Dans un précédent article, nous avions fait le tour des manipulations plus ou moins classiques autour des périphériques /dev/loop. Depuis le noyau 2.6.4, le chiffrement des systèmes de fichiers se fait via le device mapper et dm-crypt. L'occasion rêvée de revoir notre copie et de faire plus ample connaissance avec le device mapper et dmsetup.

Le device mapper est une nouvelle infrastructure intégrée dans la série 2.6 du noyau Linux (2.6.4 et plus). Celle-ci permet de fournir une couche d'abstraction qui se place au-dessus d'un périphérique bloc et fournit elle-même un tel périphérique.
Il est ainsi possible d'écrire des mappers permettant toutes sortes d'opérations allant du mirroring à la concaténation. ~~dm-crypt~~ est un mapper permettant le chiffrement d'un périphérique en mode bloc de manière transparente en utilisant la nouvelle API du noyau 2.6.

Device mapper et loop

Le device mapper ne signe pas la mort des ~~/dev/loop~~ et autres ~~losetup~~, bien au contraire. La complémentarité est exemplaire.
Le gros avantage proposé par le device mapper est d'offrir une seule et même infrastructure permettant la modification, à la volée, des périphériques en mode bloc.
Dans ce contexte, ~~cryptoloop~~ et ~~loop-aes~~ n'ont plus aucune utilité et le périphérique loop reprend sa place " normale ".
Ceci est encore et toujours parfaitement viable :

```
% dd if=/dev/zero of=./image bs=512 count=20480
20480+0 enregistrements lus.
20480+0 enregistrements écrits.
10485760 bytes transferred in 0,095682 seconds

% sudo mkfs.ext3 image
mke2fs 1.27 (8-Mar-2002)
image1 is not a block special device.
Proceed anyway? (y,n) y
Filesystem label=
OS type: Linux
[...]
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting
  information: done

% mount -o loop -t ext3 -v image /mnt/loo
```

Mais, nous pouvons à présent remplacer cette dernière commande par ceci :

```
 % losetup /dev/loop0 image
% blockdev --getsize /dev/loop0
20480
% echo "0 20480 linear /dev/loop0 0" | dmsetup create image
% mount -t ext3 -v /dev/mapper/image /mnt/loo
```

Ceci peut sembler long. C'est effectivement le cas, puisque cet exemple ne montre pas toute la puissance du système. Cependant la structuration est clairement visible. On utilise ~~losetup~~ pour associer l'image au périphérique ~~/dev/loop0~~.
La commande ~~blockdev~~ nous permet d'obtenir la taille du périphérique en secteurs, car nous en avons besoin pour créer la table de configuration du device mapper.
Enfin, nous utilisons ~~dmsetup create~~ pour créer littéralement le périphérique en mode bloc ~~/dev/mapper/image~~.
La table de configuration utilise une syntaxe particulière. Chaque ligne concerne une " cible " et utilise la forme suivante : secteur_début nbr_secteurs type arguments.
Les informations concernant les secteurs sont " virtuelles " et sans rapport avec le ou les périphériques existant physiquement.
Ici, nous avons utilisé un type ~~linear~~ en spécifiant ensuite en guise de paramètres le périphérique bloc source et le secteur de départ.
Le mapper ne permet pas le montage du système de fichier, mais la manipulation transparente du ou des périphériques bloc source :

```
% umount /mnt/loo
% dmsetup remove image

% dd if=/dev/zero of=./image2 bs=512 count=20480
% losetup /dev/loop1 image2

% cat | dmsetup create fusion
0 20480 linear /dev/loop0 0
20480 20480 linear /dev/loop1 0
^D
% mke2fs /dev/mapper/fusion
% mount /dev/mapper/fusion /mnt/loo
% df
[...]
/dev/mapper/fusion 10827 13 10700 1% /mnt/loo
```

```
/dev/mapper/fusion 19827 13 18790  1% /mnt/loo
```

Après avoir annulé les manipulations précédentes (umount et dmsetup remove), nous générons une nouvelle image de 10Mo et la lions avec /dev/loop1. Nous utilisons enfin dmsetup pour créer un nouveau périphérique en mode bloc nommé fusion.
Notre table est ici constituée de deux lignes. La première définissant la zone débutant au secteur 0 et d'une taille de 20480 secteurs et la seconde débutant au secteur 20480 et de taille identique.
L'initialisation d'un système de fichiers ext2 sur /dev/mapper/fusion ne pose pas le moindre problème et après montage, df nous affiche bien une taille totale correspondant environ à la taille des deux images. Ce type d'agrégation de périphériques bloc est utilisé au niveau du device mapper par des projets comme LVM, par exemple.

Les cibles du device mapper

Nous venons de voir une utilisation de la cible linear, mais il en existe d'autres. Le module noyau dm-mod.ko fournit de base trois cibles :

- linear que nous venons d'utiliser ;
- striped qui permet de répartir les données sur plusieurs périphériques.

```
% cat | dmsetup create strippy
0 40960 striped 2 32 /dev/loop0 0 /dev/loop1 0
^D
% mke2fs /dev/mapper/strippy
% mount /dev/mapper/strippy /mnt/loo

% df
[...]
/dev/mapper/strippy  19827  13  18790 1% /mnt/loo
```

On répartit ici les données sur deux périphériques (striped 2) avec un chunk de 32 secteurs. Le périphérique virtuel fait le double de la taille d'une image.
Les 16 premiers Ko du périphérique virtuel sont ceux de loop0, les 16 suivants sont ceux de loop1, les suivants correspondent aux 16 prochains Ko de loop0, etc. Il s'agit également d'une agrégation, mais les données sont réparties.

- error est une cible de test permettant de provoquer des erreurs d'E/S.

D'autres modules sont disponibles (/usr/lib/[...]/kernel/drivers/md et offrent des cibles différentes. dm-mirror est un bel exemple permettant de faire, comme son nom l'indique du mirroring. dm-snapshot permettra, pour sa part, de faire des " instantanés " des périphériques. Mais celui qui nous intéresse ici, c'est dm-crypt de Christophe Saout.
dmsetup est l'utilitaire permettant de configurer les périphériques. Une commande passée en argument permettra de créer un périphérique (create), de les lister (ls), d'obtenir des informations (info, status) ou encore de les supprimer (remove).
La page de manuel de l'utilitaire contient la syntaxe et quelques exemples.

Note:
En cas d'erreur dans la syntaxe de la table, dmsetup crée tout de même l'entrée /dev/mapper . Deux solutions s'offrent alors à vous, supprimer le périphérique puis le recréer ou modifier la table à la volée avec dmsetup load après un dmsetup suspend.

Chiffrement, la mort de cryptoloop

Comme son nom l'indique, dm-crypt permet, à l'instar de cryptoloop ou loop-aes, de créer un périphérique bloc avec des données chiffrées. Le device mapper de chiffrement est intégré au noyau 2.6 et utilise l'API cryptographique de ce dernier.
Contrairement aux précédentes solutions, maintenant à considérer comme obsolètes, dm-crypt ne nécessite pas de patcher le noyau ou d'installer des sources de modules supplémentaires. Tout est déjà prêt à fonctionner.
En se tenant à dmsetup, vous pouvez déjà utiliser la cible de chiffrement. Nous verrons plus loin qu'une manière plus simple de procéder existe.
Avant toute chose, votre noyau doit pouvoir supporter les algorithmes de chiffrement standards. Pour savoir ce qu'il en est, listez simplement le contenu de /proc/crypto :

```
% cat /proc/crypto
name         : md5
module       : kernel
type         : digest
blocksize    : 64
digestsize   : 16
```

Très souvent, par défaut, peu de (voire aucun) algorithmes sont supportés. Chargez les modules adéquats pour combler cette lacune (/lib/modules/[...]/kernel/crypto).
AES est l'algorithme utilisé en standard, pas la cible dm-crypt :

```
% modprobe aes
% cat /proc/crypto
name         : md5
module       : kernel
```

```
type         : digest
blocksize    : 64
digestsize   : 16

name         : aes
module       : aes_i586
type         : cipher
blocksize    : 16
min keysize  : 16
max keysize  : 32
```

Notez que aes est un alias du module aes_i586. Les informations fournies par /proc/crypto sont très utiles à ce niveau.
La taille de la clef est un élément déterminant pour la configuration du device mapper. Notre premier exemple utilisera une clef de 32 octets.
Pour générer cette dernière avec une syntaxe utilisable dans les tables dmsetup, nous utiliserons dd sur /dev/random et hexdump pour formater la clef :

```
% dd if=/dev/random bs=32 count=1 | \
  hexdump -e '32/1 "%02x" "\n"'
1+0 enregistrements lus.
1+0 enregistrements écrits.
32 bytes transferred in 0,001003 seconds (31904 bytes/sec)
3d1c6769ab6a6cb66cd4068067e38b6adc7d8ae0dea12451ae106048250cf817
```

Nous récupérons tout simplement 32 octets sur /dev/random et générons une clef hexadécimale. Dès lors, nous pouvons utiliser dmsetup (nous avons nettoyé les manipulations précédentes) :

```
% cat | dmsetup create chiffre
0 20480 crypt aes-plain 3d1c6769ab6a6cb66cd4068067e38b6
adc7d8ae0dea12451ae106048250cf817 0 /dev/loop0 0
^D

% mke2fs /dev/mapper/chiffre
% mount /dev/mapper/chiffre /mnt/loo
% cp watchdog.pdf /mnt/loo
% umount /mnt/loo
% dmsetup remove chiffre

% cat | dmsetup create chiffre
0 20480 crypt aes-plain 3531fbcab12b239f79a4d5a44b677999
da158a66aa197060de3f1a182f991a28 0 /dev/loop0 0
^D

% mount -t ext2 /dev/mapper/chiffre /mnt/loo
mount: wrong fs type, bad option, bad superblock
on /dev/mapper/chiffre,

% dmsetup suspend chiffre
% dmsetup reload chiffre
0 20480 crypt aes-plain 3d1c6769ab6a6cb66cd4068067e38b6
adc7d8ae0dea12451ae106048250cf817 0 /dev/loop0 0
^D
% dmsetup resume chiffre

% mount /dev/mapper/chiffre /mnt/loo
% ls /mnt/loo
.  ..  lost+found  watchdog.pdf
```

Cette succession de commandes montre la création du périphérique chiffre à l'aide de la cible crypt utilisant un chiffrement aes-plain (sur une seule ligne en réalité).
Nous initialisons, montons et utilisons un système de fichiers ext2 puis déconfigurons l'ensemble. L'utilisation d'une autre clef montre clairement la protection.
Enfin, nous profitons de l'occasion pour modifier la table sans supprimer le périphérique et accédons à nouveau correctement au système de fichiers.
Nous faisons usage ici d'AES et d'une clef de 32 octets (256 bits). Mais nous pouvons tout aussi simplement utiliser BlowFish (modprobe blowfish) en générant une clef adéquate (dd if=/dev/random bs=56 count=1 | hexdump -e '56/1 "%02x" "\n"') et en spécifiant crypt blowfish dans la table.
La manipulation des clefs est quelque peu pénible pour une utilisation courante. Heureusement, un outil spécifique pour dm-crypt a été créé.

Cryptsetup

cryptsetup est un utilitaire bien plus convivial que dmsetup pour manipuler les systèmes de fichiers chiffrés. Jugez plutôt :

```
% cryptsetup -y create cryptoset /dev/loop0
Enter passphrase:
Verify passphrase:
```

```
Verify passphrase:
% mke2fs /dev/mapper/cryptoset
% mount /dev/mapper/cryptoset  /mnt/loo
```

C'est bien plus simple que d'utiliser des clefs générées avec dd tout en restant parfaitement compatible :

```
 % dmsetup table
cryptoset: 0 20480 crypt aes-cbc-plain
   c693e5b5b02c492742f44e669ca3ab6e2ae2272b
   b7ceea712519243ca1f8dbe9 0 7:0 0
```

Cette table pourrait être directement réutilisée avec dmsetup create, par exemple. Le périphérique 7:0 correspond au couple majeur/mineur de /dev/loop0.
Mais cryptsetup présente un autre énorme avantage, son intégration dans le système via un fichier de configuration spécifique et un script d'init. Si nous prenons le cas de la distribution Debian stable, la simple modification du fichier /etc/crypttab est suffisante.
Ce fichier, reposant vaguement sur le principe de fstab, permet des associations entre périphériques virtuels, source et chiffrement :

```
intuix  /dev/sda1 none cipher=blowfish
```

Ici le périphérique (et l'entrée dans /dev/mapper) s'appellera intuix (fabricant de la clef USB utilisée), le périphérique source sera /dev/sda1, la clef sera non spécifiée (nécessitant une entrée manuelle) et une option précisera un chiffrement BlowFish. La page de manuel de crypttab regroupe les options et leur syntaxe.
Il ne reste plus, ensuite qu'à redémarrer le service via le script d'init :

```
 % /etc/init.d/cryptdisks restart
Stopping crypto disks: intuix(stopping).
Starting crypto disks: intuix(starting)...
Enter passphrase:
.
```

Nous obtenons un /dev/mapper/intuix que nous pouvons monter et utiliser à souhait. Notez qu'une clef USB pose quelques problèmes à ce niveau en raison de sa nature " nomade ".
Le démarrage du service en l'absence de cette dernière provoquera l'affichage d'une erreur Command failed: Bloc de périphérique requis.

CryptoHome

Nous pouvons pousser plus loin la configuration en proposant un système de fichier chiffré sur clef USB. Les indications qui vont suivre concernent une distribution Debian stable et devront être adaptées pour toute autre distribution en fonction des configurations d'usage.
Pour mettre en place le système, nous commençons par installer libpam-mount. Il s'agit d'un module PAM méritant presque un article complet.
Il permet de monter automatiquement un système de fichier via le système d'authentification modulaire unifié PAM. Le service concerné est login, mais il pourra être facilement étendu à d'autres comme xdm ou gdm.
Commençons par ajouter la prise en charge du module pour le service en ajoutant une ligne à /etc/pam.d/login :

```
@include common-pammount
```

Un fichier common-pammount a été installé en même temps que le paquet libpam-mount et contient :

```
auth       optional   pam_mount.so use_first_pass
session    optional   pam_mount.so use_first_pass
```

Pour l'heure, nous sommes en phase de test, mais le moment venu nous passerons la directive optional en s. Pour la suite, nous suivrons la documentation Debian et commençons par créer une clef SSL protégée par un mot de passe identique à celui du compte utilisateur concerné :

```
% dd if=/dev/urandom bs=1c count=32 | \
openssl enc -aes-256-ecb > /home/test.key

32+0 enregistrements lus.
32+0 enregistrements écrits.
32 bytes transferred in 0.000987 seconds (32421 bytes/sec)
enter aes-256-ecb encryption password:
Verifying - enter aes-256-ecb encryption password:% openssl enc -d -aes-256-ecb -in /home/test.key
```

Puis, nous créons notre périphérique virtuel en utilisant la clef USB :

```
cryptsetup -c twofish -h sha512 -s 256 create \
_dev_sde1 /dev/sde1
enter aes-256-ecb decryption password:
```

Enfin, nous initialisons le système de fichiers et le montons avant d'y copier les données :

```
% mke2fs /dev/mapper/_dev_sde1
% mount /dev/mapper/_dev_sde1 /mnt/loo
% cd /home/moi
% find . -xdev | cpio -pm /mnt/loo
```

```
% find . -xdev | cpio -pm /mnt/loo
4061 blocks
% umount /mnt/loo
% cryptsetup remove _dev_sde1
```

L'utilisateur concerné est ~~moi~~ et nous avons, à présent, une version chiffrée des données de son répertoire personnel sur la clef USB.
Il ne nous reste plus qu'à configurer ~~pam_mount~~ de manière à automatiquement monter ce système de fichier lors de l'utilisation sur service ~~login~~.
Pour cela, nous ajoutons une ligne dans ~~/etc/security/pam_mount.conf~~ :

```
volume moi crypt - /dev/sde1 /home/moi
nodev,nosuid,fstype=ext2,cipher=twofish,
hash=sha512,keysize=256 aes-256-ecb /home/test.key
```

Tout est en place pour une première authentification. Par défaut, les options d'affichage en mode debug sont activées (et c'est heureux).
Nous voyons clairement le déroulement des opérations en même temps que nous nous connectons. Ces traces sont également enregistrées dans ~~/var/log/auth.log~~.
On y retrouve ainsi la configuration utilisée :

```
pam_mount: user:          moi
pam_mount: server:
pam_mount: volume:        /dev/sde1
pam_mount: mountpoint:    /home/moi
pam_mount: options:       nodev,nosuid,fstype=ext2,
    cipher=twofish,hash=sha512,keysize=256
pam_mount: fs_key_cipher: aes-256-ecb
pam_mount: fs_key_path:   /home/test.key
```

Ainsi que les commandes lancées :

```
pam_mount: realpath of volume (/home/moi) is (/home/moi)
pam_mount: command: /bin/mount [-t] [crypt] [-onodev,nosuid,fstype=ext2,
cipher=twofish,hash=sha512,keysize=256] [/dev/sde1] [/home/moi]
pam_mount: command: /usr/sbin/pmvarrun [-u] [moi] [-d] [-o] [1]
pam_mount: pmvarrun says login count is 1
pam_mount: done opening session
```

La commande ~~realpath~~ n'est pas un problème pour la connexion, mais son absence est bloquante pour la déconnexion alors que le paquet du même nom n'est pas requis lors de l'installation.
Sans ~~realpath~~, le périphérique virtuel n'est pas supprimé par ~~cryptsetup~~ lorsque l'utilisateur n'est plus là.
~~pmvarrun~~ est un élément très important. C'est cette commande qui tiendra un compte des connexions en cours pour notre utilisateur.

Lorsque le nombre descend à 0, le système de fichiers est démonté et le périphérique est éliminé.
Si vous rencontrez des problèmes pour cette étape, assurez-vous que l'option ~~CLOSE_SESSIONS~~ soit à ~~yes~~ dans votre ~~/etc/login.defs~~ et que le contenu de ~~/var/run/pam_mount/moi~~ corresponde au nombre de sessions ouvertes.
Si ce n'est pas le cas, ajustez la valeur avec ~~pmvarrun -u moi -o -1~~ (où ~~moi~~ est l'utilisateur concerné et ~~-1~~ permet de décrémenter de 1).
Une fois l'utilisateur ~~moi~~ connecté, on doit avoir un mapper ~~_dev_sde1~~ et un système de fichiers monté. Dès la déconnexion, on retrouve dans les logs les commandes utilisées :

```
pam_mount: command: /usr/sbin/pmvarrun [-u] [moi] [-d] [-o] [-1]
pam_mount: pmvarrun says login count is 0
pam_mount: going to unmount
pam_mount: command: /usr/bin/umount.crypt [/home/moi]
pam_mount: waiting for umount
pam_mount: pam_mount execution complete
pam_mount: user is moi
```

On retrouve ~~pmvarrun~~ et on constate qu'il s'agissait là de la dernière connexion. Cet état de fait provoque le démontage et l'utilisation interne de ~~cryptsetup remove~~.
Cet exemple montre une nouvelle utilisation du device mapper ~~dm-crypt~~. Ici, il ne s'agit plus de chiffrer le système de fichiers avec un mot de passe, mais avec une clef AES protégée par un mot de passe.
Ceci rend dépendante la clef USB de la machine hôte, mais ce n'est pas vraiment un problème. C'est une utilisation assez rare et on lui préfèrera souvent le chiffrement d'une partition du disque spécialement dédiée.
En cas de vol de la machine, il sera très difficile (voire impossible) au " vilain " d'accéder aux données chiffrées sans le mot de passe protégeant la clef. Bien entendu, en cas de problème concernant l'intégrité du système de fichiers et en particulier des données contenues dans ~~test.key~~, c'est vous qui serez coincé. Mais la sécurité est à ce prix, malheureusement.
Dans tous les cas, observez une phase de test et inspectez chaque message du système. Bon nombre de commandes entrent en jeu et le mauvais fonctionnement de l'une d'entre elles peut rendre l'ensemble incohérent (persistance du mapper, décomptage des sessions, etc.).

CryptoRoot et swap

Je ne m'étendrai pas ici sur ces deux possibilités. Chiffrer le swap ajoute une certaine sécurité, puisque les données de mémoire swappées ne sont plus accessibles via un boot sur CD par exemple. Le chiffrement de la racine du système est quelque chose de très sécurisé, mais également de très risqué

système est quelque chose de très sécurisé, mais également de très risqué.
Le CryptoRoot howto livré avec cryptsetup explique en détail, et étape par étape, comment procéder pour basculer votre système sur une partition racine chiffrée. Voici, dans les grandes lignes, comment procéder. L'image du noyau et de l'initrd devront se trouver sur une partition séparée et non chiffrée. On commencera par configurer une partition dédiée pour la future racine via /etc/crypttab (voir plus haut).
Une fois le mapper installé et le système de fichiers initialisé, il vous faudra copier le contenu de la racine actuelle vers le périphérique chiffré avec cp -axv ou cpio -pm.
Editez le fstab de la future racine de manière à l'adapter au mapper et chrootez sur celle-ci.
Montez ensuite procfs, sysfs et votre partition /boot et lancez mkinitrd -o pour créer une nouvelle image initrd prenant en compte tout cela.
Enfin, configurez votre bootloader en ajoutant /dev/mapper/xxx comme partition racine et le tour est joué.
Veillez à faire des sauvegardes avant toute tentative. Bien que cette technique permette de conserver l'ancienne partition racine parfaitement utilisable, une telle prudence n'est pas un luxe.

CD-ROM

Les disques (internes, USB, autres) ne sont pas les seuls supports qu'il soit possible de chiffrer afin de protéger les informations qu'ils contiennent. Les supports de stockage amovibles comme les CD-R et les DVD méritent qu'on y consacre quelques lignes. Il est bien plus facile de se faire voler un CD contenant la comptabilité d'une société qu'un PC complet. Ceci est également valable pour les sauvegardes stockées à distance (chez des employés ?). Pour créer un CD chiffré utilisable via cryptsetup, on commence par créer une image ISO standard :

```
% mkisofs -R -r -V "essai" -v -o \
 essai.iso LA_SOURCE
```

On utilisera ensuite le très pratique aespipe (paquet du même nom avec Debian), afin de chiffrer l'image :

```
 % aespipe -H sha256 -e aes256 < \
 essai.iso > essai_chiffre.iso
Password:
```

Dès lors, l'image est déjà utilisable :

```
% losetup /dev/loop0 essai_chiffre.iso
% cryptsetup -c aes -s 256 -h sha256 \
create cryptocd /dev/loop0
Enter passphrase:
% mount -oro /dev/mapper/cryptocd /mnt/cdrom
```

Si libpam-mount est installé (mais pas forcément utilisé), vous pourrez profiter des scripts mount.crypt et umount.crypt et vous simplifier la vie :

```
% losetup -d /dev/loop0
% mount -t crypt \
-oro,loop,cipher=aes,hash=sha256,\
keysize=256,fstype=iso9660 /dev/loop0 /mnt/cdrom
Enter passphrase:
% umount.crypt /mnt/cdrom
```

Nous déconfigurons /dev/loop0, car le script est capable de monter directement une image et utiliser losetup au besoin. Pour obtenir un CD chiffré, il suffira de graver essai_chiffre.iso avec cdrecord.

Conclusion
Les projets comme CryptoLoop et Loop-AES sont toujours supportés de-ci, de-là. C'est le cas, par exemple, pour pam_mount, mais il faut bien avouer que la technique proposée par le device mapper est plus souple. De plus, dm-crypt est largement documenté sur le site officiel (http://www.saout.de/misc/dm-crypt/ et http://www.saout.de/tikiwiki/tiki-index.php). On trouvera dans la masse de documentation des indications pour tout ce qu'il est possible de faire avec cette solution, des partitions racine sur clef USB au CryptoRAID en passant par LVM2 sur dm-crypt...