debemos decidir si queremos que nuestra clave sea imperecedera o que caduque en *x* tiempo.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. Menú *Keys >> New Key*
3. Elegimos la opción *Expert Mode*
4. Introducimos nombre, correo electrónico, y elegimos algoritmo, tamaño y fecha de expiración
5. Introducimos por duplicado el passphrase a usar

GPG

1. Ejecutamos el comando *gpg --gen-key*
2. Elegimos el tipo de clave (1-DSA y ELGamal (por defecto); 2-DSA (firmar), 4-RSA (firmar))
3. Elegimos el tamaño (1024-4096) y confirmamos
4. Elegimos la caducidad (nunca, n días, n semanas, n meses, n años) y confirmamos
5. Introducimos nombre y apellidos
6. Introducimos la dirección de correo electrónico
7. Introducimos un comentario para la clave (opcional)
8. Podemos cambiar algún dato o confirmar de forma general la clave
9. Introducimos por duplicado el passphrase a usar
10. Esperamos a que termine la generación de entropía para la clave
11. *Si es necesario generar una clave de cifrado, ejecutamos el comando *gpg --edit-key <KeyID>*
12. *Ejecutamos el comando *addkey* y nos autentificamos mediante el passphrase
13. *Elegimos el tipo de clave (2-DSA (firmar); 3-ElGamal (cifrar); 4-RSA (firmar); 5-RSA (cifrar))
14. *Elegimos el tamaño (1024-4096) y confirmamos
15. *Elegimos la caducidad (nunca, n días, n semanas, n meses, n años) y confirmamos

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. Menú *Claves >> Nueva Clave*
3. Elegimos el algoritmo y el tamaño, introducimos el nombre, correo electrónico, comentario (opcional), caducidad y el passphrase de la clave por duplicado

Cambiar el passphrase

¿Qué?

Quizá en algún momento deseemos poder cambiar el passphrase que seleccionamos en la creación de nuestro par de claves. Con cualquier software compatible con openPGP es posible.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Key Properties*
3. Seleccionamos *Change Passphrase...*
4. Nos autentificamos mediante el passphrase e introducimos por duplicado el nuevo passphrase

GPG

1. Ejecutamos el comando *gpg --edit-key <KeyID>*
2. Ejecutamos el comando *passwd* y nos autentificamos mediante el passphrase
3. Introducimos por duplicado el nuevo passphrase

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Editar Clave Privada*
3. Seleccionamos *Cambiar Contraseña*
4. Nos autentificamos mediante el passphrase e introducimos por duplicado el nuevo passphrase

Consultar el fingerprint

¿Qué?

Cuando queremos añadir a nuestro anillo de claves una clave pública ajena para ser usada en comunicaciones seguras, debemos poder cerciorarnos de que esa clave pertenece de hecho a esa persona. Para ello existe el **fingerprint o huella digital**. El fingerprint es el hash de la clave pública. Así, cuando queremos obtener la clave pública de una persona, debemos obtener de esa persona por un medio seguro (preferentemente en persona) el fingerprint hexadecimal de su clave pública. Luego bajamos su clave de un servidor de claves (más adelante hablaremos de ello) y comprobamos que el fingerprint coincide (ya conocemos los algoritmos hash y sabemos en qué se basa su seguridad).

Por ejemplo, este sería el fingerprint de mi clave pública (***KeyID 0xAF9593E1***):

6FAB 9799 C61A A7D2 A409 8A53 6F6F 3938 AF95 93E1

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Key Properties*
3. Campo Fingerprint (seleccionamos la opción *hexadecimal* si no está activada)

GPG

1. Ejecutamos el comando *gpg --fingerprint <KeyID>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BID sobre la clave*
3. Campo detalles, *huella dactilar*

Importar una clave

¿Qué?

Tanto si se trata de una copia de seguridad de nuestras claves privadas como de claves públicas que deseamos tener en nuestro anillo de claves, una opción muy importante es la de poder importar claves de armaduras ASCII. Existe otro método de importación de claves, a través de un servidor de claves, pero lo trataremos más adelante. Este método es válido únicamente para importar ficheros *.asc de armadura ASCII que contenga una o más clave/s pública/s y/o privada/s.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. Menú *Keys >> Import...*
3. Seleccionamos la opción *Import*

GPG

1. Ejecutamos el comando *gpg --import <fichero>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. Menú *Claves >> Importar Claves*

Exportar una clave pública

¿Qué?

Para poder distribuir nuestra clave pública y que la gente pueda comunicarse de forma segura con nosotros, debemos poder exportar nuestra clave pública. Existe otro método de exportación de claves, a través de un servidor de claves, pero lo trataremos más adelante. Este método es válido únicamente para generar un fichero de armadura ASCII que contenga una o varias claves públicas.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Export*

GPG

1. Ejecutamos el comando *gpg --armor --export <KeyID> <fichero>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Exportar Claves*

Exportar una clave privada

¿Qué?

Un punto muy importante es la realización de copias de seguridad de nuestras claves privadas, de forma que una eventual pérdida de datos importante en nuestro sistema no nos ocasione la pérdida total de nuestras claves criptográficas. Supongo que está claro que el perder una clave privada supone perder cualquier fichero o mensaje encriptado a ella... Es importante señalar que éste es el único método de exportación de claves privadas. **NO** es posible exportar claves privadas a un servidor de claves. Este método genera un fichero de armadura ASCII que contiene una o varias claves privadas (según en qué implementaciones de openPGP, la exportación de una clave privada conlleva también la pública).

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Export*
3. Marcamos la casilla *Include Private Key(s)*

GPG

1. Ejecutamos el comando gpg *--armor --export-secret-keys <KeyID> <fichero>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Copia de Seguridad*

Exportar una clave pública al Servidor de Claves

¿Qué?

Existen servidores especiales que contienen miles (me atrevería a afirmar que millones) de claves públicas para que cualquier persona pueda buscar y descargar a su anillo de claves. Este método facilita enormemente el intercambio de claves, así como la distribución de firmas mediante la actualización de claves (alguien firma nuestra clave y la sube al servidor, y después nosotros la actualizamos en nuestro anillo).

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Send To >> Domain Server*

GPG

1. Ejecutamos el comando *gpg --keyserver <KeyServ> --send-key <KeyID>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. Menú *Servidor >> Enviar Claves*

Buscar una clave en el Servidor de Claves

¿Qué?

Ya sabemos para qué sirve un servidor de claves y cuál es su funcionamiento. Igual que podemos colgar nuestra clave pública para que cualquier usuario la descargue, podemos buscar la clave pública de cualquier usuario.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. Menú *Server >> Search...*
3. Seleccionamos el servidor de claves y las opciones de búsqueda para realizar la query

GPG

1. Ejecutamos el comando *gpg --keyserver <KeyServ> --search-key <KeyID>*

GPA

1. **NO** soporta la búsqueda de claves

Importar una clave del Servidor de Claves

¿Qué?

Una vez localizada la clave, podemos descargarla desde el servidor de claves a nuestro anillo de claves personal.

¿Cómo?

PGP

1. Una vez realizados los pasos de búsqueda del apartado anterior...
2. *BDR sobre la clave >> Import to Local Keyring*

GPG

1. Ejecutamos el comando *gpg --keyserver <KeyServ> --recv-key <KeyID>*

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. Menú *Servidor >> Descargar Claves*

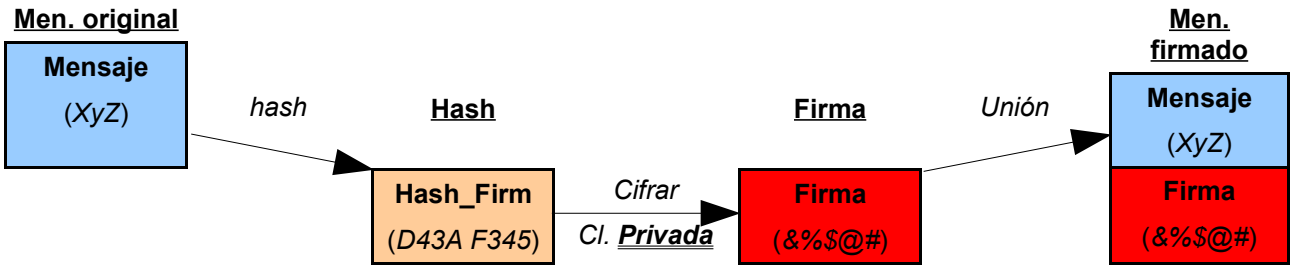
Firmar una clave pública (firma exportable)

¿Qué?

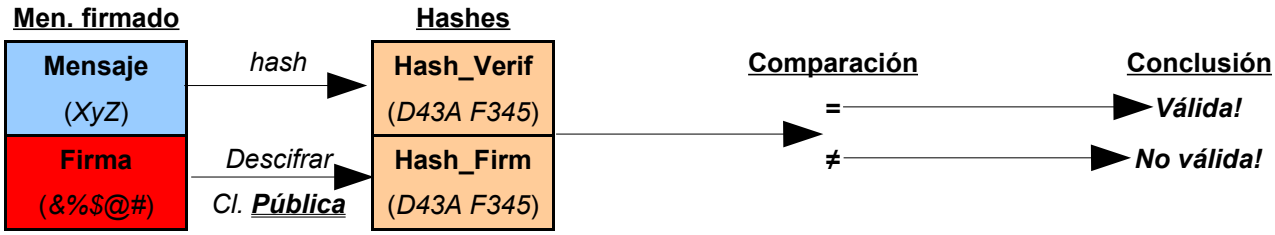
Es el momento de introducir y explicar un concepto muy importante: la firma digital del sistema PGP. Ya hemos comentado y sabemos que en los sistemas openPGP se utiliza un sistema criptográfico asimétrico que se sirve de dos claves complementarias (pública y privada). Mucha gente piensa -erróneamente- que la clave pública sirve para encriptar y la privada para desencriptar. **NO**. Ambas claves son **complementarias**, y como tal ambas pueden ser usadas -y de hecho son usadas- para ambas funciones, de forma que lo que una encripta, la otra lo desencripta. ¿Qué utilidad nos reporta esto? Pues la firma digital.

Más adelante trataremos el problema del cifrado de información, pero efectivamente en ese caso el cifrado se realiza a la clave pública para que el destinatario pueda descifrarlo con su clave privada. En el caso de la firma digital es al revés. Una firma digital (es indiferente si se trata de una clave, un fichero o un correo) es el hash del objeto a firmar encriptado a la clave **privada** del firmante. Así, cualquier persona con la clave pública del firmante (que debería ser accesible desde un servidor de claves) puede descifrar ese criptograma para obtener el hash, realizar por su cuenta un hash del objeto firmado (obviamente usando el mismo algoritmo hash) y compararlos: si ambos hashes son idénticos, la firma es válida, en caso contrario la firma o el objeto han sido manipulados.

Proceso de firma:



Proceso de verificación:



Este mecanismo de firma digital permite que cualquier persona con nuestra clave pública (accesible a cualquiera) y un sistema openPGP capaz de descifrar el criptograma con ella, realice una comparación del hash realizado por el firmante y el suyo propio, de forma que pueda autentificar la firma o no.

Al firmar una clave pública con nuestra propia clave privada estamos otorgando al usuario de esa clave pública nuestra confianza a todos los efectos. Este tipo de firmas son exportables, es decir, al exportar (tanto a un fichero como a un servidor de claves) la clave pública, nuestra firma irá incluida en ella.

Un sistema de firma digital cumple los siguientes preceptos:

1. **Integridad de la información:** La información firmada no puede ser modificada sin alterar la validez de la firma.
2. **Autenticidad del origen del mensaje:** Solamente el propietario de la clave privada (y la capacidad de activarlo mediante el passphrase correcto) es capaz de firmar una información.
3. **No repudio del origen:** El usuario que ha generado la firma no puede repudiar el mensaje.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Sign*
3. Marcamos la casilla *Allow signature to be exported*
4. Nos autentificamos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --sign-key <KeyID>*
2. Seleccionamos la confianza en la clave a firmar (0, 1, 2 ó 3)
3. Confirmamos y nos autentificamos mediante el passphrase

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Firmar Claves*
3. Nos autentificamos mediante el passphrase

Firmar una clave pública (firma NO exportable)

¿Qué?

En esencia se trata del mismo proceso que la firma exportable, con la evidente diferencia de la imposibilidad de exportar estas firmas. Estas firmas son solamente locales, y **NO** se conservarán al exportar la clave pública firmada a un fichero o un servidor de claves.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Sign*
3. Nos autentificamos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --lsign-key <KeyID>*
2. Seleccionamos la confianza en la clave a firmar (0...3)
3. Confirmamos y nos autentificamos mediante el passphrase

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Firmar Claves*
3. Marcamos la casilla *Firmar solo localmente*
4. Nos autentificamos mediante el passphrase

Establecer el grado de confianza en el usuario

¿Qué?

El concepto de confianza funciona localmente, y es un dato que nos sirve para, en procesos de descifrado o comprobación de firmas digitales, comprobar en un vistazo el grado de validez de los datos. Existen varios grados según la implementación openPGP. En PGP tenemos tres grados además de la **confianza implícita** (sólo disponible para pares de claves pública/privada). En GPG/GPA tenemos cinco grados (desconocida, sin confianza, poca confianza, confianza total, confianza por completo).

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Key Properties*
3. Graduar el medidor del apartado *Trust Model*
4. *Marcar la casilla *Implicit Trust*

GPG

1. Ejecutamos el comando *gpg --edit-key <KeyID>*
2. Ejecutamos el comando *trust*
3. Seleccionamos el nivel de confianza (1...5)

GPA

1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Establecer confianza en el usuario*
3. Seleccionamos el nivel de confianza (1...5)

Borrar una clave

¿Qué?

Puede que en algún momento queramos borrar una o varias claves públicas o privadas (porque hayan caducado, porque ya no nos sean útiles...). En tal caso podemos simplemente eliminarlas de nuestro anillo de claves local.

¿Cómo?

PGP

1. Abrir el programa *PGPkeys*
2. *BDR sobre la clave >> Delete*
3. Confirmamos el proceso

GPG

1. Claves públicas: *gpg --delete-key <KeyID>*
2. Claves privadas: *gpg --delete-secret-key <KeyID>*
3. Pares de claves: *gpg --delete-secret-and-public-key <KeyID>*
4. Confirmamos el proceso

GPA

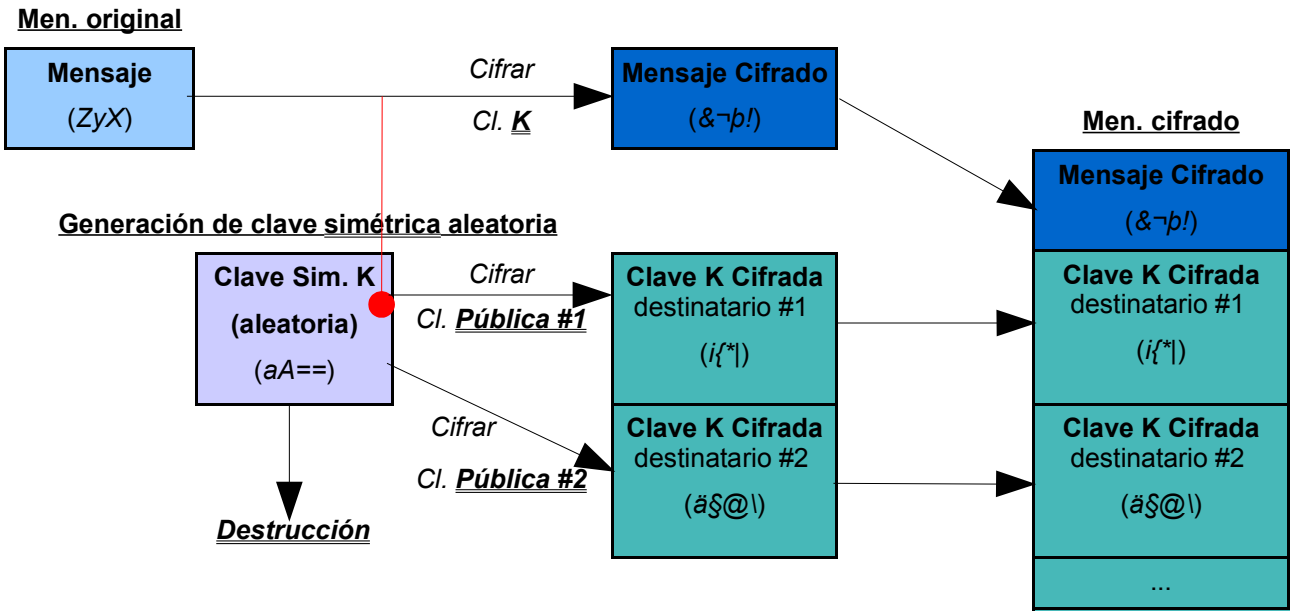
1. Abrir el programa *GPA (Editor del anillo)*
2. *BDR sobre la clave >> Borrar Claves*
3. Confirmamos el proceso

Encriptación con el sistema openPGP

¿Qué?

Llegamos a una de las partes más importantes del sistema PGP. Ya hemos visto durante todos los procesos relacionados con la clave la importancia de la criptografía asimétrica en PGP, al igual que hemos visto la importancia de los algoritmos hash en la parte de firmas digitales. Ahora ha llegado el turno de analizar la importancia de la criptografía simétrica en el sistema PGP.

Otro de los conceptos erróneos más extendidos sobre PGP es que toda el peso del cifrado y descifrado recae sobre los algoritmos asimétricos (RSA o DH/DSS). Esto es **falso**, pues en realidad la encriptación a claves asimétricas de semejante tamaño (1024-4096 bits) es un trabajo computacionalmente pesado, y la encriptación de grandes ficheros consumiría gran cantidad de tiempo. Por otro lado, con lo que sabemos sobre criptografía simétrica, es fácil darse cuenta que el principal método de criptoanálisis es la estadística por comparación, y que cuantos más criptogramas tengamos cifrados a un mismo código, éste será más vulnerable. Así pues, un criptosistema simétrico solamente es seguro cuando además de usar una clave suficientemente segura (256 bits), ésta es usada una única vez. Este principio es aplicado a los protocolos informáticos seguros (SHTTP, SSL, SSH...) y también al sistema PGP. Veamos cómo:



En primer lugar, se genera una **clave simétrica aleatoria (*K*)**. Esta clave ha de ser lo más aleatoria posible, y para ello se genera entropía (capturando datos tan banales como flujo de datos entre dispositivos, red, periféricos...). El algoritmo y tamaño de la clave normalmente puede ser elegido por el usuario (típicamente se usa AES de 256 bits). Usando esta clave *K* se cifra el mensaje original.

En segundo lugar se cifra la clave *K* a la clave pública de cada uno de los destinatarios del mensaje cifrado, de forma que cada uno de ellos pueda usar su propia clave privada para descifrar la clave de cifrado del mensaje y poder así obtener el mensaje original. Esta parte constituida por la clave *K* cifrada a distintas claves públicas se adjunta al mensaje cifrado y constituye el criptograma.

Por último, la clave *K* es destruida de forma **segura**.

El mensaje puede estar, además de encriptado, firmado. Un mensaje cifrado y firmado incluirá en el criptograma además una parte de firma, constituida por el hash del mensaje original (NO del cifrado) encriptado a la clave privada del firmante.

Al tratarse *K* de una clave aleatoria de un solo uso, el sistema se considera seguro, y la existencia de la combinación de tres tipos de criptosistemas combinados permite una distribución segura de claves a la vez que una enorme seguridad a la hora de cifrar y firmar mensajes mediante el método descrito.

¿Cómo?

PGP

1. Abrir el programa *PGPmail*
2. Seleccionar la opción *Encrypt* (segunda empezando por la izquierda)
3. Seleccionar el fichero a encriptar
4. Arrastrar a la ventana de *Recipients* los destinatarios del fichero.
5. *Podemos opcionalmente marcar alguna casilla si deseamos opciones extra (*Text Output*, *Input Is Text*, *Wipe Original* (borra de forma segura el original), *Secure Viewer*, *Conventional Encryption* (encriptación exclusivamente simétrica), *Self Decrypting Archive*)
6. *En caso de firmar además de cifrar, debemos autentificarnos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --encrypt-files <fichero>*
2. Introducimos los *<KeyID>* a los que destinamos el fichero

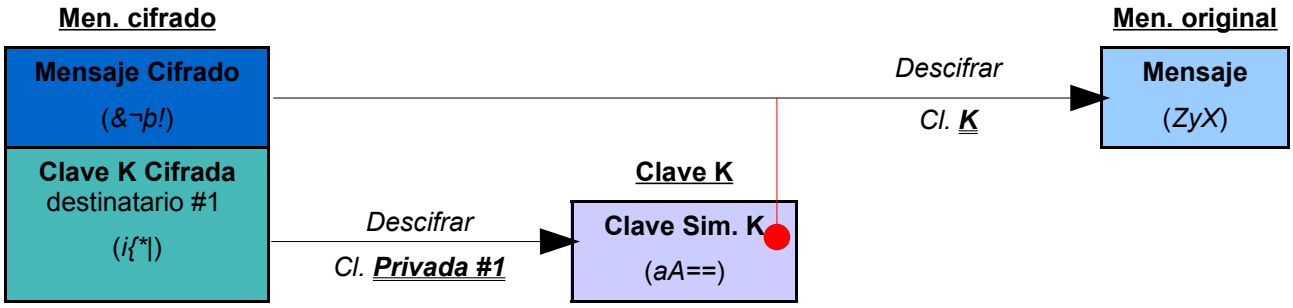
GPA

1. Abrir el programa *GPA (Gestor de Archivos)*
2. Menú *Abrir >> Archivo*
3. Seleccionamos la opción *Cifrar*
4. Seleccionar el fichero a encriptar
5. Seleccionamos los destinatarios del fichero

Desencriptación con el sistema openPGP

¿Qué?

Conocido el proceso de cifrado, el proceso de descifrado es el inverso:



Uno de los destinatarios del mensaje cifrado recibe el criptograma completo y de él extrae el mensaje cifrado y la clave *K* cifrada a su clave pública (si no estuviera, ese mensaje no se habría cifrado para él). Mediante su clave privada -previa autentificación mediante passphrase- descifra la clave simétrica *K*. Con la clave *K* ya puede desencriptar el mensaje cifrado para obtener el mensaje original.

¿Cómo?

PGP

1. Abrir el programa *PGPmail*
2. Seleccionar la opción *Decrypt/Verify* (tercera empezando por la derecha)
3. Seleccionar el fichero a desencriptar
4. Nos autentificamos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --decrypt-files <fichero>*
2. Nos autentificamos mediante el passphrase

GPA

1. Abrir el programa *GPA (Gestor de Archivos)*
2. Menú *Abrir >> Archivo*
3. Seleccionamos la opción *Descifrar*
4. Seleccionar el fichero a desencriptar
5. Nos autentificamos mediante el passphrase

Firma MIME/PGP con el sistema openPGP

¿Qué?

Ya hemos visto cómo funciona el sistema de firmas en openPGP cuando hablamos de la firma de claves. El proceso para firmar cualquier tipo de mensaje es idéntico. Este tipo de firma (**MIME/PGP**) es usada principalmente para firma y autentificación de ficheros.

¿Cómo?

PGP

1. Abrir el programa *PGPmail*
2. Seleccionar la opción *Sign* (tercera empezando por la izquierda)
3. Seleccionar el fichero a firmar
4. Nos autentificamos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --sign <fichero>*
2. Nos autentificamos mediante el passphrase

GPA

1. Abrir el programa *GPA (Gestor de Archivos)*
2. Menú *Abrir >> Archivo*
3. Seleccionamos la opción *Firmar*
4. Seleccionar el fichero a firmar
5. Nos autentificamos mediante el passphrase

Firma ASCII con el sistema openPGP

¿Qué?

Este es el otro tipo de firma, en **armadura ASCII**. Al ser una firma de tipo texto (llamada también firma en texto plano) es usada principalmente en la firma de correos electrónicos y ficheros de texto plano.

¿Cómo?

PGP

1. Abrir el programa *PGPmail*
2. Seleccionar la opción *Sign* (tercera empezando por la izquierda)
3. Seleccionar el fichero a firmar
4. Marcamos la casilla *Text Output*
5. Nos autentificamos mediante el passphrase

GPG

1. Ejecutamos el comando *gpg --clearsign <fichero>*
2. Nos autentificamos mediante el passphrase

GPA

1. Abrir el programa *GPA (Gestor de Archivos)*
2. Menú *Abrir >> Archivo*
3. Seleccionamos la opción *Firmar*
4. Seleccionar el fichero a firmar
5. Marcamos la casilla *Armadura*
6. Nos autentificamos mediante el passphrase

Verificación de firmas con el sistema openPGP

¿Qué?

Igual que ocurría con la firma, la verificación de firmas en sistemas openPGP es igual para cualquier tipo de mensaje. La verificación de firmas sirve tanto para firmas **MIME/PGP** como para firmas de **armadura ASCII**.

¿Cómo?

PGP

1. Abrir el programa *PGPmail*
2. Seleccionar la opción *Decrypt/Verify* (tercera empezando por la derecha)
3. Seleccionar el fichero a verificar

GPG

1. Ejecutamos el comando *gpg --verify <fichero>*

GPA

1. Abrir el programa *GPA (Gestor de Archivos)*
2. Menú *Abrir >> Archivo*
3. Seleccionamos la opción *Verificar*
4. Seleccionar el fichero a verificar

Otras funciones del sistema openPGP

Existen muchas otras opciones en los sistemas openPGP que no he nombrado. Algunas por triviales (por ejemplo la opción de encriptar y firmar) y otras porque no son de uso habitual, entre ellas:

- Generación de subclaves: Excepto para el caso de la generación de la subclave de encriptación en GnuPG, no hemos tocado este tema. Es posible la generación de nuevas subclaves (de encriptación, de firma o mixtas) a distintos nombres y correos electrónicos dentro de una misma clave.
- Borrar claves de un servidor de claves: Es posible (obviamente solo mediante autentificación por clave privada y passphrase) borrar la propia clave pública del servidor de claves.
- Revocar firmas: Existe la posibilidad de revocar una firma que hayamos hecho a una clave pública ajena. Esta opción solo la puede realizar el firmante, nunca el firmado.
- Firmas irrevocables: Igual que existen las firmas exportables y no exportables, existen otro tipo de firmas denominadas firmas no revocables. Como su propio nombre indica, son firmas que jamás podrán ser revocadas, ni por su autor original.

Y existen también algunas opciones más que son menos usuales si cabe. Para una descripción detallada de cada una de ellas, repito lo dicho al principio: leed la documentación del software que uséis.

Seguridad en el sistema PGP

Hoy en día PGP es uno de los sistemas más seguros que existen. Aún así es factible que datos cifrados que creemos seguros no lo sean y hayan sido comprometidos. La mayor parte de las veces estos problemas de seguridad son ajenos al propio sistema PGP, pero aún así es importante tenerlos en cuenta:

Ataques a la algoritmia

Intentar romper el código de cifrado simétrico de PGP es prácticamente imposible. Las algoritmos simétricos usados han demostrado su eficiencia (AES y sus finalistas entro otros), y las claves usadas son generadas aleatoriamente y usadas una única vez, siendo destruidas de forma segura después. Intentar romper por fuerza bruta un algoritmo simétrico de 128 bits con un cluster de mil millones de procesadores que probaran mil millones de claves por segundo cada uno tardaría en dar resultados más del tiempo de vida restante estimado para el universo. Eso convierte la clave simétrica prácticamente en un *secreto perfecto*.

Intentar romper el código de cifrado asimétrico con la tecnología actual es sencillamente impensable. Los cifrados son de 1024 hasta 4096 bits en algoritmos considerablemente complejos y un ataque de fuerza bruta queda totalmente descartado.

Pero hay una parte débil en el eslabón: el passphrase. Claves simétricas de 256 bits y asimétricas de 4096 son seguras, pero el passphrase correcto puede deshacer todo el entuerto sin necesidad de enfrentarse al verdadero cifrado. El hash del passphrase se guarda en la clave privada del par de claves, motivo por el que nuestra clave privada **jamás** debe verse comprometida.

Aún así, con la clave privada, sería bastante difícil obtener el passphrase, para lo cual existen dos alternativas. La primera es la fuerza bruta y puede descartarse casi desde el principio. Si tratáramos de romper por fuerza bruta MD5 (el más débil de los algoritmos de hash) en una máquina que pudiera analizar mil millones de posibilidades por segundo, el tiempo de finalización de la tarea sería de 1,07 x 10^22 años. Aún así, una contraseña débil (de menos de 8 caracteres) podría caer relativamente rápido en un ataque de fuerza bruta. Tratando claves potentes (más de 10 caracteres) el ataque es impensable.

La segunda opción es más factible. Se trata de los ataques de diccionario. Un hipotético atacante en posesión de una clave privada podría idear un algoritmo que probara unos passphrase predefinidos y comparara el hash resultante de cada uno de ellos con el hash de la clave. Este problema se soluciona no usando como passphrase alguna contraseña típica de las que pueden ser encontradas en cualquier diccionario de claves.

Un ejemplo de software de ataque de diccionario contra claves privadas es **PGPcrack**. Es muy antiguo (su última actualización, la 0.6b, data de 1996) y está pensado para el sistema PGP RSA-IDEA-MD5, pudiendo únicamente analizar claves que usen como algoritmo de hash MD5. Aún así, su interés didáctico es alto y recomiendo probarlo.

Existen otros ataques menos comunes al sistema de hash como el **ataque del cumpleaños** contra sistema MD5. Se trata de un sistema puramente estadístico que trata de encontrar una colisión fuerte en el sistema, y por tanto una cadena que devuelva el mismo hash que el passphrase original. Aún así, en las mismas condiciones que estudiamos antes para la fuerza bruta, los 1,07 x 10^22 años se reducen a 585, lo cual sigue resultando insatisfactorio como ataque. En sistemas SHA-1 o RIPEMD-160 este ataque es aún menos útil, pues el tiempo necesario para encontrar una colisión es exponencialmente mayor.

Por último, se ha demostrado efectivo contra el sistema MD5 el criptoanálisis diferencial, pues una modificación del sistema de compresión MD5 puede producir fácilmente colisiones en la salida. Aún así, aplicado al sistema PGP esto no tiene ningún valor, pues esas colisiones serían rechazadas por aceptarse inválidas.

Ataques al software criptográfico

Este otro tipo de ataque es más realista que los lanzados contra la algoritmia del sistema. En este caso se trata de modificar de alguna forma el software openPGP de un sistema para que la información cifrada se vea comprometida prácticamente desde su cifrado. Esta modificación puede venir dada por algún programa añadido o por una modificación y posterior compilación del código original.

Un primer objetivo de esta modificación puede ser la alteración de las rutinas de generación de números aleatorios. Mediante un computador es imposible obtener números aleatorios, por lo que estos sistemas de generación aleatorios se sirven de **generadores de entropía**, que se sirven de métodos como los periodos de tiempo transcurridos entre las pulsaciones del teclado, los cambios de dirección de los ejes de movimiento del ratón, el retraso de los paquetes recibidos por la red... El objetivo de un atacante aquí es modificar estas rutinas de forma que la supuesta generación aleatoria resulte en realidad un proceso predefinido y predecible. Así, saber cuál es (aproximadamente) la clave simétrica de cifrado hace relativamente sencillo el descifrar el mensaje sin necesidad de atacar al cifrado asimétrico.

Otro objetivo es similar al anterior, solo que en lugar de atacar las rutinas de generación de números aleatorios, directamente modifican las rutinas de asignación de clave simétricas. El resultado es más efectivo aún, aunque la modificación es más llamativa.

También existen ataques basados en la sustitución de la implementación de los algoritmos criptográficos por otras versiones modificadas y debilitadas, de forma por ejemplo que ignoren la clave del usuario y utilicen una clave predefinida, o que incluyan una clave “maestra” en el cifrado de datos que permita al atacante descifrar el mensaje sea cual sea el destinatario original del mismo.

Para protegerse de estos ataques es importante cuidar la integridad de nuestro sistema, así como obtener las cadenas hash de los ejecutables de nuestro software de cifrado para comprobar periódicamente la integridad del mismo. El uso de software libre y compilar nuestro propio código previamente comprobado es una gran opción (por ejemplo GnuPG).

Integridad del sistema

Este tipo de ataques generalmente no son un fin por si mismos, sino un medio para lograr otro tipo de ataque, bien sea un ataque al software criptográfico o un intento de recuperación de datos borrados. También pueden ser un ataque directo que tenga por objetivo el robo de la clave privada y la intercepción del passphrase.

Estos ataques suelen realizarse por los medios clásicos de ataque a un sistema operativo: vulnerabilidades, virus, troyanos, keyloggers, sniffers...

Un atacante que logre introducir un backdoor en un sistema puede fácilmente transferirse el fichero de clave privada del usuario e interceptar las pulsaciones del teclado para obtener el passphrase, y así comprometer por completo el sistema de cifrado sin necesidad de escribir complicadas modificaciones de código o lanzar pesados ataques al propio cifrado.

La protección contra estos ataques pasa por unas nociones básicas de seguridad informática.

Información residual

Una de las fuentes de debilidades más importantes, y una de las más desatendidas, es la información residual contenida en un disco duro. Al borrar un fichero, éste permanece en el disco a no ser que los sectores que ocupaba sean sobreescritos con nueva información.

De esta forma, si nosotros guardamos nuestras contraseñas en un fichero convenientemente cifrado y después de editar y añadir nuevas entradas lo ciframos, una incorrecta eliminación del fichero editado compromete toda la información contenida en él. De un disco duro de segunda mano, aparentemente formateado, se puede extraer muchísima información personal y confidencial.

¿Cómo solucionamos este problema? Usando utilidades de **borrado seguro**, para asegurarnos que del espacio libre de nuestro disco duro no se pueda extraer absolutamente ninguna información.

Existen multitud de aplicaciones que aplican sobreescritura de datos en una rápida pasada como método de borrado seguro, pero aunque convierte cualquier intento de recuperación de información en improbable, no es imposible. Es necesario el uso de algoritmos de sobreescritura de datos con complejos algoritmos de truncado y reescritura de datos en varias pasadas. Dentro de estos sistemas "seguros" consideramos los siguientes tipos:

- **Método ISAAC:** Existen métodos de borrado que permiten la reescritura de datos aleatorios generados con **ISAAC** (*Indirection, Shift, Accumulate, Add and Count*) en hasta 65535 pasadas. ISAAC es un generador criptográfico de números aleatorios.
  Más info: http://burtleburtle.net/bob/rand/isaacafa.html
- **DoD 5220-22.M (NISPOM 8-306):** Método basado en el **NISPOM** (*National Industrial Security Program Operating Manual*) del Departamento de Defensa de EE.UU. Se basa en la sobreescritura de 3 pasadas datos aleatorios generados con el generador **ISAAC**.
  Más info: http://www.dss.mil/isec/nispom.htm
- **Método Gutmann:** Éste método ideado por Peter Gutmann se basa en la sobreescritura compleja de 27 pasadas sobre los datos a borrar de forma segura. La recuperación es completamente imposible.
  Más info: http://www.usenix.org/publications/library/proceedings/sec96/full_papers/gutmann/

El software especializado en esta tarea normalmente ofrece dos opciones: limpiar el espacio libre de un disco y borrar de forma segura un fichero. Para mantener nuestro sistema seguro, debemos realizar una limpieza completa del espacio en nuestros discos duros periódicamente, y tomar por costumbre la eliminación de cualquier fichero por métodos seguros. Ejemplos de software de borrado seguro son:

- **Eraser:**
  (*Software libre // GPL ; Sistemas Windows*)
  http://www.heidi.ie/eraser/
  Método DoD 5220-22.M; Método Gutmann + ISAAC; ISAAC hasta 65535 pasadas.
- **GnuPG (Gnu Privacy Guard):**
  (*Software libre // GPL ; Sistemas Windows // Sistemas *NIX // etc*)
  http://www.gnupg.org/
  Método Gutmann.
- **PGP (Pretty Good Privacy):**
  (*Freeware // Propietario ; Sistemas Windows*)
  http://www.pgp.com/
  Método DoD 5220-22.M de 3 a 27 pasadas.
- **Wipe:**
  (*Software libre // GPL ; Sistemas *NIX*)
  http://wipe.sourceforge.net/
  Método Gutmann.
- **Steganos Security Suite:**
  (*De pago // Propietario ; Sistemas Windows*)
  http://www.steganos.com/en/sss/
  Método DoD 5220-22.M; Método Gutmann.

Medios no convencionales

Existen otros métodos no convencionales de ataque a criptosistemas. Algunos son tan sencillos como el espionaje con cámaras de vídeo o micrófonos de una estancia. Sin embargo otros son bastante más complejos.

La modificación del hardware es un método probado. Introducir un sniffer de hardware en un sistema puede ser a veces mucho más efectivo que su homólogo de software. Un teclado modificado con un transmisor de radio puede comprometer un criptosistema completo.

Otro método no muy conocido es **TEMPEST** (http://www.eskimo.com/~joelm/tempest.html). Según este estudio, midiendo las radiaciones electromagnéticas de un monitor desde un lugar remoto es posible reconstruir su imagen al detalle. Algunos expertos opinan que es irrealizable, y otros afirman que es un hecho. Cierto es que, medio en broma, en la generación de claves muy potentes en GnuPG (por ejemplo mi clave es una RSA 4096) el software te advierte que también las radiaciones del monitor y teclado (inalámbrico, se entiende) pueden ser vulnerables a ataques.

Distribución de este documento

Este documento se distribuye en formato PDF realizado bajo OpenOffice.org 1.1.0.

Ficheros a distribuir:

Nombre: "Criptosis.pdf"
Descripción: Documento principal.

Nombre: "Criptosis.pdf.sig"
Descripción: Firma digital PGP del fichero "Criptosis.pdf" realizada por el autor.

Nombre: "hash.txt"
Descripción: Contiene la cadena hash MD5 de los ficheros "Criptosis.pdf" y "Criptosis.pdf.sig".

Datos adicionales:

El hash MD5 puede resultar útil para comprobar la integridad del fichero descargado, pero no es garantía de la inalterabilidad del documento, pues puede haber sido alterado junto a la cadena de hash.

Para comprobar la completa autenticidad e inalterabilidad del fichero, usar el sistema PGP para validar el fichero .sig (MIME/PGP) de firma. Cualquier modificación no autorizada del documento hará que la firma del mismo no sea válida, y ésta es imposible de falsificar.

Death Master

Autentificación:

-----BEGIN PGP MESSAGE-----
qANQR1DBwUwDJoT5ygJgu7ABEACCSz/Cz53005Bh3MzQxVgSK5daoEtCEpuDamVt
5Winx8pgvCYaQt2TuCppi8LPBslVyb/E/TxU4l3CaFNLbTG8HwCyV5EE3ekSJSZZ
Ej7H7gfnkcOBbC1lhD5NjH/ALWFOymY8YyKaaiar9RAJ6DzBzBLp76MLnsEzWOoP
wbSxPv7yduBygTQaa1hw8IrC3a9r3WW5ZZUP08tlW3F/rz7ybBIgXPoDzUb7Q62R
gHKGoknw2ThssBMuWcG+ZPbXaoj/j+V5masX2fkurX2fzk4JLEHg7j7lGXc8xGil
GqwUUq8y7rEAsGAMG6Ot2IWqpBxqiOLlLy2V7hGRVTkeieSurMOTAS4XWNr40Vq8
S4FNKlty+fXEUED4U0PXr7FsPnGaRUHRwFvyBHJWju2JMH/1DTdxRBtcXvhHbNIs
oek+PQbDK3d8tUB2bI+K1Owyq6cBNK+JFLmwv6X2NlTAXm398DSm3dok8J1OuKbJ
lK4RIdMyXeguOv/CQO8JP8OY2hHXjdDrq0r0ksci3ae3lxqme+hgUlACHCAT1Ytt
bpakbINf7qQlxAPjSC4btv9C1k1pq2KSiOR6s0wQh0/L8Re+8OFFVOGxoeUWnbJ8
abF/RfLXdjXue8/sEa7i7nLoweO4w1l55PjdLDP03J//7Ht9J31wSL2HX4Ba4DCw
hcOlTdLCkwFSRdJczCcdWQpxT/k7fHbZGc+Ne8BewJxdfuA9LcnTTfo8FlkWEdtZ
pjHEMH2WpnjsiQWuqKAik3PHbL++67MaEXbod0pKxTUjR1/sJaW1l7YKNkrzF86J
7dGrc8YwCL9QfpQpUqfxHydrZAmDNT30NyrLF57XGh+aygB7ZQ8Ol7J1Yr6aGQ+Z
ZtTSqGdxkHbD4aSCfY/ur/koMiWMTTNN5A95HaZiLT9uEa3eKwHowudDsCXv88Pt
TpQ0CbkvokV5kAZa0OxndOd8N8NVrGf96GNuroqPSrl8j9GsdefmudaLH8CKgVoc
KwF7+sYD1LBB+/mfRJufgTAXbq/xvuCMBxAY5dsYQnqD0Na3FBH9dyejva31Tvp8
HK2FVwBrj7NTVNkhcpciO0rBOKNTcmvgpo+qWIGVcxKaX4hcZH4y2m+hzQz9AizW
WCGLFbmEq+AE7pHWqORcQ0ntVutNXhrlrCgTAHagOlrpHvcbNY75aVOkw3Aa5jpM
aSgqMffoG9vMj2yYGk2OMnkl90jXhWTzeYccVvsJuCqa2HDw51V2rFEZXJBt9Pny
nM39qm7CDs6hwd+aUhF9oZhO06a3uSuZScyE8GkIK0Ki59KzyCV9Bv5TS4Nh8l9c
e+jjSildy3ANrbbHm8aS3MWQvFFQGM3ac5eC4RpIzAu89pysYc8CTpGEpjTai+VG
vHM7PLtCMEucDftYCSCarQcxGwoUuA/jAmxjeCKtNFnZ2OiVyfp2w3YOjY2EJhH6
fAa/RVWY4/fWJdoM0Ka+mknT5OjTcV8rcWK3pYjXbf7V4uRmKZNj/gX2GqFKSlve
AnacBAZqB/HgNnW7JjTt6jDf8Igb/5hu0R1qmms5CCGQc12OQEL1UbXe0xZ4Q003
b1RWgy23RcSZjkOdp3rOVJxEm9mGJYZkwcM8MEkThFpO5T0Tf+Ai5LZ5OgcbCE79
uPtPbMjzE4TFirDSwvwy2dmGTjuk6A9x39hyToJl2Bz193q1Djci19K4Tqp7cObz
l7KxklHpCM15X2m4b6GPg9n2TNHWXuuF13MlZzf5PbQk+6LIJP98HgY2iIo1PGTs
PHcKDbj3zqFQj6PkKD/B30Q3WXxoubHFNvW0ipOyW6H77FmzpKYjc/o1
=Ad78
-----END PGP MESSAGE-----

Licencia

Criptosistemas Informáticos – #hackxcrack @ irc.irc-chatpolis.com – Versión 1.0

Este documento contiene los temas tratados, así como información adicional relacionada con la **Charla de Criptosistemas Informáticos** expuesta el día 28 de Febrero de 2004 a las 22.00 en el **canal de IRC del foro hackxcrack** (#hackxcrack @ irc.irc-chatpolis.com) por **Death Master**.

Computer Cryptosystems – #hackxcrack @ irc.irc-chatpolis.com – Version 1.0

This document contains the treated topics, as well as additional information related with the **Computer Cryptosystems Speech** that took place the 28th February 2004 at 22.00 in the **hackxcrack forum's IRC channel** (#hackxcrack @ irc.irc-chatpolis.com) by **Death Master**.

*** End Of File ***