Plataformas de cifrado en Linux

FICHEROS 100% SEGUROS

Que nuestras fotos ya no se puedan tocar sino ver, y que nuestras cartas se puedan almacenar pero no doblar, son sin duda dos de los síntomas más significativos que hacen que tomemos consciencia de que nuestra vida está plenamente integrada en el mundo digital. Fragmentos de nuestras vidas son almacenados diariamente en los ordenadores personales.

POR DAVID GASCÓN CABREJAS

Esta situación puede hacer que el robo del portátil o de un disco duro se convierta en un problema mucho mayor que el derivado de la simple pérdida de los dispositivos, por ello cada día más se requiere el uso de sistemas de cifrado a nivel personal.

Está claro que en última instancia somos nosotros mismos los que debemos proteger la información que poseemos. En este artículo se expone cómo usar las distintas herramientas.

Conceptos Previos

Inicialmente vamos a introducir los conceptos del estado de los datos en cada momento. En la Figura 1 podemos ver las distintas fases por las que pueden pasar los datos en su proceso de escritura a un sistema de ficheros cifrado. En ella se exponen casos como el volcado de la memoria RAM a disco, o su paso por los distintos módulos de cifrado, tanto a nivel de Kernel como de espacio de usuario. **Importante:** Haz copias de seguridad de la información y haz pruebas antes de realizar el cifrado sobre los datos reales para no perder nada por posibles errores o malfuncionamiento.

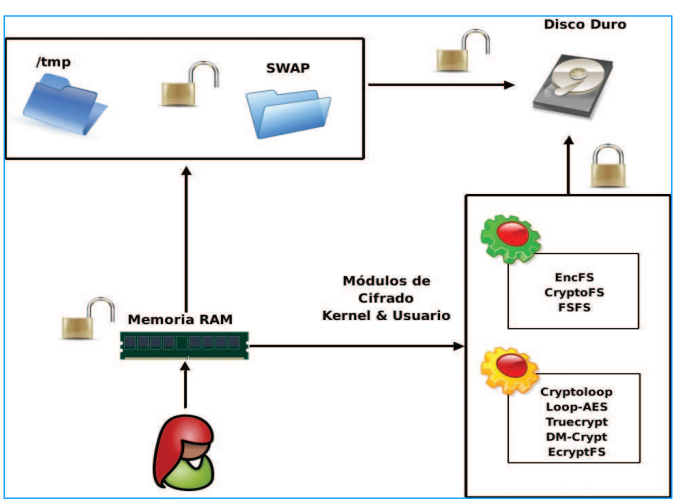

Figura 1: Estado de los datos en cada momento de su almacenaje.

- **RAM (Random Memory Area)**: Por aquí pasa toda la información, tanto cifrada como sin cifrar. Ej: Abrimos con un editor de textos un fichero de texto alojado en un sistema de ficheros cifrado. Cuando abrimos el fichero los bytes leídos pasan a RAM cifrados, de aquí son leídos por la aplicación de descifrado y volcados de nuevo a RAM por ella en texto plano (sin cifrar). Después son nuevamente leídos por la aplicación que el usuario había ejecutado (en nuestro caso el editor de textos). Esto es así porque todos los bytes que se mueven por nuestra máquina siguen un proceso complejo de jerarquías que se sitúan entre el disco duro y el usuario (o la aplicación que éste use). Concretamente, cada bit leído de disco pasa por la memoria RAM, y por todos los niveles de la cache del procesador L1,L2,L3…, antes de llegar a la unidad de procesado. En todos los pasos, la información es tratada tanto cifrada como sin cifrar. En principio esto no es un problema, pero podría serlo, puesto que hay veces en que la información contenida en RAM puede ser volcada a disco.
- **SWAP (Memory Paging)**: EL espacio SWAP es un espacio usado por el sistema cuando se necesita liberar espacio en RAM y cuando se aplican los procesos de suspensión o hibernación (técnicas de ahorro de energía) de una máquina (normalmente dispositivos con baterías como portátiles), donde se aplica un volcado completo de la memoria RAM a disco. Como ya hemos dicho, en RAM se encuentran los datos tanto cifrados como sin cifrar, por lo que puede darse el caso de volcado de la información en claro a disco.
- **Ficheros DUMP (Memory Paging) y directorio /tmp**: Cuando estamos trabajando con una aplicación y se da un error, es posible que se realice un volcado de la pila de ejecución (almacenada en RAM) del proceso a disco con la finalidad de que puedan ser estudiadas las causas del fallo. Un ejemplo son los ficheros denominados Core Dump, que son un caso más de las posibilidades de

volcado a disco de información sin cifrar desde RAM. En otros casos el sistema utiliza la creación de ficheros temporales (muchas veces almacenados en la carpeta /tmp) para controlar aspectos como qué fichero está abierto, o incluso información del mismo usada en ejecución mediante el uso de alguna aplicación.

Preparación del disco a cifrar

La única forma de eliminar para siempre la información de una parte del disco es sobreescribir esa zona con información aleatoria repetidas veces, de forma que no se pueda volver a obtener ni mediante métodos complejos de técnicas de lectura de polaridad magnética con el uso de fuerzas microscópicas (magnetic force microscopy ¿ mfm). Podéis encontrar más información en [1]. Este proceso se realiza en 2 escenarios:

- **Antes** de crear la partición cifrada: En el espacio de disco que vamos a crear nuestro sistema de ficheros, de forma que un análisis del mismo nunca pueda dar información sobre cuanto espacio está siendo utilizado con información cifrada.
- **Después** de cifrar la información: En el disco donde guardábamos la información sin cifrar, de forma que sea eliminada y sólo quede su copia en el espacio cifrado.

En el primer escenario la sobre escritura podemos realizarla sólo 1 vez, sin embargo, en el segundo, para sea efectiva, podemos crear un script que realice la tarea varias veces (en el ejemplo 3). Hay que especificar el tamaño del disco (en el ejemplo 1GB). El parámetro "of" nos indica dónde se sobreescribirán los datos, ya sea un disco (hda, hdb), una partición (hda1, hdb3), un dispositivo externo (sda1, sdb) o un fichero (/root/fichero).

```
veces=3
while [ $veces -ge 1 ]
do
   dd if=/dev/urandom➥
   of=/dev/hdb bs=1M count=1024
   veces=`expr $veces - 1`
done
```

La fuente de extracción de información aleatoria puede ser /dev/urandom o /dev/random. Con la primera fuente se genera antes, pues usa datos internos del sistema para generar la información aleatoria. Con el segundo la información posee mayor entropía (aleatoriedad, impredecibilidad), aunque esta fuente de números aleatorios sólo la genera cuando el sistema produce información no conocida a priori. Para hacer una prueba ejecuta *#cat /dev/random* y mueve el ratón, cuanto más rápido, más información nueva le das al sistema y más rápido puede generar nueva información aleatoria. Sin embargo, para el ejemplo actual necesitamos generar muchísima, por lo que usaremos la fuente /dev/urandom.

Cifrado a nivel de Kernel

Tal y como podemos apreciar en la Figura 2, el cifrado a nivel de Kernel hace que el usuario crea que está escribiendo de forma normal a disco. Sin embargo, las llamadas de lectura/escritura a las zonas de disco protegidas pasan a través de módulos como TrueCrypt, Loop-AES, Cryptoloop…, los cuales usan los interfaces Device Mapper (/dev/mapper/) y Loop Devices (/dev/loop) para transformar esa información usando librerías de cifrado como las CryptoAPI, y haciendo que todas las aplicaciones a nivel de usuario no sepan si trabajan con una zona de disco normal o contra una cifrada.

Cryptoloop

Cryptoloop [1] fue uno de los primeros sistemas de cifrado soportado completamente a nivel de Kernel (desde la rama 2.4). Hoy día se están abriendo paso nuevas plataformas, sin embargo, sigue estando presente en el código principal del núcleo aún siendo considerado como "deprecated" (sin futuro desarrollo activo). Sin duda, esto se debe a la facilidad de uso y a lo extendido que está.

Pros:

- Sencillo de instalar y configurar.
- No requiere ninguna modificación de las herramientas en el espacio de usuario.
- Permite una amplia lista de algoritmos de cifrado: AES, Blowfish, Twofish, …
- Permite crear sistemas de ficheros con "journaling" (ext3, resiserfs, …) tanto en loop devices (particiones, discos, dispositivos USB) como en ficheros dentro del sistema.
- Ideal para iniciar el aprendizaje de las plataformas de cifrado de sistemas de ficheros a nivel de Kernel en Linux.

Contras:

- Problemas con los ficheros que presentan marcas de agua. Éstas son patrones de información que se repiten a intervalos regulares, de forma que si se utiliza un vector de inicialización a la hora de cifrar los bloques del disco (generalmente de 512 bytes) y éste se repite cada x bloques, es posible detectar la existencia de un fichero con esta "meta información", haciendo que se pueda identificar la existencia del mismo dentro de la zona cifrada sin necesidad de averiguar la clave usada. Éste se usa en contenidos de información específicos para poder ser detectados. Imaginemos un DVD que se le da a la prensa antes de su comercialización. Para que veáis cómo funciona esta técnica os dejo un link a una página con el código fuente de una de estas aplicaciones, el código es muy sencillo y sirve para que nos demos cuenta de las debilidades que pueden tener estos sistemas de cifrado. Más información en [3].
- No es tan eficiente como otras implementaciones que también trabajan sobre Loop Devices, como Loop-AES. Esto es debido a que la implementación actual usa algoritmos de cifrado programados mediante librerías C , y actualmente se está trabajando con su implementación a nivel de ensamblador (lenguaje máquina), lo que hace que la optimización sea máxima para su ejecución por parte de los procesadores de la familia Intel i386.
- El cifrado de la información se realiza con la clave sin añadir nada de información aleatoria, lo que se denomina "añadir sal a la clave" (o "salt password"). Este procedimiento permite la generación de claves de cifrado distintas para cada sector, haciendo que un ataque mediante fuerza bruta para

Listado 1: Módulos de CryptoAPI cargados en el Sistema

```
01 # cat /proc/cryto
02    name         : aes
03    driver       : aes-i586
04    module       : kernel
05    priority     : 200
06    type         : cipher
07    blocksize    : 16
08    min keysize  : 16
09    max keysize  : 32
10    ...
11
12    name         : blowfish
13       d r i v e r            :
   blowfish-generic
14    module       : kernel
15    priority     : 0
16    type         : cipher
17    blocksize    : 8
18    min keysize  : 4
19    max keysize  : 56
20    ...
```

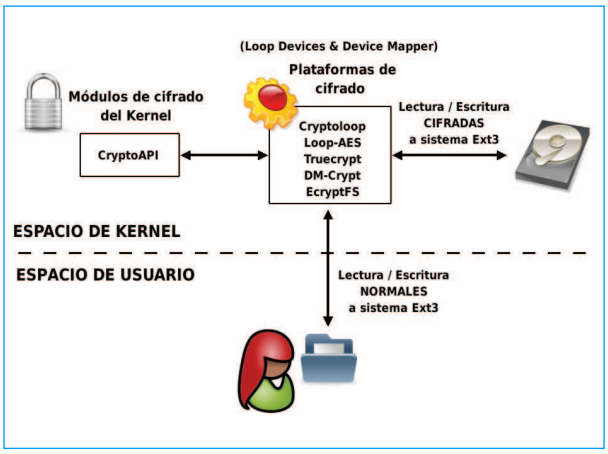

Figura 2: Funcionamiento del cifrado a nivel de Kernel.

averiguar la clave sea prácticamente imposible. En otras plataformas como Loop-AES se usan hasta 65 claves generadas mediante esta técnica con el uso de información aleatoria y de los algoritmos de cifrado simétricos usados por gpg.

Instalación: Como ya se ha comentado, los módulos de Crytoloop se encuentran soportados en la rama principal del Kernel, por ello su compilación y uso resultan tremendamente sencillos.

```
Device Drivers ->
Block devices -> Loopback
device support = YES
Device Drivers -> Block devices
-> Cryptoloop device support =
YES
General -> Cryptographic API =
YES
```

Se pueden instalar como módulos que se cargan bajo demanda o como elementos internos al kernel. La decisión variará dependiendo del uso que se le vaya a dar. En el proceso de pruebas podemos marcarlos como módulos <M>, y una vez hayamos decidido lo podemos volver a compilar, esta vez integrándolo en el kernel <*>, pues obtendremos un mejor rendimiento. En el ejemplo actual los voy a integrar directamente en el kernel, tal y como vemos en la Figura 3. Una vez hayamos arrancado el sistema con nuestro nuevo kernel, lo primero que vamos a hacer es comprobar que realmente podemos usar los módulos de cifrado elegidos, así como ver información interesante de ellos como el tamaño de clave min/max , tamaño de bloque , etc. (ver Listado 1).

En caso de haberlos compilado por módulos hay que cargarlos:

```
# modprobe cryptoloop
# modprobe
[módulo_cif_elegido:
aes, blowfish,
 twofish ...]
```

Si queremos que se carguen al iniciar el sistema:

```
# echo “cryptoloop”
 >>
/etc/modules
```

El sistema de cifrado mediante Loopback nos permite mapear tanto dispositivos físicos, como discos, o bien hacerlo directamente sobre los ficheros, lo que es especialmente interesante a la hora de facilitar su portabilidad. En el siguiente ejemplo vamos a crear un fichero de 100MB, llamado espacioCifrado, y vamos a utilizarlo como sistema de ficheros cifrado dentro del disco para almacenar todos los datos que queremos proteger.

Primero creamos el fichero con información aleatoria:

```
# dd if=/dev/urandom
of=/root/espacioCifrado bs=1M
count=100
```

Si hubiésemos elegido un disco, una partición o un dispositivo externo USB, el nombre del fichero “espacio_cifrado” hubiera sido sustituido por algo como hdb (disco), hdb1 (partición), sdb1 (dispositivo externo USB).

Ahora vamos a unir el fichero creado con el interfaz Loopback de forma que podamos leer los bloques del fichero a través del mismo. Hemos elegido el primer interfaz, pero podríamos elegir cualquiera de los 8 disponibles (loop0 .. loop7). Especificamos además qué módulo de los de cifrado que compilamos inicialmente vamos a usar: aes, blowfish, twofish, … Nos preguntará el password sólo una vez, así que tenemos que asegurarnos de que lo haremos bien (podemos escribir la contraseña en un fichero y copiarla y pegarla en consola mediante ctrl + insert de manera que nos aseguremos de que no existen errores tipográficos).

```
# losetup -e aes /dev/loop0
/root/espacioCifrado
```

Para crear el sistema de ficheros primero elegimos el tipo que queramos: ext3, ext2, vfat, reiserfs… Lo que vamos a realizar como primera operación cifrada dentro del fichero elegido es la creación de las tablas de inodos (enlaces a bloques dentro del sistema de ficheros). Importante: Este paso sólo lo haremos la primera vez, el resto de las veces que queramos acceder a la información ejecutaremos el paso anterior y posterior a éste.

```
# mkfs.ext3 /dev/loop0
```

Ahora ya podemos montar y trabajar con el sistema de ficheros. Para ello creamos una carpeta /mnt/cifrado.

```
# mount -t ext3 /dev/loop0
/mnt/cifrado
```

A partir de este momento todas las operaciones de lectura/escritura a ese directorio pasarán por el módulo Loopback, el cual realizará el proceso de cifrado y descifrado. A la hora de desmontar el sistema de cifrado y de liberar el interfaz Loopback haremos:

```
# umount /mnt/cifrado
# losetup -d /dev/loop0
```

Hay que tener en cuenta que los pasos de enlazar el sistema con Loopback y de montar el sistema pueden ser realizados directamente por el comando mount + opciones, el cual nos pedirá la clave que usamos a la hora de crearlo la primera vez. Incluso podemos incluirlo directamente en el fichero de montaje automático /etc/fstab, de forma que al iniciar la máquina nos pida el password y monte el sistema

```
# mount -o loop=/dev/loop0
/root/espacioCifrado
/mnt/cifrado
/etc/fstab :
/root/espacioCifrado
/mnt/cifrado ext3 user,
loop=/dev/loop0, encryption=aes
```

Loop-AES

Esta plataforma de cifrado también usa el modelo de Loop Devices, mediante un interfaz del tipo Loopback; pero en este caso es la misma aplicación Loop-AES [4] la que trae su propio módulo para el kernel y su “set” de aplicaciones para el usuario (incluyendo mount, losetup…), donde vienen solucionados muchos de los problemas que hemos comentado con Cryptoloop.

Pros:

- Es funcional a partir de la versión 2.0 del Ker-

nel, haciendo que su compatibilidad sea total con cualquier sistema que tengamos funcionando con Linux.

- Utiliza hasta 65 llaves distintas, las cuales las crea añadiendo información aleatoria ("adding salt procedure"). Cada llave cifra un sector determinado de unos 512 bytes que se va repitiendo continuamente. Este modelo de llaves múltiple se denomina: multi-key-v3. Esta forma de generación y uso de las claves hace que el sistema sea inmune a los ataques de diccionario.

Contras:

- No permite el uso de sistemas de ficheros con "journaling" en Loop Files creados en el sistema (sistemas de ficheros creados dentro de un fichero de tamaño elegido). Sí que se pueden usar sobre discos, particiones y dispositivos externos, pues se respeta el orden de escritura requerido (siempre que hayamos desactivado la escritura mediante búfer cache).
- Si se quiere aprovechar el grado de máxima seguridad hay que llevar un fichero con las 65 llaves siempre encima (ej: en un pen drive USB).
- En caso de suspensión o hibernación del equipo, las claves que residen en memoria RAM son copiadas a disco (para poder obtener un estado idéntico tras la reanudación), y por ello podrían ser vulnerables a un ataque. Se recomienda no usar estos mecanismos por seguridad.

Lo primero es recompilar el Kernel desactivado del módulo loopback, pues Loop-AES trae su propio módulo.

```
Device Drivers ->➥
Block devices -> Loopback➥
device support = NO
Device Drivers ->➥
Block devices -> Cryptoloop➥
device support = NO
Loadable module support ->➥
Enable loadable module support➥
= YES
```

Necesitamos tener en el sistema los ficheros con el código fuente de las librerías dietlibc (normalmente los encontraréis en vuestra distribución bajo el nombre de dietlib-dev). Vamos al directorio donde hemos descomprimido Loop-AES:

```
# make clean
# make KEYSCRUB=y
```

Con la opción keyscrub activada al compilar lo que conseguimos es que al almacenar las claves en RAM no se guarden tal cual, sino que se almacenen de forma desordenada para que no se pudieran extraer de ahí. Esta opción (aunque no está propiamente documentada) podría evitar los problemas previos que se han comentado acerca de la suspensión de los equipos. Al ejecutar make instalará en el sistema el módulo (en caso de compilación exitosa), por lo que si no queremos que se almacene con el resto de módulos que se crean al compilar el Kernel, podemos indicar expresamente una ruta :

```
# make INSTALL_MOD_PATH=➥
/.../.../directorio_elegido
```

Ahora probamos que el módulo funciona correctamente:

```
# modprobe loop
# ls /dev/loop*
```

Si todo está bien, habremos obtenido /dev/loop0, /dev/loop1, …, /dev/loop7. Ahora tenemos que instalar las herramientas a nivel de usuario para trabajar con Loop-AES. Para ello, vamos a aplicar un parche a las aplicaciones que se encuentran dentro del paquete util-linux que pertenece a la rama de aplicaciones del usuario dentro del kernel.

Compilación de las herramientas del usuario: Primero obtenemos el código fuente de la última release estable 2.12r de [5], le aplicamos el parche y compilamos e instalamos las herramientas modificadas así como las manuales para el "man" del sistema (ver Listado 2).

Si queremos que el módulo "loop" se cargue automáticamente cada vez que arrancamos la máquina hay que añadirlo:

```
# echo "loop" >>➥
/etc/modules
```

Tal vez, si estamos creando un entorno de máxima seguridad nos interese que gpg añada hasta 128 iteraciones recursivas en el cifrado del fichero de claves, lo que implica que cada operación sea 128 veces más lenta. Para ello parcheamos la aplicación gpg con código incluido junto con los fuentes. En nuestro caso no es necesario, pues nos interesa un sistema seguro pero que además sea eficiente. Si se está interesado, siempre se puede acudir al manual general de Loop-AES y buscar información sobre el parcheo de gpg para aumentar las iteraciones sobre cada clave.

```
# make tests
```

Si obtenemos ***Test results OK*** es que todo ha ido bien. En caso contrario habría que repasar los pasos anteriores.

Utilización: Como ejemplo vamos a realizar el cifrado de una partición con almacenamiento externo de las claves. Disco USB a cifrar : /dev/sda1, que montaremos sobre /usbCifrado; como almacenamiento externo de las claves daremos el lápiz USB mapeado en /dev/sdb1 montado sobre /usbClaves. Primero vamos a generar un fichero con las 65 claves que usaremos para cifrar la información de los sectores de la partición elegida y lo almacenaremos directamente en un fichero dentro del disco USB externo, el cual ha sido montado en el directorio /disco_USB:

```
# head -c 3000 /dev/urandom➥
| uuencode -m - | head -n 66 |➥
tail -n 65 | gpg –symmetric➥
–cipher-algo AES256 -a >➥
/usbClaves/claves.gpg
```

Lo que hemos hecho ha sido coger 3000 caracteres generados de forma aleatoria por la fuente /dev/urandom, transformarlos a caracteres ASCII mediante uuencode, acotar el número de líneas a 65 y pasar estas líneas por la aplicación gpg con la finalidad de aplicar el algoritmo de cifrado simétrico AES de 256 bytes. Podemos elegir cualquiera entre los siguientes: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH. Se nos preguntará qué password usar para este último proceso. Este será el password que nos pedirán las herramientas (parcheadas de Loop-AES) para acceder a nuestro sistema de ficheros. Una vez lo introduzcamos será pasado mediante una tubería a "gpg", que será quien descifre el fichero y devuelva las 65 claves que habíamos almacenado a las aplicaciones mediante el uso de la tubería que habíamos establecido previamente. Ahora que ya tenemos las claves que usaremos, y una vez protegidas, lo que hacemos es insertar el lápiz USB (en nuestro caso mapeado en /dev/sda1) y lo llenamos por completo (1GB) de información aleatoria, al igual que lo hacíamos con Cryptoloop.

```
# dd if=/dev/urandom➥
of=/dev/sda1 bs=1M count=1000
```

Ahora vamos a unir uno de los Loop Devices creados por el módulo del Kernel "loop" de Loop-AES con el disco USB: crearemos su sistema de ficheros (ext3), lo montaremos y obtendremos información para verificar que el proceso se ha realizado con éxito. Como algo-

ritmo de cifrado, yo recomiendo el uso de AES256. Nos pedirá la clave una sola vez, por lo que hay que tener especial cuidado con no introducirla mal.

```
# losetup -e AES256 -K
/usbClaves/claves.gpg
/dev/loop0 /dev/sda1
# mkfs.ext3 /dev/loop0
# mount /dev/loop0 /usbCifrado
# losetup -a
/dev/loop/0: [000c]:30508
(/dev/sda1) encryption=AES256
multi-key-v3
```

Efectivamente en el resultado vemos que hemos mapeado /dev/sda1 en el loopback /dev/loop0 usando como algoritmo de cifrado AES256 y el sistema multi-key-v3 que funciona mediante las 65 claves generadas. Para desmontarlo por completo:

```
# umount /usbCifrado
# losetup -d /dev/loop0
```

Este mismo ejemplo funcionaría igual usando una partición de un disco interno en vez del USB, por ejemplo, hdb3 en vez de sda1. En este último caso sería recomendable incluir una línea en /etc/fstab para realizar el montaje automático cada vez que arrancara el sistema:

```
/dev/hdb3 /discoInterno
ext3 defaults,noauto,
loop=/dev/loop0,
encryption=AES256,
gpgkey=/usbClaves/claves.gpg
0 0
```

DM-Crypt

Entre la sencillez e inseguridades de Cryptoloop, y la complejidad y eficiencia de Loop-AES, tenemos DM-Crypt [6], una plataforma apoyada por lo desarrolladores del Kernel y que usa sus herramientas de cifrado: CryptoAPI. En realidad, lo que la diferencia con Cryptoloop es que ésta utiliza el módulo "Loop Devices", mientras que DM-Crypt usa el módulo "Device Mapper" a la hora de realizar las operaciones de cifrado y descifrado de la información.

Pros:

- Permite el uso de LUKS (Linux Unified Key Setup), por lo que podemos realizar cambios de las claves sin tener que volver a cifrar la información del disco.
- Integrado dentro del Kernel mediante el uso de Device Mapper y CryptoAPI.

Contras:

- La creación de sistemas de ficheros cifrados sobre ficheros requiere el uso de una herramienta fuera del conjunto de las Crytpsetup, lo hace por medio de Losetup, por lo que hay que puentear Loop Devices (/dev/loop*) con los Device Mapper (/dev/mapper/*).

Instalación: Recompilamos el Kernel activando:

```
Device Drivers >
Multi-device support (RAID y
LVM) = YES
General -> Cryptographic API
(los que queramos) = YES
```

Ahora ya tenemos el módulo del Kernel funcionando. El siguiente paso es instalar el paquete con las librerías necesarias para que las herramientas del espacio de usuario puedan comunicarse con el Device Mapper. Podemos encontrar el paquete en nuestra distro con el nombre de dmsetup, device-mapper package / tools … O podemos compilarlo desde el código fuente. Para ello nos bajamos el código de *ftp://sources.redhat.com/pub/dm/* . Lo descomprimimos, lo compilamos siguiendo el proceso habitual (*./configure; make; make install*) y comprobamos que el módulo y que las librerías se han instalado correctamente:

```
# tar -xzvf
device-mapper.1.02.13.tgz
#dmsetup targets
striped          v1.0.2
linear           v1.0.1
error            v1.0.1
```

En este punto tiene que haberse creado un controlador del interfaz: /dev/mapper/control.

Ahora vamos a instalar las herramientas que usará el usuario para manejar Device Mapper a través de dmsetup. Se encuentran dentro del paquete Cryptsetup. Su compilación es costosa, ya que requiere de la existencia de las librerías libgcrypt y los ficheros de desarrollo de libdevmapper. En vez de trabajar con la rama normal de Cryptsetup vamos a usar una modificación que incluye LUKS (Linux Unified Key Setup), a la que denominaremos Cryptsetup-LUKS y que podemos encontrar en [7]. Básicamente, lo que LUKS permite es la utilización de un espacio en las cabeceras de los discos para guardar información utilizada para cifrar el resto del disco. Lo que hacemos con los passwords y los keyfiles

Listado 2: Parcheado, instalación y configuración Loop-AES

```
01 # patch -p1 < ../util-linux-2.12r.diff
02 # CFLAGS=-O2 ./configure
03 # make SUBDIRS="lib mount"
04 # cd mount
05 # install -m 4755 -o root mount umount /bin
06 # install -m 755 losetup swapon /sbin
07 # rm -f /sbin/swapoff && ( cd /sbin && ln -s swapon swapoff )
08 # rm -f /usr/share/man/man8/{mount,umount,losetup,swapon,swapoff}.8.gz
09 # install -m 644 swapon.8 swapoff.8 /usr/share/man/man8
10 # install -m 644 mount.8 umount.8 losetup.8 /usr/share/man/man8
11 # rm -f /usr/share/man/man5/fstab.5.gz
12 # install -m 644 fstab.5 /usr/share/man/man5
13 # mandb
```

que damos a Cryptsetup-LUKS es usarlos para generar una "master-key" que se guardará mediante cifrado simétrico en ese espacio de cabecera. Cada vez que accedamos con éxito al disco descifraremos esa llave y la usaremos para las operaciones de lectura y escritura cifradas en el disco.

Funcionamiento: Vamos a crear una partición cifrada en sda1:

```
# cryptsetup –verbose
–cipher "aes-cbc-essiv:sha256"
–key-size 256
–verify-passphrase luksFormat
/dev/sda1
```

No podemos pasar keyfile a la hora de crear, pero podemos añadirlo así cuando cambiemos el password. Ahora mapeamos el nuevo sistema con el Device Mapper:

```
# cryptsetup luksOpen
/dev/sda1 cifradoUSB
```

```
# ll /dev/mapper/
total 0
15941 brw-rw---  1 root disk➥
254,  0 2006-12-01 05:23➥
cifradoUSB
2950  crw-rw---  1 root root➥
10, 63 2006-11-30 19:47➥
control
# mkfs.ext3➥
/dev/mapper/cifradoUSB
# mount /dev/mapper/cifradoUSB➥
/cifrado
```

Para desmontarlo:

```
# umount /cifrado
# cryptsetup luksClose➥
/dev/mapper/cifradoUSB
```

Truecrypt

La plataforma Truecrypt [8] es una de las preferidas actualmente debido principalmente a la capacidad de creación de volúmenes secretos y a que los sistemas de ficheros cifrados pueden verse bajo GNU/Linux y bajo sistemas Windows.

Pros:

- Multiplataforma GNU/Linux, Windows
- Una de las más activas en el desarrollo
- Una de las documentaciones más detalladas sobre la metodología de cifrado usada
- Posibilidad de iteraciones de cifrado en cascada mediante múltiples algoritmos
- Permite gran multitud de sistemas de cifrado simétrico y de funciones hash, haciendo que lo podamos personalizar al máximo
- Permite el cambio de claves y keyfiles sin necesidad de tener que volver a cifrar la información.
- No conoce qué algoritmo de cifrado usar, por lo que usa prueba/error con todos los soportados una vez introducimos la clave, añadiendo complejidad ante los ataques
- Permite el uso de tantos passwords y keyfiles como queramos, posibilitando la creación de sistemas de cifrado con claves distribuidas (reparto de poderes)

Contras:

- Requiere de un Kernel igual o mayor de la rama 2.6.5
- La permisividad de cambio de claves por otras más complejas no varía la dificultad del cifrado

Instalación: Lo primero es descargar el código fuente. En la página se pueden bajar paquetes para distintas distribuciones como Ubuntu, Fedora y OpenSuse. Sin embargo, en este artículo vamos a hacerlo desde el código fuente, de forma que todo el mundo sepa cómo hacerlo independientemente de la distro que use. Para compilar TrueCrypt necesitamos tener el código fuente del Kernel que estemos usando en nuestra máquina, concretamente en el directorio /usr/src/linux-xxx . Donde xxx es la versión que tenemos, para conocerla ejecutamos "uname -r".

Hay que tener en cuenta que a partir del kernel 2.6.18, y para evitarnos los problemas de compilación, hay que realizar un pequeño cambio en el fichero del kernel: drivers/md/dm.h, concretamente, a partir de la línea #define DM_NAME "device-mapper" hay que añadir:

```
#define MSG_PREFIX➥
"truecrypt: "
#define DM_MSG_PREFIX➥
MSG_PREFIX
```

Ahora vamos a la carpeta Linux dentro del código fuente de TrueCrypt y ejecutamos:

```
# ./build.sh
```

Una vez hayamos realizado el proceso con éxito, pasamos a instalar el módulo dentro de las carpetas del kernel y los ficheros binarios que el usuario utilizará para manejarlas en las correspondientes de ejecución del sistema, así como la ayuda dentro del "man".

```
# install.sh
Kernel/truecrypt.ko
Cli/truecrypt
Cli/Man/truecrypt.1
```

Al igual que en las plataformas anteriores, si queremos que se cargue el módulo automáticamente al arrancar el sistema hacemos:

```
# echo "truecrypt" >>➥
/etc/modules
```

Respecto a la configuración del Kernel, hay que tener activados los módulos de Device Mapper y Loop Devices, así como instaladas en el sistema sus respectivas herramientas (dmsetup y losetup).

Utilización: Si no está activo cargamos el módulo y comprobamos que se ha cargado con éxito:

```
#modprobe truecrypt
#lsmod
Module  Size Used by
truecrypt  160260  0
```

Como ejemplo vamos a crear un sistema de ficheros dentro de un fichero. Esta opción es especialmente interesante por la posibilidad de llevarnos todo el sistema en un DVD, o en un disco duro externo simplemente como si fuera un fichero más. Si queréis aplicarlo sobre una partición o un disco, una vez más, sólo tenéis que cambiar el nombre del fichero por el del dispositivo: /dev/hda3, /dev/hdb, /dev/sda2, … Primero creamos la partición cifrada:

```
# truecrypt –type normal➥
–random-source /dev/urandom➥
–size 5G –filesystem FAT➥
–encryption Serpent-Twofish-AES➥
–hash Whirlpool –keyfile➥
ficheroClaves  ... -c➥
sistemaCifrado
```

Veamos una a una las opciones elegidas:

- *–type normal*: estamos creando un sistema de ficheros cifrado que estará ubicado de forma normal (no escondido) dentro del espacio de bloques dedicado dentro del disco
- *–random-source*: especifica la fuente de donde extraerá la información, para rellenarlo en caso de que queramos sobreescribir esa zona del disco, algo que es totalmente recomendable.
- *–size 5G*: el tamaño de la partición, podemos especificarlo en KB ,MB y GB mediante K,M.G. Generar gigas de información aleatoria cuesta, así que tened paciencia.
- *–filesystem FAT*: hemos elegido esa partición porque el sistema de ficheros que estamos creando no es demasiado grande y porque nos interesa que sea portable a otras plataformas. En principio podemos elegir cualquier sistema de ficheros que soporte nuestro sistema operativo (ojo con el tema de la portabilidad).
- *–encryption Serpent-Twofish-AES*: algoritmo de cifrado simétrico. En nuestro ejemplo hemos usado el más seguro, pues mezcla 3 de ellos (serpent, twofish y AES) en lo que se denomina cifrado en cascada.
- *–hash Whirlpool*: como algoritmo hash hemos elegido Whirlpool, que a diferencia de SHA-1 no se le ha encontrado ninguna debilidad a día de hoy en tema de colisiones.
- *–keyfile ficheroElegido*: podemos añadir cualquier fichero como fichero de claves (independientemente del tipo que sean). Por ejemplo, podemos usar un mp3 con nuestra canción favorita, una foto con los amigos, … eso sí, solo se emplea 1MB de cada fichero, pero podemos añadir todos los que queramos.

- *-c sistemaCifrado*: es el nombre que daremos al fichero a crear

La única pregunta que hemos dejado sin responder ha sido el password que se añadirá a los ficheros de claves, el cual conviene dar de forma interactiva, ya que se desactiva la salida al terminal de la contraseña (*echo off*). Ahora creamos una partición de 1GB escondida dentro de ella y cambiando los algoritmos de cifrado (de nuevo nos preguntará por el password):

```
# truecrypt –type hidden➥
–random-source /dev/urandom➥
–size 1G –filesystem FAT➥
–encryption AES-Twofish-Serpent➥
–hash Whirlpool –keyfileßs
ficheroClaves ... -c➥
sistemaCifrado
```

Seguidamente podemos montar la partición normal o la secreta dependiendo de qué password y qué ficheros de claves le pasemos como parámetros, en la Figura 4 podemos ver un diagrama que lo explica.

```
# truecrypt sistemaCifrado➥
/mnt/cifrado
```

Hay que tener en cuenta que si montamos de esta manera la partición cifrada en el espacio normal no respetará la zona de la secreta (porque no sabe de su existencia), y podría sobreescribirla. Si queremos montarla de forma que además se proteja la zona secreta (para trabajar con ella en condiciones normales), podemos hacerlo así:

```
# truecrypt -P➥
sistemaCifrado /mnt/cifrado
```

Lógicamente nos pedirá los passwords y ficheros de claves de ambas zonas para poder verificar su existencia y obtener la posición en los bloques del disco de los mismos. Veamos si todo es correcto:

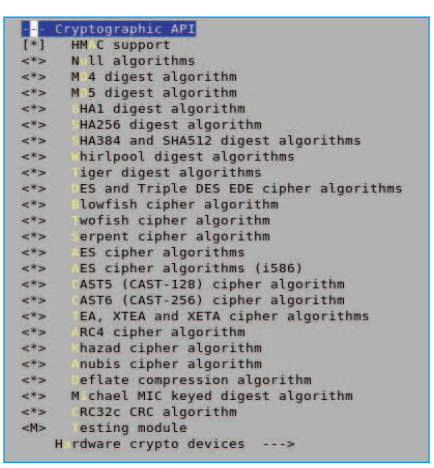

Figura 3: Módulos de cifrado de CrytoAPI incluidos en el Kernel.

```
# truecrypt -vl
/dev/mapper/truecrypt0:
 Volume: /mnt/cifrado
 Type: Hidden
 Size: 1073741824 bytes
 Encryption algorithm:➥
 AES-Twofish-Serpent
 Mode of operation: LRW
 Read-only: No
 Hidden volume protected: No
```

Para desmontar el volumen:

```
# truecrypt -d /mnt/cifrado
```

Cifrado a nivel de espacio de Usuario

Ahora vamos a ver cómo funcionan las plataformas disponibles para su uso directamente desde el espacio de usuario. Éstas trabajan directamente con los ficheros, y no con particiones o discos.

Fuse

Sin duda alguna, el uso del módulo FUSE (Filesystem in Userspace) ha permitido que las aplicaciones del usuario se comuniquen con otros sistemas de comunicación más complejos a través de simples llamadas al sistema de ficheros. Un ejemplo de esto lo vemos en el uso de las cuentas de Gmail como método de almacenamiento en disco [11] , puesto que se pueden montar como un sistema de ficheros más y FUSE se encarga de transformar nuestras lecturas/escrituras en mails que se leen y mandan a la cuenta. Hay que activarlo y recompilar el Kernel:

```
File systems  -->
        <M> Filesystem in
 Userspace➥
        support
```

Una vez lo tengamos activo se habrá creado el interfaz /dev/fuse. Otros paquetes que necesitamos son: fuse-utils, libfuse2, libfuse-dev, estos últimos para compilar mediante el código fuente las aplicaciones que veremos a continuación.

CryptoFS

Una de las cosas más destacables de la plataforma CryptoFS [9] es que nunca hay fallos al intentar acceder a la zona cifrada. Lo que sucede es que al introducir una clave errónea los ficheros se descifran, dando como resultado algo que no tiene sentido. Esta táctica es muy buena para prevenir ataques, sobre todo en el caso de que se intente acceder a una información de la que se desconoce por completo su contenido.

Pros:

- Muy sencillo de compilar y usar.
- Si tenemos instalado GNU-PG usará su aplicación PIN para pedirnos el password, lo que incrementa la finalidad

Utiliza los algoritmos de cifrado de las libgcrypt, las cuales están realmente extendidas.

Contras:

- Los nombres de los ficheros en el espacio cifrado son de distinto tamaño según lo sea su nombre real. Lo que puede llevar a un atacante a saber qué espacio del disco cifrado corresponde a cada fichero.

Necesitaremos las librerías libgcrypt (ej: libgcrypt11-dev), así como las Glib (gtk > 2.6), y por supuesto los de desarrollo de FUSE, así como las librerías.

```
# ./configure ; make ; make➥
install
```

Ahora generamos un fichero de configuración en el directorio donde estarán nuestros ficheros cifrados, por ejemplo: /cifrado . El fichero será : /cifrado/.cryptofs. Ejemplo de fichero de configuración:

```
[CryptoFS]
cipher=AES256   #algoritmo de➥
cifrado simétrico
md=MD5          #algoritmo de
 hash➥
usado para crear la clave de➥
cifrado
blocksize=2048  #tamaño de➥
bloque
salts=256       #número de
 llaves➥
usadas
```

Lo montamos:

```
# cryptofs –root=/cifrado➥
/mnt/cifrado
```

Ya podemos mover la información a este directorio. Vista del contenido a través del filtro de CryptoFS manejado por FUSE:

```
# ll /mnt/cifrado
fichero1.txt
fich2.txt
```

Vista directa de los ficheros almacenados cifrados en disco:

```
# ll /cifrado
xaaSAS6SaA1
S1lk8e32d
```

Para desmontarlo:

```
# umount /mnt/cifrado
```

ENCFS

ECFS [10] trabaja a través de las librerías de cifrado de OpenSSL.

Pros:

- Permite elegir si queremos cifrar los nombres de cada fichero o sólo los datos.
- Permite que la longitud del nombre de los ficheros cifrada sea en todos la misma (múltiplo del tamaño del bloque). De forma que no puedan saber viendo los ficheros cifrados si tienen nombres cortos o largos.
- Permite generar los valores de los vectores de inicialización (salt) mediante la ruta completa del fichero. De esta forma el nombre del fichero "claves.txt" no se cifrará igual en /mnt/cifrado0 que en /mnt/cifrado1
- Permite que varios usuarios oculten sus propios ficheros dentro de un mismo sistema creado mediante EncFS . Dependiendo del password que demos, mostrará los ficheros cuyo descifrado haya sido exitoso con esa contraseña.

Contras:

- Genera metadatos en un fichero llamado .encfs5 que se encuentra en el sistema raíz de la carpeta de cifrado y que contiene información necesaria para el descifrado de los ficheros. Lo que lo convierte en un keyfile que se ha generado sin nuestro permiso.
- No existe la posibilidad de cambio de password en el sistema. Hay que mover los ficheros, crear uno nuevo, y cifrarlos de nuevo.
- De la misma manera no existe la posibilidad de cambiar otras propiedades del sistema de ficheros. Cualquier cambio exige creación de un nuevo sistema y nuevo cifrado del contenido.

Compilación: Para compilarlo nosotros mismos necesitamos librlog-dev, libssl-dev. En algunas distribuciones como Debian existe un paquete ya precompilado.

```
# tar -xzvf encfs-1.3.1-1.tgz
# ./configure
; make ; make➥
install
```

Ahora vamos a especificar el raíz de nuestro sistema de ficheros. Será la carpeta que contenga toda la información cifrada. Básicamente es una carpeta normal creada en el sistema, que va a contener una serie de ficheros con la característica especial de que su nombre y contenido reales están cifrados. Elegimos /cifrado, a continuación especificamos un directorio donde montaremos la carpeta raíz y sobre el cual FUSE nos hará creer que se trata de un directorio normal: /mnt/cifrado. La primera vez que montemos nuestro sistema, es cuando hemos de elegir las opciones específicas.

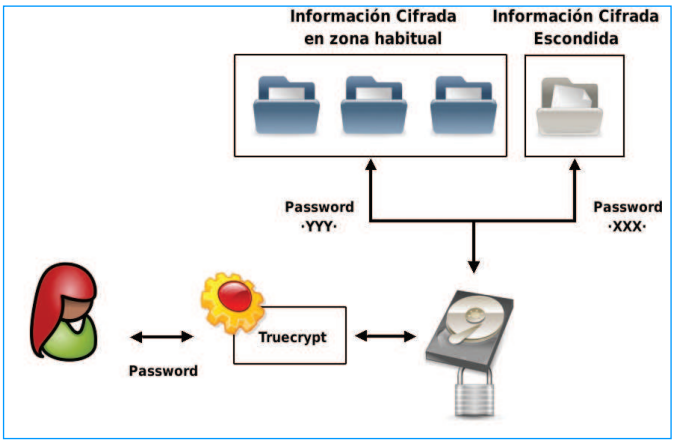

Figura 4: Funcionamiento de la zona secreta de TrueCrypt.

```
# encfs /cifrado➥
/mnt/cifrado –idle 10 –anykey➥
-extpass=➥
/usr/bin/ssh-askpass-fullscreen
```

Hay que tener cuidado, porque como ya se ha comentado en los contras, una vez especificadas las opciones no se pueden cambiar.

- *–idle*: especifica el tiempo que el sistema estará montado en inactividad. Esta opción es interesante para que, una vez ya no la estamos usando, se desmonte automáticamente, por ejemplo:10 min
- *–anykey*: para permitir múltiples usuarios, cada uno cifrando sus ficheros con su propia clave sobre el mismo sistema.
- *–extpass*: si queremos que sea otra aplicación más segura la que obtenga el password de nosotros y luego se lo pase a EncFS (ej: ssh-askpass).

El uso del modo experto permite configurar parámetros como queráis. Os nombro las opciones, y entre paréntesis la que recomiendo: algoritmo de cifrado (AES 256), tamaño del bloque (512), tipo de cifrado sobre los nombres de los ficheros (Block), dependencia de nombre de la ruta completa (Sí), vectores de inicialización para cada fichero (Sí), cifrado de datos dependiente del path completo (No). Para el resto de las veces lo podemos montar simplemente con:

```
# encfs /cifrado➥
/mnt/cifrado –stdinpass <➥
/mnt/usb/key
```

La opción -stdinpass hace que no nos pregunte el password, sino que lo tome directamente de la entrada estándar.

Ahora ya podemos trabajar con los ficheros. Vamos a listarlos para que veamos cómo esconde sus nombres de los ficheros y su longitud (ver Listado 3).

Para desmontarlo:

```
# umount /mnt/cifrado
```

Casos complejos de uso

Cifrado con claves distribuidas en USBs

Tal y como podemos ver en la Figura 6, la finalidad es asegurar la confidencialidad de la información, de forma que sólo se pueda acceder a ella mediante el uso de un conjunto de claves distribuidas entre varias personas. Concretamente, hemos elegido para ello la distribución de claves complejas en dispositivos lápices USB, los cuales son baratos, de uso extendido y de tamaño reducido. Se recomienda comprar varios de ellos (1 por persona) de poca capacidad, pues lo que vamos a almacenar ocupa unos pocos KB y es mejor dedicarlo exclusivamente para el uso que aquí se describe. Damos por hecho que se han seguido la guía de compilación y uso de TrueCrypt descritos previamente y que está funcionando el sistema de manejo de dispositivos UDEV, pues viene incluido en toda la rama 2.6 del Kernel.

Primero recogemos información de cada dispositivo USB que vamos a usar para almacenar los ficheros de claves (desde ahora denominado "llaves") . Para ello vamos a escoger parámetros propios del hardware a analizar, como el número de serie y el número identificativo del producto. Supongamos que el USB está mapeado en /dev/sda

```
# udevinfo -a -p➥
/sys/block/sda | grep -i serial
```

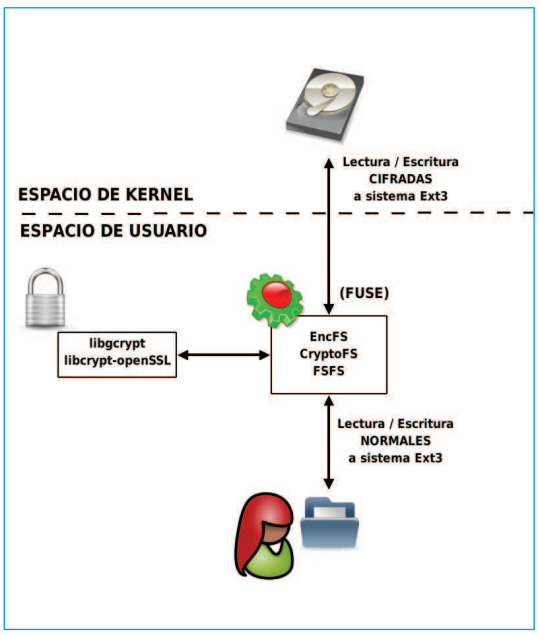

Figura 5: Funcionamiento del cifrado a nivel de Usuario.

```
SYSFS{serial}=="212304887990"
# udevinfo -a -p /sys/block/sda
| grep -i idproduct
SYSFS{idProduct}=="0056"
```

Con estos datos vamos a generar una regla en udev, de forma que cada vez que detecte este dispositivo lo mapee con un nombre concreto. Por ejemplo: Imaginemos que el USB pertenece a uno de los usuarios: "María Luisa" . Entonces crearemos un fichero llamado */etc/udev/rules.d/10-cifrado.rules* y escribiremos:

```
BUS=="usb", SYSFS{serial}==
"212304887990",
SYSFS{idProduct}=="0056",
KERNEL=="sd?1", NAME="%k",
SYMLINK="mariaLuisa"
```

Reiniciamos el demonio udev:

```
/etc/init.d/udev restart
```

Ahora, si volvemos a insertar el USB vemos como se crea */dev/mariaLuisa*, por lo que ya no nos importa si el sistema lo mapea en /dev/sda1, /dev/sdb1, … Una vez hayamos hecho todo con todos los usuarios, creamos un script para montar todos los USB mapeados en los dispositivos especiales en una carpeta elegida.

```
mount /dev/mariaLuisa
/llaves/mariaLuisa
mount /dev/ramon
 /llaves/ramon
mount /dev/luisma
/llaves/luisma
```

Al montar las llaves USB a partir de los dispositivos especiales, nos aseguramos de que el script siempre usará los dispositivos correctos, aunque tengamos muchos más conectados a la máquina. El siguiente paso es almacenar un keyfile dentro de cada USB. Esto ha de hacerlo cada usuario de forma privada y confidencial. El tamaño a tener en cuenta como máximo será de 1MB, y puede ser cualquier tipo de fichero, desde uno creado mediante números aleatorios, hasta otro fichero que contenga la canción/película (en cualquier formato) elegida por el usuario. En el proceso de creación del volumen cifrado se exigirá que cada usuario introduzca además un password (que pueda recordar) que generará una contraseña única que será la concatenación de todos ellos. Este paso no es obligatorio, pero es la única opción para mantener al sistema protegido en caso de que alguien robara todos los lápices USB de los usuarios. En el ejemplo crearemos un fichero de 1GB que contendrá nuestro sistema de ficheros para claves distribuidas:

```
# truecrypt –type normal
–random-source /dev/urandom
–size 1GB –filesystem FAT
–encryption Serpent-Twofish-AES
–hash Whirlpool -k
/llaves/mariaLuisa/keyfile -k
/llaves/ramon/keyfile -k
/llaves/luisma/keyfile -c
sistemaCifrado
```

Nos preguntará por el password, que ha de ser escrito por los tres usuarios. El proceso ha de realizarse siempre en el mismo orden y sin que nadie mire al otro en el proceso de introducción del mismo. Como el script de introducción de passwords está en modo echo off, no se mostrarán los caracteres escritos en la pantalla. Por ejemplo: si tenemos los passwords p1, p2 y p3, cuando el sistema nos pida el password nos quedará algo como "p1p2p3".

Si todo va bien, ya tenemos creada la partición para el sistema de ficheros cifrado. Desde ahora, para montar y acceder al sistema habrá que hacer:

```
# truecrypt -k /llaves/
mariaLuisa/keyfile -k /llaves/
ramon/keyfile -k /llaves/
luisma/keyfile sistemaCifrado
/mnt/cifrado
```

En ese momento se nos pedirá el password (compuesto) para poder acceder al mismo. Una vez terminado el proceso de desmontan los dispositivos y el sistema de mapeo Truecrypt.

```
umount /llaves/mariaLuisa
umount /llaves/ramon
umount /llaves/luisma
truecrypt -d
```

Como comentario extra, decir que podríamos crear una zona escondida dentro de la zona creada tal y como se especificaba en la primera parte del artículo sobre el uso de TrueCrypt.

Asegurando el sistema

Tal y como se ha expuesto en la introducción de este artículo, hay ocasiones en las que la información puede "escaparse" sin cifrar del espacio RAM volátil y acabar escrita en el disco. Este es el caso de la memoria Swap, que es el espacio de disco que el Kernel usa cuando los recursos de memoria RAM se ven agotados para volcar la información existente en ella y dejar paso a información más reciente. El problema es que aunque usemos módulos de cifrado, la información que está en RAM se almacena sin cifrar, ya que el módulo correspondiente sólo actúa con las operaciones a disco.

Solución: cifrar la partición dedicada a swap con claves aleatorias (pues es zona de paso), de forma que todo lo que se vuelque a disco esté cifrado. Supongamos que nuestra partición de swap es hda2, primero desactivamos la partición que actualmente está siendo usada com "swapoff -a". Y añadimos en /etc/fstab :

```
/dev/hda2 none swap sw,
loop=/dev/loop3,encryption=
AES256    0    0
```

Listado 3: Listado de ficheros cifrado y sin cifrar

```
01 # ls /cifrado/.*
02 .encfs5
03 fJDoXw38eS-mAoZarDF6mxFn
04 H1FLupPiA7XXGYM4nKOaMzhA
05 imJtA7eT,JtTBFGku5Dm4DSS
06 # ls /mnt/cifrado/.*
07 nombreCorto
08 nombreDeTamañoMedio
09 nombreDeTamañoMuyMuyLargo
```

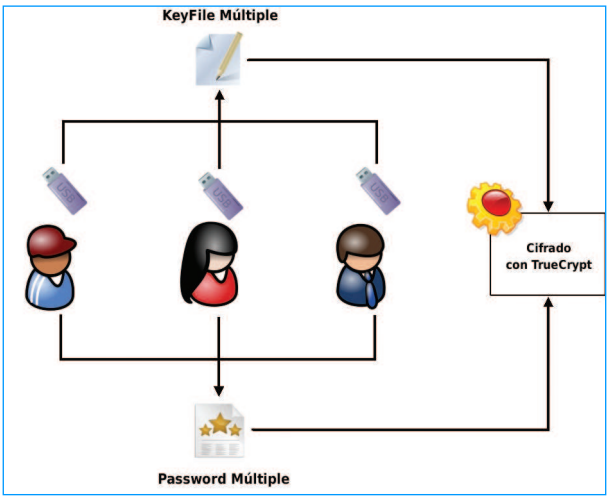

Figura 6: Cifrado con keyfiles y passwords distribuidos.

Mediante el comando "swapon -a" podemos ver cómo la deja funcionando sin la necesidad de proporcionar ninguna clave.

```
# swapon -a
Configurando espacio de➥
intercambio versión 1, tamaño =➥
501731 kB
no label, UUID=5f60200f-a850-➥
4511-9155-af9c5460ae8
```

Aunque no se especifique, se generan claves aleatorias para cifrar la información allá contenida, pues la finalidad es que se almacene temporalmente y que al reiniciar el sistema se vuelque al espacio de disco correspondiente y se generen nuevas para la información venidera.

Es posible que la próxima vez al arrancar tengamos problemas, pues normalmente la partición de swap se carga antes que el resto de particiones, y modprobe intentaría escribir en /var/log/ksymoops/ antes de que existiera el sistema raíz y daría errores de escritura.

Ahora vamos a cifrar la carpeta temporal del sistema /tmp, ya que es uno de los sitios donde, en ocasiones, el sistema deja restos de ficheros que se han abierto. Supongamos que vamos a usar la partición /dev/hda3 donde previamente hemos creado un sistema de ficheros ext3 para almacenar los ficheros temporales.

```
/dev/hda3 /tmp ext3➥
defaults, loop=/dev/loop4,➥
encryption=AES256,➥
phash=random/1777   0   0
```

Los permisos 1777 harán que el sistema de ficheros cifrado creado sobre /tmp permita el acceso de lectura, escritura y búsqueda. Además crea la carpeta /tmp con el bit pegajoso activado (sticky bit), este permite que sólo los propietarios de los ficheros puedan renombrarlos o borrarlos de la carpeta. Recordemos que cada vez que se recompile el kernel hay que generar de nuevo el módulo loop de Loop-AES, pues es dependiente de la versión del núcleo. Una vez hemos asegurado el entorno de trabajo, tenemos que centrarnos en el cifrado en sí de los datos del usuario. Para ello podemos utilizar cualquiera de las herramientas descritas en el artículo anterior como Cryptoloop, Loos-AES, TrueCrypt, EncFs, DM-Crypt, …

Otros sistemas de ficheros: Pasado y futuro

Pasado:

- CFS [12] (en funcionamiento desde 1992) usa el demonio NFS para realizar escritura cifrada en local. Trabaja a nivel de usuario.
- a [13] permite la creación de capas de cifrado independientes unas de otras, de forma que cada capa dé acceso a determinados ficheros y donde cada capa esté con su contraseña. Es una idea similar a la de múltiples passwords de EncFS. Trabaja a nivel de usuario.
- FreeOfte [14] permite acceder a volúmenes cifrados como dm-crypt, cryptoloop desde Windows. Trabaja a nivel de Kernel.

Futuro:

- Nivel de Kernel: EcryptFS [15] Reciente aparición en el Kernel (2.6.19) y que aún se encuentra en modo experimental. Trabaja directamente con los ficheros, no hay limitación del espacio a usar ni hay que definirlo al inicio. Como contra, destacar que si se cambia la clave exige volver a cifrar la información de todo el fichero. Por el momento está en estado experimental, si queremos compilarlo mediante el código fuente incluido dentro del kernel hay que activar ciertas opciones del modo experimental y del manejo de las llaves en el sistema.

```
Code maturity level options➥
  -->
  [*] Prompt for development➥
  and/or incomplete code/drivers
Security options  -->
  [*] Enable access key➥
  retention support
```

Ahora ya podemos seleccionar el módulo del nuevo sistema de ficheros:

```
File systems  -->
 Miscellaneous filesystems -->
  <M> eCrypt filesystem layer➥
  support (EXPERIMENTAL)
```

- Nivel de Usuario: FSFS [16] Al igual que CFS, usa el demonio NFS para crear dos canales de comunicación entre las diferentes máquinas de forma que la información y comandos de acceso a los datos siempre se transmitan cifrados. Lo primero, decir que la información ya de por si se almacena cifrada en el disco, pues todas las escrituras que se realizan son manejadas a través del módulo FUSE que cifra y descifra la información que pasa. Por ello podemos transmitir la información tal cual a diferencia de la implementación de NFS sobre Ipsec, que tiene que realizar la tarea de cifrado/descifrado de la información todo el tiempo. Sin embargo, las peticiones de información a nivel del sistema de ficheros que se realizan también han de ser cifradas (recordemos el manejo de ficheros a través de la red), por ejemplo "Leer 500 Bytes del i-nodo xxx con un offset de yyy". Para ello se crea un canal seguro mediante una conexión TLS v1 usando AES de 256 Bits. ■

RECURSOS

[1] *http://apeiron.laotracara.com/eliminar-la-informacion-del-disco-duro-para-siempre*

[2] *http://tldp.org/HOWTO/Cryptoloop-HOWTO/*

[3] *http://www.securiteam.com/exploits/5UP0P1PFPM.html*

[4] *http://loop-aes.sourceforge.net/*

[5] *ftp://ftp.kernel.org/pub/linux/utils/util-linux/util-linux-2.12r.tar.gz*

[6] *http://www.saout.de/misc/dm-crypt/*

[7] *http://luks.endorphin.org/dm-crypt*

[8] *http://www.truecrypt.org*

[9] *http://reboot.animeirc.de/cryptofs/*

[10] *http://arg0.net/encfs*

[11] *http://richard.jones.name/google-hacks/gmail-filesystem/gmail-filesystem.html*

[12] *http://www.crypto.com/software/*

[13] *http://www.freenet.org.nz/python/phonebook/*

[14] *http://www.freeotfe.org/docs/index.htm*

[15] *http://ecryptfs.sourceforge.net*

[16] *http://fsfs.sf.net*