

1. ¿Qué es la criptografía?

Criptografía es una palabra que proviene de las palabras griegas criptos y graphos. Literalmente significa “escritura oculta”, es decir, la criptografía es el arte de ocultar la información de forma que sólo pueda verla aquel receptor que el emisor desea.

Hoy en día la criptografía es fundamental para el desarrollo de las tecnologías de la información y las comunicaciones, puesto que las redes (y particularmente Internet) han pasado de ser un invento de “cuatro amigos”, a estar expuestas a todo tipo de ataques, “piratas” y violaciones de la privacidad que se puedan imaginar. Por lo tanto son necesarios métodos criptográficos eficientes y robustos para mantener la privacidad.

Aquí simplemente trataremos los aspectos prácticos de la criptografía, es decir, como cifrar nuestros datos y hacer que las demás personas los puedan obtener, dando unas pinceladas antes de los fundamentos teóricos que sustentan la criptografía. También trataremos otra aplicación de la criptografía, que es la autenticación de entidades. No entraremos en los métodos de criptoanálisis (es decir, cómo romper los crifrados, lo opuesto a la criptografía) salvo de forma meramente anecdótica, puesto que no son de interés para la mayoría de usuarios de ordenadores.

2. Fundamentos teóricos de la criptografía

La criptografía se fundamenta en dos ramas de las matemáticas: la matemática discreta (o teoría de números) y la teoría de la información. Los fundamentos de ésta última los sentó Claude Shannon con la publicación de su famoso artículo “*A Mathematical Theory of Communication*”. Otras obras claves para el desarrollo de la criptografía fueron “*Communication Theory of Secrecy Systems*”, también de Shannon, y “*New Directions in Cryptography*” de Whitfield Diffie y Martin Hellman, en el que introducían la criptografía asimétrica.

2.1. Criptosistemas

Un criptosistema es la agrupación de:

- Un conjunto de mensajes sin cifrar (M).
- Un conjunto de mensajes cifrados (C).
- Un conjunto de claves (K).
- Un conjunto de transformaciones de cifrado (E_k).
- Un conjunto de transformaciones de descifrado (D_k).

Cualquier criptosistema debe cumplir la condición:

$$D_k(E_k(m))=m$$

Es decir, si aplicamos la transformación de descifrado al mensaje cifrado, usando la clave adecuada (k), obtenemos de nuevo el mensaje en claro.

Debemos tener cuidado con nuestros criptosistemas, puesto que se pueden dar condiciones indeseadas. Por ejemplo, puede haber una clave k tal que $E_k(E_k(m))=m$, es decir simplemente tenemos que volver a cifrar el mensaje para obtenerlo en claro. Incluso puede haber casos en los que $E_k(M)=M$, lo cual es peor todavía, porque estaríamos transmitiendo el mensaje en claro. Claro está que esto depende del criptosistema en concreto, de forma que en un criptosistema bien diseñado (es decir, unas transformaciones bien diseñadas, que es en lo que tenemos que decidir) el número de claves de este tipo (claves débiles) tiende a cero.

En función del uso de las claves, podemos distinguir los criptosistemas en dos grandes grupos:

- Criptosistemas simétricos: se usa la misma clave k para el cifrado y el descifrado. Presentan el inconveniente de que antes hay que distribuir la clave para poder tener una transmisión segura, con lo cual nos planteamos cómo distribuir la clave de forma segura.
- Criptosistemas asimétricos: se usan dos claves distintas para el cifrado y el descifrado, que son la clave pública k_p y la clave privada o secreta k_s . Estos criptosistemas no presentan el problema de la distribución de claves, pero son mucho más costosos computacionalmente. También se pueden usar para la autenticación.

Habitualmente los mensajes se codifican con criptografía simétrica, y para transmitir las claves simétricas se usa criptografía asimétrica.

2.3. Cantidad de información y Entropía

Dada una variable aleatoria discreta X de rango $\{x_i\}$ $i=1\dots n$ se define la cantidad de información de cada evento x_i como:

$$I_i(X) = -\log(P(x_i))$$

Esto quiere decir que si hay un evento con una alta probabilidad, nos proporcionará poca información que ocurra (de antemano estábamos casi seguros de que iba a ocurrir eso), y si hay poca probabilidad de que ocurra, nos aportará mucha información, de forma que si ocurre algo imposible, nos aportará una infinita cantidad de información, y si ocurre algo seguro, no nos aportará ninguna información.

Si promediamos la cantidad de información de cada evento, lo que obtenemos es la entropía de la variable aleatoria:

$$H(X) = -\sum_{i=0}^{N-1} P(x_i) \log(P(x_i))$$

Donde N es el número de valores de la variable aleatoria.

La entropía puede ser interpretada como el número medio de símbolos necesarios para representar cada valor de la variable aleatoria.

2.4. Números primos y factorización

Los números primos son aquellos enteros tales que son divisibles sólo por si mismos y por la unidad. Muchos algoritmos criptográficos se basan en este tipo de números debido a su difícil tratamiento.

La factorización es el proceso inverso a la multiplicación, es decir, dado un número x, obtener otros números tales que al multiplicarlos obtenemos x.

Habitualmente se buscan dos primos grandes y después se multiplican, de forma que tenemos que conocer alguno de los factores para conocer el otro (suponiendo que ya sabemos x). La fuerza de estos métodos reside en que no existe una solución eficiente para la obtención de los factores conocido x.

2.5. Logaritmos discretos

Otros algoritmos utilizan exponenciaciones dentro de grupos finitos para codificar. Se usan números de muchos bits tanto en las bases como en los exponentes. Aunque existen algoritmos eficiente para realizar estas exponenciaciones (realizarlas a base de multiplicaciones resultaría inviable), no hay métodos eficientes para resolver los logaritmos discretos, es decir:

$$c = \log_b(a) \pmod{n} \Leftrightarrow a \equiv b^c \pmod{n}$$

Los algoritmos que usan exponenciaciones basan su éxito en esta ineficiencia. Este problema está muy ligado con el de factorización, de hecho, se demuestra que si se puede realizar el logaritmo discreto, se puede factorizar fácilmente, aunque todavía no se ha podido demostrar el recíproco.

2.5. Secuencias aleatorias

Como se ha dicho anteriormente, en muchos sistemas de cifrado lo que se hace es usar criptografía simétrica para el intercambio de mensajes, y enviar primero la clave simétrica usando criptografía asimétrica. Las claves asimétricas suelen ser claves generadas en el momento de forma aleatoria, llamadas también por ello *claves de sesión*.

Las claves de sesión deben ser completamente aleatorias, ya que si un atacante pudiese adivinar cuál es la clave que vamos a usar en una determinada comunicación, nuestra seguridad estaría comprometida.

Cuando hablamos de computadoras, no podemos hablar de aleatoriedad total, puesto que en teoría un ordenador es una máquina de estados regida por unas variables bien definidas, y por tanto determinista.

Veremos tres tipos de secuencias aleatorias, aunque no entraremos en las descripciones de los generadores aleatorios, por estar fuera del alcance de este documento.

Secuencias pseudoaleatorias

Los generadores pseudoaleatorios producen secuencias **finitas** y **periódicas** de números según la semilla dada. Lo único que podemos conseguir con estos generadores son secuencias de largo periodo, pero son totalmente inservibles en criptografía porque son **totalmente predecibles**. Estas secuencias también son llamadas *secuencias estadísticamente aleatorias*.

Secuencias criptográficamente aleatorias

Una secuencia criptográficamente aleatorias es aquella en la que no se puede predecir el siguiente valor que va a tomar, aún conociendo los valores anteriores y el algoritmo de generación. Realmente lo que se busca es que sea computacionalmente intratable ese problema, aunque en teoría se pueda predecir.

Estas secuencias aún tienen un problema, y es que necesitan una semilla, y si un atacante consigue averiguar esta semilla, podría reproducir la secuencia. Para la semilla necesitamos un número totalmente aleatorio.

Secuencias totalmente aleatorias

Aunque como ya hemos dicho no podemos hablar de total aleatoriedad en un computador, definiremos un nuevo tipo de secuencias. Una secuencia es totalmente aleatoria si no puede ser reproducida de manera fiable.

Bien, entonces, ¿qué necesitamos realmente? ¿números realmente aleatorios? No. Lo que realmente queremos son secuencias irreproducibles e impredecibles. Para conseguir esto, simplemente tenemos que emplear un generador criptográficamente aleatorio alimentado por una semilla totalmente aleatoria.

3. Algoritmos.

3.1. Algoritmos simétricos

Estudiaremos ahora los algoritmos de cifrado simétricos, y dentro de éstos un conjunto específico: los cifrados por bloques.

Los cifrados por bloques dividen el mensaje en partes de longitud fija, y le aplican dos transformaciones básicas: la confusión y la difusión. La confusión consiste en ocultar la relación entre el mensaje en claro, el mensaje cifrado y la clave, de forma que sea difícil establecer relaciones entre los tres. La difusión consiste en distribuir la influencia de cada bit por todo el mensaje, de forma que cambiar un bit en el mensaje en claro, provoque un gran cambio en el mensaje cifrado. En principio la confusión sería suficiente si almacenamos suficientes pares de bloque en claro y bloque cifrado (todos los posibles), pero esto resulta inviable en cuanto nuestro tamaño de bloque crece, así que lo que se hace es mezclar la confusión con patrones pequeños con la difusión. Los algoritmos que hacen esto se denominan *algoritmos de producto*.

Un elemento fundamental de un algoritmo por bloques son las S-cajas. Una S-caja es un sistema que coge un determinado número de bits, les aplica una sustitución simple y devuelve otra cadena de bits, no necesariamente del mismo tamaño. Las S-cajas se denotan como S-cajas de entrada x salida, por ejemplo una S-caja que recoge 8 bits y entrega 12, es una S-caja 8x12. Las S-cajas aplican las transformaciones de sustitución sobre el mensaje (la confusión).

Otro elemento muy común en los algoritmos de producto son las Redes de Feistel. Muchos algoritmos (como DES o Blowfish), dividen los bloques en dos mitades, L y R (de *Left* y *Right*) y luego siguen el proceso siguiente de forma iterativa n veces:

$$\left. \begin{array}{l} L_i = R_{i-1} \\ R_i = L_i \oplus f(R_{i-1}, K_i) \end{array} \right\} \text{ para } i < n$$

$$\left. \begin{array}{l} L_n = L_{n-1} \oplus f(R_{n-1}, K_n) \\ R_n = R_{n-1} \end{array} \right\} \text{ para } i = n$$

La función f que aparece es propia de cada algoritmo y será una de sus características esenciales. Para descifrar el bloque simplemente tenemos que aplicar la misma transformación pero usando las claves en orden inverso (es decir empezando por K_n y terminando por K_1).

El algoritmo DES

DES es un algoritmo de producto que usa bloques de 64 bits con claves de 56 bits. Consta de una Red de Feistel de 16 rondas, y dos permutaciones que son inversas ($P_i = P_i^{-1}$), una al principio y otra al final. En la función f se usan ocho S-cajas de 6x4 bits.

Hasta el momento no se ha descubierto ningún problema de diseño en DES, de forma que sigue siendo válido, pero los avances tecnológicos han hecho que sea sencillo romperlo mediante fuerza bruta debido a su pequeña longitud de clave. Consta de algunas claves débiles, aunque su número es despreciable frente al número total de claves.

Debido a su "caída", han surgido distintas variantes de DES. De ellas la única que da mejor resultado es la del Triple DES, que consiste usar dos claves distintas K_1 y K_2 y hacer lo siguiente:

$$C = E_{k_1}(D_{k_2}(E_{k_1}(M)))$$

Es decir ciframos con la clave 1, desciframos con la 2, y volvemos a cifrar con la 1. De esta forma la clave resultante es la concatenación de K_1 y K_2 y obtenemos una clave de 112 bits.

El algoritmo Blowfish

Blowfish es un algoritmo propuesto como sustituto del DES para aplicaciones no militares y libre de patentes. También está diseñado para usar sólo operaciones elementales computacionalmente hablando (XOR, suma e indexación de arrays).

Utiliza bloques de 64 bits con un tamaño de clave variable de hasta 448 bits. El algoritmo tiene dos etapas: una en la que genera todas las claves necesarias para las iteraciones y otra en la que cifra propiamente. Al igual que DES usa una red de Feistel de 16 iteraciones. En cada iteración sólo realiza sumas y XORs y cuatro indexaciones de arrays. También al igual que en DES, basta aplicar el algoritmo de nuevo usando las claves en orden inverso para descifrar.

El algoritmo IDEA

IDEA es otro algoritmo de producto datado en 1992. Usa bloques de 64 bits y claves de 128 bits. Al igual que Blowfish, utiliza operaciones de bajo coste computacional (XOR, suma módulo 2^{16} y producto módulo $2^{16} + 1$).

No tiene claves débiles, aunque si algunas que pueden dar cierta ventaja a la hora del criptoanálisis, sin embargo encontrárselas es prácticamente imposible (probabilidad 2^{-96}).

3.2. Algoritmos asimétricos

Este tipo de algoritmos fueron introducidos en los años 70 por Whitfield Diffie y Martin Hellman. La diferencia con los anteriores es que usan un par de claves para cifrar la comunicación, en vez de sólo una. A dichas claves se les llama *clave privada* y *clave pública*, K_s y K_p respectivamente. Existen muchos algoritmos asimétricos pero muchos de ellos resultan o inseguros o impracticables, ya sea por su elevada longitud de clave o porque el mensaje cifrado es bastante más largo que el mensaje sin cifrar. Pocos algoritmos han resultado ser realmente útiles, y menos aún conocidos. El más conocido, por ser el primero que se incorporó en PGP (la primera herramienta criptográfica para el usuario de a pie) es RSA. Otros algoritmos como ElGamal o DSA también están teniendo éxito.

Las claves usadas por estos algoritmos son siempre mucho más largas que las de los algoritmos simétricos. Si antes hablábamos de claves de 128 bits como muy seguras, ahora hablaremos de claves que rondan los 1024 bits o los sobrepasan. También son computacionalmente mucho más costosos.

Aplicaciones de los algoritmos asimétricos

Estos algoritmos sirven a dos propósitos: la protección de la información (como los simétricos) y la autenticación del mensaje.

La protección de la información se lleva a cabo cifrando el mensaje con una de las claves y descifrándolo con la otra. Supongamos que B quiere enviarle un mensaje a A. Entonces A le dará a B su clave pública, B cifrará el mensaje con ella y se lo enviará a A, que lo descifrá con su clave privada (fig. 1).

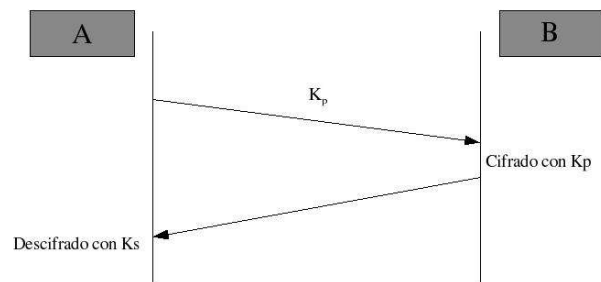


Figura 1. B quiere enviarle un mensaje a A. Para ello debe conocer primero la clave pública de éste.

La autenticación tiene un par de pasos más. Ahora se usa a mayores una función resumen (*hash*), es decir una función que proporciona un pequeño conjunto de bits a partir del mensaje, mucho menor que éste, y que es casi exclusivo (es muy difícil encontrar otro mensaje para el que la función resumen dé el mismo resultado). Supongamos que ahora es B quien recibe un mensaje de A y quiere certificar que realmente es de A. Como B conoce la clave pública de A, A le envía un *hash* del mensaje cifrado con su clave privada, B lo descifra con la clave pública de A y lo compara con su propio *hash*. Si son iguales el mensaje es auténtico, en caso contrario ha habido algún problema (fig. 2).

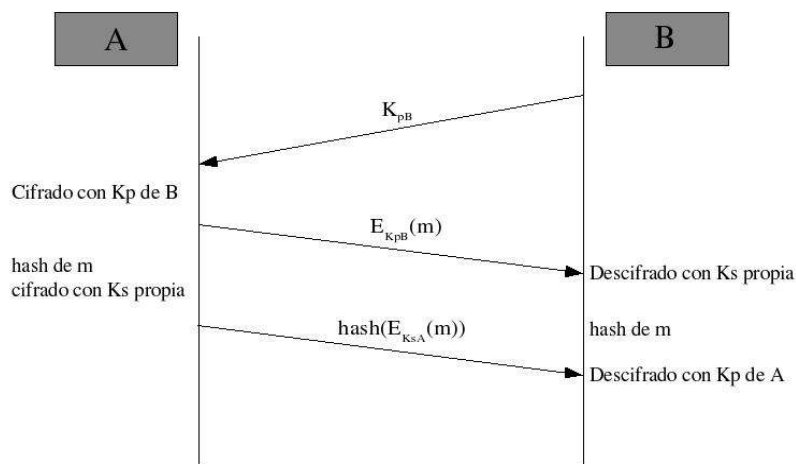


Figura 2. Ahora el remitente envía además un hash cifrado con su clave privada

En el párrafo anterior se pone de manifiesto una de las propiedades más curiosas de los algoritmos simétricos: dado un par de claves, si una se usa para codificar, la otra se puede usar para decodificar, da igual cuál se use para cada cosa.

El algoritmo RSA

Este es el algoritmo más famoso (quizá por la controversia que le rodea) y más usado actualmente para criptografía asimétrica. Hasta el año 2000 estuvo bajo patente de los Laboratorios RSA, de modo que se dejó de usar en favor de otros algoritmos libres hasta que estuvo libre de patentes.

Su funcionamiento se basa en el problema de factorización comentado en la primera sección. Para obtener las dos claves, primero se buscan dos primos grandes y se multiplican, con lo cual el atacante se enfrentará al problema de encontrar esos dos factores, problema que evidentemente es intratable (sino el algoritmo no serviría de nada), o a un problema de

logaritmos discretos (también intratable).

Respecto a su seguridad, realmente no hai nada demostado: ni que sea seguro ni que deje de serlo. Simplemente se confía en la dificultad de la factorización y de los logaritmos discretos, problemas que podrían dejar de ser “problemáticos” si avanza la computación cuántica. En teoría puede haber problemas en cuanto a los primos generados para obtener la clave, ya que estos números son generados con pruebas estadísticas, es decir, no se sabe a ciencia cierta si el número generado es primo, sino que se sabe que lo es con alta probabilidad. Si lo números generados no son realmente primos, el algoritmo simplemente deja de funcionar.

Existen también varias vulnerabilidades:

- Claves demasiado cortas: si la clave es menor de 768 bits, se podría llegar a descifrar el mensaje en un lapso de tiempo suficiente. Actualmente se recomiendan claves de al menos 1024 bits.
- Claves débiles: en RSA siempre existe un número de claves que dejan el mensaje tal como estaba. Realmente el número de estas claves es muy pequeño y no suponen un problema.
- Ataques de suplantación de identidad (*man in the middle*): Un atacante podría situarse entre los dos extremos de la comunicación, e intercambiar las claves que se envían ambos extremos introduciendo las suyas. Esto provoca que ni A ni B se enteren realmente de que alguien está escuchando sus mensajes (o incluso modificándolos). Para solucionar estos ataques, se han implementado grupos llamados *anillos de confianza*, que son grupos de personas que firman sus claves para garantizar su autenticidad (lo veremos más adelante). Este ataque no sólo afecta a RSA sino a cualquier otro algoritmo asimétrico.

4. GnuPG

Veremos ahora cómo instalar y configurar el paquete de criptografía GnuPG. Este paquete nos permitirá cifrar y firmar digitalmente nuestros correos, además de implementar también los anillos de confianza.

Este paquete usa los algoritmos asimétricos DSA y El Gamal. No usa RSA por haber estado sujeto a patentes en su día.

4.1 Instalación

Para instalar el paquete en Ubuntu, no tenemos más que hacer:

```
apt-get install gnupg
```

Si queremos instalarlo desde los fuentes, necesitaremos un compilador (gcc), las herramientas binutils, y el paquete make. Después sólo tenemos que seguir las instrucciones que podemos encontrar en http://webber.dewinter.com/gnupg_howto/spanish/gpgminicomo-2.html#ss2.1

4.2 Generando las claves

Una vez tenemos instalado el paquete, lo primero que tenemos que hacer es generar nuestras claves pública y privada. Para ello simplemente tenemos que ejecutar:

```
gpg --gen-key
```

El programa nos hará varias cuestiones:

- Tipo de clave. El tipo por defecto (DSA/El Gamal) servirá para nuestros propósitos.
- Tamaño de clave. A mayor tamaño más seguridad. Menos de 768 bits puede ser inseguro, y 2048 puede llevar a un comportamiento lento. El tamaño por defecto, 1024, es adecuado para nuestros propósitos.
- Caducidad de la clave. Para mayor seguridad podemos renovar nuestra clave periódicamente, aunque actualmente si elegimos una clave de 1024 bits o más no deberíamos tener problema alguno.
- La información para generar el identificador de usuario: Nombre, e-mail y comentario. El identificador se formará así: Nombre (comentario) <e-mail>
- Una contraseña para proteger la clave.

Es importante que durante la generación de la clave hagamos otras cosas, tengamos otros procesos abiertos, etc. Esto permitirá al generador de claves obtener datos aleatorios de calidad, y así aumentar la entropía de la clave.

Después de generar la clave ésta quedará añadida a nuestros anillos de claves `$HOME/.gnupg/pubring.gpg` y `$HOME/.gnupg/secring.gpg`.

Uno de los datos que nos proporciona el programa al generar la clave es el fingerprint o huella dactilar. Este dato nos sirve para autenticar la clave de otras personas y así no caer en ataques de intermediario o man-in-the-middle. Cuando le demos a alguien nuestra clave, debemos darle a través de un medio seguro, nuestro fingerprint, para que el receptor sepa que realmente la clave que recibió es nuestra. Del mismo modo cuando nos den una clave debemos autenticarla con su correspondiente huella.

4.3 Uso

Una vez tenemos las claves, lo que tenemos que hacer es distribuir nuestra clave, recibir claves de otras personas e integrar GnuPG en el cliente de correo.

Para distribuir nuestra clave primero debemos exportarla. En principio sólo tendremos nuestra propia clave en el fichero de claves públicas, con lo que bastará hacer:

```
gpg --export -a
```

El parámetro `-a` hará que la salida esté en formato ASCII. Como sólo está nuestra clave, la salida será ésta. Si tenemos más claves simplemente tenemos que indicar el nombre que especificamos al generar la clave.

Para incorporar a nuestro fichero de claves nuevas claves simplemente tenemos que hacer:

```
gpg --import nombre_fichero_de_claves
```

Una vez tenemos otras claves, tenemos que firmarlas. Esto sirve para que el programa sepa si se puede fiar de determinadas claves o no. Por defecto las claves no están firmadas y por tanto no se confía en ellas.

Integración con Mozilla Thunderbird

Para usar GnuPG con Mozilla Thunderbird debemos instalar el paquete `mozilla-thunderbird-enigmail`. Una vez instalado, configurarlo es una tarea sencilla:

- Abrimos las propiedades de las cuentas (*Herramientas -> cuentas*) y seleccionamos la cuenta sobre la que queremos aplicar el cifrado y firma.
- Seleccionamos el apartado *OpenPGP Security*. Aquí debemos habilitar la seguridad OpenPGP.
- En las opciones inferiores podemos especificar si la clave la seleccionamos nosotros manualmente o la selecciona el programa identificando el propio mail de la cuenta, y si queremos que se cifren y/o firmen todos los mensajes.
- En las opciones avanzadas simplemente tenemos que tocar la ruta del ejecutable de GnuPG si lo hemos instalado en algún sitio distinto del sitio por defecto (`/usr/bin/gpg`).

A partir de este momento todos nuestros mensajes serán cifrados/firmados si así lo hemos elegido. Cada vez que recibamos un mensaje cifrado, el programa nos pedirá la frase de paso para abrir la clave privada y descifrarlo.

5. Cryptoloop

Desde la rama 2.6 del kernel, éste dispone de soporte para el uso de sistemas de archivos cifrados. Esta aplicación es especialmente interesante para el transporte de datos en unidades portátiles (pendrives, tarjetas SD, disquetes, Cds, DVDs, discos USB, etc), puesto que si los perdemos, cualquier atacante podría apoderarse de nuestros datos (o sin ser un atacante, puede ser una ex-novia furiosa...).

Cryptoloop ha sido la implementación elegida para el kernel 2.6 hasta la última versión a día de hoy. En su lugar a partir de ahora se usará dm-crypt, que usa el *device mapper*. Sin embargo, aún no existen herramientas de espacio de usuario para usar esta característica, así que de momento nos quedaremos con cryptoloop.

El ejemplo que veremos aquí será para cifrar un sistema de archivos de una unidad portable, concretamente un pendrive. Cifrar cualquier otro sistema de archivos es una tarea similar.

5.1. Configuración del kernel y parcheo de utilidades

Para que esto funcione necesitamos habilitar las funciones de la Cryptoloop API en la configuración del kernel. Después debemos elegir que algoritmo queremos usar. La recomendación es que los pongamos todos como módulos, por si algún día queremos usar otro distinto del habitual (AES).

Después de compilar e instalar el nuevo kernel, debemos parchear las utilidades de usuario necesarias. Éstas son las del paquete *util-linux*, especialmente *losetup*. Podemos obtener los parches en:

- *losetup*: <http://www.stwing.org/~sluskyb/util-linux/losetup-combined.patch>
- *util-linux*: <http://www.ece.cmu.edu/~rholzer/cryptoloop/util-linux-2.12-kernel-2.6.patch>

Para aplicar los parches seguiremos el proceso habitual y después instalamos el paquete con los parches aplicados.

5.2. Preparación del sistema de archivos

Ahora crearemos nuestro sistema de archivos cifrado usando un cifrado simétrico AES. AES es uno de los algoritmos más seguros actualmente, ya que ha sido muy analizado y no se han encontrado debilidades serias. También podríamos usar Triple-DES, aunque no ganaríamos nada, sólo perderíamos en velocidad.

Estos son los pasos que debemos seguir para crear el sistema cifrado:

- Rellenar la partición con datos aleatorios. Este paso no es estrictamente necesario, pero nos ayuda a aumentar la seguridad, puesto que a un atacante le será más complicado buscar patrones en el sistema de archivos.

```
dd if=/dev/random of=/dev/sda1 bs=1M
```

- Seleccionar cifrado y tamaño de clave. Como dice más arriba, la recomendación es usar AES y una clave de 512 bits.
- Asociar un *loop device* a la partición que queremos cifrar. De esta forma las peticiones a esa partición pasarán primero por el loop device. Hay que tener en cuenta que debemos tener activado el soporte de loop device en el kernel, y si lo

hemos puesto como módulo tener el módulo cargado (seguramente el kernel lo cargue automáticamente).

```
losetup -e aes-512 /dev/loop0 /dev/sda1
```

El sistema nos pedirá una clave. Debemos poner una clave difícil de adivinar, para evitar ataques de diccionario o fuerza bruta. La clave que pongamos se usa para generar la clave de cifrado mediante un *hash*, así que no podremos cambiarla fácilmente más tarde.

- Crear el sistema de ficheros. Aquí se usa reiserfs, pero se puede usar el sistema que más guste.

```
mkfs.reiserfs /dev/loop0
```

- Deshacer la asociación con el *loop device*.

```
losetup -d /dev/loop0
```

5.3. Uso del sistema cifrado

Para montar el sistema, ahora debemos indicarle que queremos usar el *loop device* en lugar de la partición original.

```
mount -t reiserfs /dev/sda1 /mnt/pendrive -oencryption=aes-512
```

El sistema nos pedirá el password. Si lo ponemos mal simplemente obtendremos un sistema ilegible, no es que el sistema compruebe la password (porque no la guarda en ningún sitio). Esto es porque lo único que se hace es hacerle el *hash* a la contraseña y descifrar. Si sale algo coherente bien, sino se ha puesto mal.

Si se quiere se puede añadir al *fstab* con las mismas opciones que en el *mount*.