

Stockage des données confidentielles sous GNU/Linux

Piotr Tyburski



Seul l'usage des algorithmes cryptographiques avancés peut garantir la sécurité des données confidentielles. Les outils disponibles dans Linux permettent de chiffrer les fichiers simples, les répertoires sélectionnés ou même les partitions entières. Analysons les méthodes de chiffrement des données importantes.

Imaginons la situation suivante : nous venons d'être employés dans une société qui s'occupe de la conception de dispositifs électroniques et de logiciels pour ces dispositifs. Avec le contrat, nous devons signer le document appelé *confidentiality agreement*, d'après lequel toute information concernant la société, quelque soit le support, est confidentielle et est protégée contre toute divulgation aux personnes non-autorisées. Le dernier paragraphe nous avertit des conséquences si ces informations étaient révélées – c'est vraiment très désagréable.

Mais pour travailler, nous avons besoin de temps en temps de prendre avec nous de la documentation ou du code source. Pour cela, nous utilisons notre ordinateur portable. Qu'est-ce qui se passe si nous le perdons ou quelqu'un le vole et trouve sur son disque dur les documents désignés comme *confidentiels* ? Il est raisonnable de nous protéger contre une telle possibilité – réfléchissons comment le faire.

Chiffrement des fichiers simples

Si nos fichiers confidentiels ne sont pas trop nombreux, et que leur taille ou quantité ne

doivent pas être gardées secrètes, nous pouvons utiliser le paquet *gpg* (*Gnu Privacy Guard*) étant le correspondant du PGP (*Pretty Good Privacy*), mais plus récent et publié sous licence GPL. Il permet le chiffrement symétrique et asymétrique (avec clé publique) (cf. l'Encadré *Chiffrement symétrique et asymétrique*).

Imaginons une telle situation : nous avons une spécification très importante d'un nouveau projet enregistrée dans le fichier portant le nom *DD32X.pdf*. Nous voulons cacher son contenu contre un accès non-autorisé. Tapons alors la commande suivante :

```
$ gpg --output cyph1.1 --symmetric \  
--cipher-algo TWOFISH DD32X.pdf
```

Cet article explique ...

- comment, à l'aide des outils gratuits, assurer la protection cryptographique des données confidentielles.

Ce qu'il faut savoir ...

- fonctionnement du système Linux.

Démarrage rapide – chiffrage d'un fichier simple

Pour chiffrer un fichier simple, par exemple *DD32X.pdf*, nous tapons la commande suivante :

```
$ gpg --output cyph1.1 --symmetric --cipher-algo TWOFISH DD32X.pdf
```

La version chiffrée du fichier est placée dans le fichier *cyph1.1*. Maintenant, nous pouvons supprimer la version non cryptée du fichier. Exécutons cette opération de façon sûre :

```
$ shred -n 35 -z -u DD32X.pdf
```

Quand nous voulons déchiffrer le fichier, il suffit de taper la commande suivante :

```
$ gpg --output DD32X.pdf --decrypt cyph1.1
```

Dans l'exemple ci-dessus, on admet que le paquet *gpg* est installé ou nous utilisons *Hakin9 Live*.

Démarrage rapide – création d'un répertoire chiffré

Pour créer un un répertoire chiffré, nous tapons la commande :

```
# encfs ~/.crypto ~/secret
```

À la suite de l'exécution de cette commande, le répertoire chiffré *~/crypto* est créé. Sa version déchiffrée se trouve dans le répertoire *~/secret* (c'est dans ce répertoire qu'il faut placer les fichiers à protéger). Quand l'accès au répertoire chiffré n'est plus nécessaire, nous pouvons le démonter à l'aide de la commande :

```
# fusermount -u ~/secret
```

Dans l'exemple ci-dessus, il est admis que le paquet *EncFS* est installé ou nous utilisons *Hakin9 Live*.

Démarrage rapide – création d'un système de fichiers chiffré

Pour créer un système de fichiers chiffré d'une taille de 100 Mo, nous tapons la commande :

```
$ dd if=/dev/urandom of=crypto.raw bs=1k count=100000
# losetup -e aes-256 /dev/loop0 crypto.raw
# mkfs.ext3 /dev/loop0
# losetup -d /dev/loop0
```

À la suite de l'exécution de cette commande, le système de fichiers chiffré est créé dans le fichier *crypto.raw*. Si nous voulons l'utiliser, il faut le monter au moyen de la commande :

```
# mount -t ext3 crypto.raw /mnt/crypt -oencryption=aes-256
```

Le système de fichiers sera monté dans le répertoire */mnt/crypt*.

Dans l'exemple ci-dessus, il est admis que le noyau est bien configuré et le paquet *util-linux* est mis à jour (la description détaillée dans le HOWTO approprié – cf. l'Encadré *Sur le réseau*) ou nous utilisons *Hakin9 Live*.

gpg nous demande le mot de passe qui protégera nos données, et ensuite, demande de le confirmer. Le mot de passe devrait être compliqué, mais en même temps facile à retenir (cf. l'Encadré *Mots de passe d'une grande entropie*).

Analysons la commande que nous avons tapée. Nous avons demandé à *gpg* de chiffrer le fichier *DD32X.pdf* avec la clé symétrique (*--symmetric*), à l'aide de l'algorithme *TWOFISH* (*--cipher-algo TWOFISH*). Si cette dernière option n'était pas

déterminée, l'algorithme par défaut *CAST5* serait utilisé. Le fichier d'origine était *cyph1.1* (*--output cyph1.1*). Si un fichier portant ce nom existait déjà, le programme aurait demandé si vous vouliez le remplacer. Pour terminer, il faut maintenant supprimer la version non chiffrée.

Une simple suppression du fichier, par exemple à l'aide de la commande *rm DD32X.pdf*, n'est pas sûre. Cette opération ne supprime pas vraiment le fichier, seule l'information des structures décrivant le contenu du système de fichiers est supprimée, mais le contenu du fichier reste sur le disque (c'est comme si nous avions supprimé un chapitre d'un livre du sommaire, et avions oublié d'arracher les feuilles de ce chapitre). Dans ce cas, une personne malintentionnée peut voler notre disque, et ensuite, y extraire les fragments d'un fichier important.

Pour nous protéger contre cette situation, nous devons remplacer le contenu du fichier. Pour cela, tapons la commande :

```
$ shred -n 35 -z -u DD32X.pdf
```

À la suite de cette opération, le fichier sera remplacé 35 fois par des déchets (*-n 35*), ensuite par des zéros (*-z*), et à la fin supprimé (*-u*).

Hélas, cette méthode n'est pas efficace dans les systèmes de fichier utilisant la journalisation (*ext3*, *ReiserFS*, *xfs*, *jfs*). La journalisation consiste en l'enregistrement dans un lieu séparé (appelé journal – en anglais *journal*) des informations supplémentaires sur toute modification effectuée dans le système de fichiers (dans le système *ext3* ce ne sont pas seulement les informations sur la structure du système, mais les données mêmes). Elle permet de reconstituer des informations lors de la phase de restauration dans le cas d'une panne. Dans cette situation nous ne pouvons pas être sûrs que le remplacement d'un fichier par une chaîne aléatoire supprime complètement son contenu – les systèmes de fichiers avec journalisation ne permettent