# Systèmes de fichiers chiffrés

|  |
| --- |
| **Avertissement** …  Les manipulations présentées ici ont le potentiel de rendre le système inopérant (inbootable), de détruire des données si des erreurs sont faites dans l'exécution des commandes.<color blue> Vous devez faire une sauvegarde de l'ensemble de vos données, de préférences sur un disque amovible qui ne sera pas connecté pendant les manipulations</color> (évite le risque de confusion de partitions).  Les opérations d'écriture de données aléatoires sur une partition, de création d'un conteneur chiffré, de formatage provoquent la destruction de toutes les données contenues sur la partition visée, utilisez ce tutoriel en connaissance de cause. |

# dm-crypt et cryptsetup

## Systèmes de fichiers chiffrés

La plupart des administrateurs ont déjà utilisé un logiciel pour chiffrer des fichiers, tels que GnuPG. Cette technologie, appelée chiffrement au niveau des fichiers, est conçue pour chiffrer des fichiers spécifiques, généralement pour assurer leur transport sécurisé. Le but des systèmes de fichiers chiffrés est similaire, mais est normalement exécuté pour une raison différente.

Il peut arriver une situation dans laquelle on doit transférer des milliers de fichiers vers une autre personne sur une clé USB. Le chiffrement de chaque fichier individuellement serait un processus fastidieux (tout comme le processus de déchiffrement). Le chiffrement de l'ensemble du système de fichiers offre un avantage en termes de sécurité (en fait, plus d'avantages) que le chiffrement de fichiers individuels.

En quoi le chiffrement de tout le système de fichiers est-il meilleur que le chiffrement de fichiers individuels ?   
Il faut tenir compte du fait que, lorsque l’on chiffre un seul fichier, certaines des données du fichier (propriété, autorisations, horodatage, ...) restent visibles pour une menace potentielle. Si on chiffre l’ensemble du système de fichiers, ces informations ne seront disponibles que si le système de fichiers est déchiffré.

De plus, si un ordinateur portable contient des données sensibles, le chiffrement des systèmes de fichiers peut empêcher tout accès non autorisé en cas de vol ou de perte d’un ordinateur.

Sous Linux, l’outil cryptsetup offre un moyen simple et commode pour chiffrer un disque complet et donc protéger tous les fichiers inclus. Ce disque complet peut toutefois être une partition, une clé USB, ou un conteneur. L’outil cryptsetup s'appuie sur le module noyau dm-crypt, activé dans toutes les distributions Linux récentes. De plus, cryptsetup fonctionne étroitement avec la surcouche LUKS (Linux Unified Key Setup) qui permet de gérer les clés de chiffrement de façon très souple (ainsi un conteneur peut être partagé par 8 utilisateurs). Cette souplesse est possible

### Bien chiffrer

Pour que le chiffrement soit efficace, il ne faut pas faire d'erreur lors de la mise en place. Il est important de bien réfléchir aux implications.

Chiffrer uniquement une partition de données peut sembler suffisant, mais cela laissera exposées de nombreuses informations contenues dans les répertoires cachés dans son répertoire personnel. Les cookies, caches et profils des fureteurs (*browsers*) Internet, de nombreux clients de messagerie y stockent les répertoires des courriels, l'historique de vos sessions de clavardage.

Il faut donc au minimum chiffrer son répertoire personnel, et pour que cela soit vraiment efficace, il faut également toujours chiffrer la partition d'échanges (*swap*). Tout ce qui est écrit sur la partition d’échange sera donc accessible à des techniques de récupération de données. Si le poste de travail posséde peu de mémoire vive ou encore on laisse des applications ouvertes inactives pour de longues périodes, on risque de voir des données sensibles résider sur la partition d’échange. Si on utilise la mise en veille sur disque l'ensemble de la session sera écrite sur la partition d’échange et cela peut totalement annihiler les efforts de protection. Cela est d'autant plus lourd de conséquences que dans certaines circonstances la clé de chiffrement des conteneurs luks peut se trouver écrite dans le fichier d’échange.

Le cas de la partition système est plus nuancé car si on souhaite seulement se prémunir contre le vol de son ordinateur, il n'est pas toujours indispensable de chiffrer la partition système. Si en revanche on a des impératifs de sécurité stricts, il est toujours possible de chiffrer le système afin d’éviter la mise en place d'un cheval de Troie ou autre espiogiciel destiné à enregistrer les mots de passe ou clé de chiffrement (attaque dite *Evil maid*). Il faudra alors au minimum conserver une partition /boot non chiffrée. De plus, il est possible de de placer la partition /boot sur un média amovible (clé USB) afin qu'elle ne puisse pas être manipulée à son insu.

Enfin si on décide de ne pas chiffrer la partition système, il vaudrait mieux chiffrer les partitions ou répertoires /tmp (données temporaires), /var/ (en particulier pour /var/mail et /var/log). De plus, en fonction de l'usage du système, il est prudent de chiffrer /var/www et autres répertoires.

**Mots de passe**Il est essentiel d’utiliser un mot de passe de bonne qualité ou une clé générée de manière aléatoire et stockée sur un média amovible lui-même chiffré (par mot de passe). Il ne sert à rien d'utiliser un robuste chiffrement si le conteneur est protégé par un mot de passe faible.

Les partitions destinées à être chiffrées devront également être préparées par l'écriture de données (pseudo) aléatoires auparavant, sinon la cryptanalyse en sera facilitée, surtout sur un disque neuf où les blocs ont été mis à zéro en usine.

La préparation peut être effectuée avec une commande comme …  
# dd if=/dev/urandom of=<Périphérique>  
# dd if=/dev/urandom of=/dev/sdc1

ou alternativement …  
# badblocks -wt random <Périphérique>  
# badblocks -wt random /dev/sdc1

### Précisions quant au temps utilisé.

Le temps utilisé va varier qu'on veuille utiliser le réservoir d'entropie de /dev/random, ou le réservoir d'entropie de /dev/urandom. dev/urandom ou badblocks ne fournissent pas des données réellement aléatoires mais pseudo-aléatoires.

Pour obtenir le niveau maximum de sécurité, il faudrait utiliser /dev/random à la place de /dev/urandom, mais pour fonctionner /dev/random nécessite un réservoir d'entropie confortable. Les données aléatoires sont dérivées du bruit ou entropie générée par l'activité de certains matériels dont les disques durs et leurs pilotes, si le réservoir est à zéro la commande dd se mettra en pause, augmentant d'autant la durée de l'opération. Vous pouvez avoir une idée de la taille du réservoir d'entropie en regardant les valeurs sous /proc/sys/kernel/random/.

Il existe des générateurs d'entropie matériels qui se présentent sous la forme d'un périphérique USB mais leur qualité varie autant que leur prix, mais ce type d'appareils est la meilleur façon d'obtenir des données aléatoires en quantité suffisante.

Des générateurs de données aléatoires logiciels existent également, ils utilisent souvent le trafic réseau ou communiquent avec un générateur d'entropie distant.

Pour un usage courant /dev/urandom fera très bien l'affaire. Si on génére des clés de chiffrement (petite quantité de données nécessaire) pour ses conteneurs LUKS préférez /dev/random.

### Limites du chiffrement

Quel que soit la méthode employée le chiffrement a ses limites.   
Le chiffrement est très efficace dans le cas d'un vol d'un système éteint mais, en revanche, le chiffrement est contournable si l'attaquant a un accès physique à la machine (remplacement du bios, espiogiciels installés dans le secteur d'amorçage du disque ou le *firmware* d'un périphérique, …).

Bien entendu, il ne faut jamais perdre de vue que l’on doit conserver des sauvegardes de ses données (chiffrées elles aussi), car en cas de problème sur les conteneurs luks (en-têtes du conteneur endommagées, mot de passe ou clé perdus, …) les données seront irrécupérables  
Il n'y a aucun espoir si le chiffrement a été correctement mis en place. Il ne reste qu'à attendre et espérer que dans un futur pas trop lointain une faille de sécurité, une avancée spectaculaire en mathématique ou un bond de la puissance de calcul des ordinateurs permettra de casser le chiffrement.

Les en-têtes de conteneurs chiffrés luks sont extrêmement sensibles à la corruption, il suffit d'un seul octet endommagé à cet endroit pour rendre le conteneur indéchiffrable (pensez perte d'un secteur du disque dur). Il est impératif de conserver une copie de sauvegarde de ces en-têtes sur un ou plusieurs média séparés (clé USB, cédérom) afin de pouvoir les restaurer.

La résistance à la cryptanalyse de ces sauvegardes est plus faible que pour les en-têtes originaux. Il est essentiel de les garder en lieu sûr. Si les mots de passe ou clés de déchiffrement sont compromis, et qu’on les modifie, il faudra faire à nouveau ces sauvegardes et détruire totalement les anciennes.   
En restaurant les anciennes en-têtes un attaquant pourrait déchiffrer le conteneur à l'aide des anciens mots de passe ou clé de déchiffrement.

**Algorithmes de chiffrement**

Linux prend donc en charge, au niveau du noyau, le chiffrement du système de fichiers natif. On peut choisir parmi plusieurs algorithmes de chiffrement symétriques pour chiffrer un système de fichiers …

* **Norme twofish**   
  Twofish prend en charge une taille de bloc de 128 bits et les clés jusqu'à 256 bits ;
* **Norme AES** (*Advanced Encryption Standard* également appelée Rijndael)   
  (pour des raisons de performances, Rijndael a été choisi comme nouveau standard AES) ;
* **Norme DES** (*Data Encryption Standard*)   
  DES est le prédécesseur de la norme AES et est maintenant considéré comme non sécurisé en raison de la petite taille de la clé. Avec la puissance de calcul actuelle, il est possible d’utiliser une force brute de 56 bits (+8 bits de parité) avec la clé DES dans un laps de temps relativement court;
* ...

Comme mentionné plus haut, une force brute de 56 bits (+8 bits de parité) avec la clé DES peut être cassée dans un laps de temps relativement court. AES utilise une taille de bloc de 128 bits et une taille de clé de 128, 192 ou 256 bits.   
Pendant de nombreuses années, une taille de clé de 128 bits était suffisante, mais avec l’introduction des ordinateurs quantiques, la *US National Security Agency* a publié des directives pour la classification des données jusqu’à Top Secret avec des clés de 256 bits.

Intel a introduit en 2010 l'ensemble AES-NI (*Advanced Encryption Standard New Instructions*).   
Ce nouveau jeu d'instructions effectue le chiffrement et le déchiffrement complètement au niveau du matériel, ce qui contribue à réduire le risque d'attaques par canal secondaire et améliore considérablement les performances AES.

Pour vérifier si son CPU prend en charge le jeu d’instruction AES-NI, la sortie de la commande suivante devrait ressembler à …  
# sort -u /proc/crypto | grep module   
module : aesni\_intel  
module : aes\_x86\_64  
module : crc32c\_generic  
module : crc32c\_intel  
module : crc32\_pclmul  
module : crct10dif\_pclmul  
module : cryptd  
module : ghash\_clmulni\_intel  
module : kernel

On peut alors charger le module …  
# modprobe aesni-intel

### Device Mapper

À partir de Linux 2.6, il est possible d'utiliser **device mapper**, un cadre de travail (*framewrok*) Linux générique pour mapper un périphérique bloc à un autre.

device mapper est utilisé pour les logiciels RAID et LVM. Il sert de filtre entre le système de fichiers d’un périphérique virtuel et les données chiffrées à écrire sur un disque dur. Cela permet au système de fichiers de se présenter déchiffré tandis que tout ce qui est lu et écrit sera chiffré sur le disque.

Un périphérique de bloc virtuel est créé dans **/dev/mapper,** qui peut être utilisé comme tout autre périphérique de bloc. Toutes les données entrantes et sortantes vont dans un filtre de chiffrement et de déchiffrement avant d'être mappées vers un autre périphérique bloc.

### Spécification LUKS

LUKS (*Linux Unified Key Setup*) est une spécification, car elle décrit comment les systèmes de fichiers doivent être chiffrés sous Linux. Elle ne fournit aucun logiciel et n’est pas une norme officielle   
(les spécifications sont communément appelées normes non officielles).

Comme il s’agit d’une spécification, LUKS n’oblige pas à utiliser un outil logiciel spécifique pour chiffrer un système de fichiers. Différents outils sont disponibles dont une qui est une implémentation basée sur un noyau appelée **dm-crypt**.

LUKS est un système de chiffrement de disque créé en 2004. À la base créé pour fonctionner sous Linux, LUKS a aujourd'hui l'avantage d'être un système indépendant du syst;me d’Exploitation et surtout indépendant du programme de chiffrement/déchiffrement que l'on utilise. Cela signifie que plusieurs programmes peuvent traiter le chiffrement/déchiffrement d'un système de chiffrement de disque LUKS, sous Linux comme sous Windows.

LUKS est un utilitaire de type *Block Layer Encryption* (Chiffrement au niveau du bloc). Cela signifie que tout est chiffré sur la partition, cela est aussi appelé *on-disk encryption* ou *full-disk encryption*. On le différencie du *File-System Layer encryption* qui lui ne chiffre uniquement qu'au niveau fichier/dossier.

|  |
| --- |
| A noter que depuis Debian Buster (version 2.1.0 de cryptsetup), le formatage luks par défaut est de type 2, alors qu'il était précédemment de type 1.  Ce nouveau type peut ne âs être pris en charge par GRUB et ne doit donc pas être utilisé pour une partition de boot ou contenant le répertoire de boot.  Pour forcer un formatage de type 1, on doit utiliser l'option --type luks1 …  # cryptsetup luksFormat --type luks1 -s 512 -h sha512 <Périphérique>  # cryptsetup luksFormat --type luks1 -s 512 -h sha512 /dev/sdb1  Pour vérifier le type de formatage …  # cryptsetup luksDump <Périphérique> | grep luks # cryptsetup luksDump /dev/sdb1 | grep luks  0: luks2  Pour modifier le type de formatage, par exemple de luks2 en luks1 : le container étant fermé et démonté …  # cryptsetup convert <Périphérique> --type luks1 # cryptsetup convert /dev/sdb1 --type luks1  **Remarque** … Il est prudent de faire une sauvegarde de l'entête avant la conversion. |

### cryptsetup

cryptsetup fournit une interface pour la configuration du chiffrement sur les périphériques en mode bloc (comme les partitions /home ou /swap), à l'aide du mappeur de périphérique du noyau Linux cible dm-crypt. Il offre une prise en charge intégrée LUKS (*Linux Unified Key Setup*).

cryptsetup est rétro-compatible avec le format sur disque de cryptoloop, mais prend également en charge des formats plus sécurisés. Ce paquet inclut la prise en charge de la configuration automatique des périphériques chiffrés au démarrage à l’aide du fichier de configuration /etc/crypttab.   
Les fonctionnalités supplémentaires sont la prise en charge de cryptoroot via initramfs-tools et plusieurs méthodes prises en charge pour lire une phrase secrète ou une clé.

En réalité, c'est l'utilitaire dm-crypt qui fait le travail de chiffrement, cryptsetup est une interface en ligne de commande permettant de gérer les fonctionnalités et les actions de dm-crypt, un peu comme iptables avec netfilter.

Chiffrement de mappeur de périphérique (dm-crypt)

dm-crypt est un sous-système transparent de chiffrement de disques dans le noyau Linux versions 2.6 et supérieur.   
Il est une partie de l'infrastructure device-mapper, et utilise des routines de chiffrement issues des API crypto du noyau. Contrairement à son prédécesseur cryptoloop, dm-crypt a été conçu pour prendre en charge des modes d'opération avancées, tel XTS, LRW et ESSIV dans le but d'éviter les attaques watermarking.   
En plus de cela, dm-crypt résout aussi quelques problèmes de fiabilité de cryptoloop1.

dm-crypt est conçu comme une cible device mapper et peut être empilé sur d'autres transformations du même type. Il peut ainsi chiffrer des disques entiers (incluant les disques amovibles), partitions, volumes RAID, volumes logiques, aussi bien que des fichiers.

Il apparait en tant que périphérique bloc, qui peut contenir des systèmes de fichiers, partitions swap ou un volume physique LVM.

Quelques distributions GNU/Linux autorisent l'utilisation de dm-crypt sur le système de fichier racine (chiffrement complet du système d'exploitation). Ces distributions utilisent initrd pour demander à l'utilisateur d'entrer un mot de passe, ou d'insérer un périphérique avant le processus normal de démarrage.

Le chiffrement de mappeur de périphérique fournit un moyen générique de chiffrement transparent de périphériques en mode bloc à l'aide de l'API du noyau et peut être utilisé en combinaison avec des volumes RAID ou LVM.

Lorsque l'utilisateur crée un nouveau périphérique bloc, il peut spécifier plusieurs options …

* chiffrement symétrique ;
* mode de chiffrement ;
* mode de génération de clé ;
* …

dm-crypt ne stocke aucune information dans un en-tête comme le fait LUKS.   
Après le chiffrement du disque, il sera impossible de le distinguer d’un disque contenant des données aléatoires. Cela signifie que l'existence de fichiers chiffrés est indétectable en ce sens qu'il ne peut pas être prouvé que des données chiffrées existent.

L'utilisateur doit suivre les options utilisées dans la configuration de **dm-crypt**, faute de quoi des données pourraient être perdues, car aucune métadonnée n'est disponible. Une seule clé de chiffrement peut être utilisée pour chiffrer et déchiffrer des périphériques bloqués et aucune clé principale ne peut être utilisée. Une fois que le mot de passe d'un périphérique bloc chiffré est perdu, il est impossible de récupérer les données.

dm-crypt ne devrait être utilisé que par des utilisateurs avancés.   
Les utilisateurs réguliers doivent utiliser LUKS pour le chiffrement du disque.

dm-crypt est un module de noyau qui permet à ce dernier de comprendre les systèmes de fichiers chiffrés. Il offre deux principales commandes pour créer et monter un système de fichiers chiffré … les commandes **cryptsetup** et **cryptmount**. Toutefois, on utilise principalement que l’une des deux commandes (la commande **cryptsetup**) pour configurer un système de fichiers chiffré.

## Chiffrement d’un disque ou partition

Alors que la plupart des logiciels de chiffrement de disque implémentent des formats différents, incompatibles et non documentés, LUKS met en œuvre un format standard sur disque standard indépendant de la plate-forme pour une utilisation dans divers outils. Cela facilite non seulement la compatibilité et l'interopérabilité entre différents programmes, mais garantit également qu'ils implémentent tous la gestion des mots de passe de manière sécurisée et documentée.

L'implémentation de référence pour LUKS fonctionne sous Linux et est basée sur une version améliorée de cryptsetup, utilisant dm-crypt comme utilitaire de chiffrement de disque en arrière-plan.

|  |
| --- |
| Dans ce qui suit, on utilise les conventions suivantes …   * Identifiant de la partition physique du container chiffré luks … **/dev/sdb1** * Nom du mappage … **secret** * Point de montage … **/data** |

**Étape 01  
Partitionnement**

La commande fdisk permet d’afficher les partitions d’un disque donné …  
# fdisk -l /dev/sdb  
Disque /dev/sdb : 10 GiB, 10737418240 octets, 20971520 secteurs  
Modèle de disque : VMware Virtual S  
Unités : secteur de 1 × 512 = 512 octets  
Taille de secteur (logique / physique) : 512 octets / 512 octets  
taille d'E/S (minimale / optimale) : 512 octets / 512 octets

Toutefoism la commande fdisk est avant-tout interactive …  
# fdisk /dev/sdb  
Bienvenue dans fdisk (util-linux 2.33.1).  
Les modifications resteront en mémoire jusqu'à écriture.  
Soyez prudent avant d'utiliser la commande d'écriture.  
  
Le périphérique ne contient pas de table de partitions reconnue.  
Création d'une nouvelle étiquette pour disque de type DOS avec identifiant de di sque 0x3f809170.  
  
Commande (m pour l'aide) :

Pour créer une table de partition vide …

**Commande (m pour l'aide) :** o (lettre o minuscule)  
Création d'une nouvelle étiquette pour disque de type DOS avec identifiant de disque 0x7862b7fd.

Afin de créer une partition tout en laissant les valeurs par défaut …  
(la partition occupera l’espace maximal disponible) …  
**Commande (m pour l'aide) :** n (lettre n minuscule)  
Type de partition  
 p primaire (0 primaire, 0 étendue, 4 libre)  
 e étendue (conteneur pour partitions logiques)

On sélectionne l’option p (p par défaut) …  
Sélectionnez (p par défaut) : p  
Numéro de partition (1-4, 1 par défaut) : ¿  
Premier secteur (2048-20971519, 2048 par défaut) : ¿  
Dernier secteur, +/-secteurs ou +/-taille{K,M,G,T,P} (2048-20971519, 20971519 par défaut) : ¿  
Commande (m pour l'aide) : w  
La table de partitions a été altérée.  
Appel d'ioctl() pour relire la table de partitions.  
Synchronisation des disques.

Une nouvelle partition 1 de taille 10 GiB a été créée.

**Étape 02  
Chiffrement**

Afin de chiffrer la partition, on utilise la commande cryptsetup, à laquelle on indique la partition à chiffrer …  
# cryptsetup luksFormat /dev/sdb1  
  
ATTENTION: Le périphérique /dev/sdb1 contient déjà une signature pour un superblock « crypto\_LUKS ».  
  
WARNING!  
========  
Cette action écrasera définitivement les données sur /dev/sdb1.  
  
Are you sure? (Type uppercase yes): YES  
Saisissez la phrase secrète pour /dev/sdb1 : <Mot de passe>  
Vérifiez la phrase secrète : <Mot de passe>

**Remarques** …  
Le chiffrement du disque peut prendre in certain temps.  
Il est important de saisir une phrase secrète forte. Il est possible d’écrire une vraie phrase avec des espaces et de la ponctuation.  
Il n’existe aucun moyen connu à ce jour pour déchiffrer un disque si on perd cette phrase.

Le chiffrement est désormais effectif. On ne peut plus monter la partition directement. Il faudra utiliser la commande dm-crypt.

**Remarque** …  
Il est possible que cryptsetup ne soit pas installé …  
# apt update && apt upgrade -y && apt install cryptsetup -y

**Paramètres pour initialiser une partition**

Commande générique  
 # cryptsetup luksFormat -c <algorithme de chiffrement> -h <algorithme de hachage> <périphérique>

Chiffrement de base …  
 # cryptsetup luksFormat -c aes -h sha256 /dev/sdb1

Chiffrement de force moyenne …  
 # cryptsetup luksFormat -c aes-xts-plain -s 512 /dev/sdb1

Chiffrement fort …  
 # cryptsetup luksFormat -c aes-xts-plain64 -s 512 -h sha512 /dev/sda1

**Étape 03  
Accès à la partition**

Il faut d’abord accéder au périphérique chiffré (le mapper avec l’utilitaire device-mapper et le module crypt) avant de pouvoir travailler sur la partition …  
# cryptsetup open /dev/sdb1 secret  
Il faudra alors saisir la phrase secrète pour /dev/sdb1.  
# ls -l /dev/mapper/secret  
lrwxrwxrwx 1 root root 7 oct 16 10:38 /dev/mapper/secret -> ../dm-0

Il est également possible de passer la commande …  
# lsblk -f /dev/sdb  
NAME FSTYPE LABEL UUID FSAVAIL FSUSE% MOUNTPOINT  
sdb  
└─sdb1  
 crypto 8fbafdc5-8704-4ecb-8bfe-7088d3a3af05  
 └─secret  
 ext4 c2e5aae6-7cb0-4f25-a036-0087310d88b0 9,2G 0% /data

On peut également utiliser la commande blkid …  
# blkid | grep secret  
/dev/mapper/secret: UUID="c2e5aae6-7cb0-4f25-a036-0087310d88b0" TYPE="ext4"

**Étape 04  
Mise en place du système de fichier**Selon ses besoins, on peut choisir le système de fichier qui convient le mieux à l’aide de la commande mkfs.  
L’exemple qui suit va formater la partition chiffrée avec le système de fichier ext4 …  
# mkfs.ext4 -L volume-chiffre /dev/mapper/secret  
mke2fs 1.44.5 (15-Dec-2018)  
Creating filesystem with 2617088 4k blocks and 655360 inodes  
Filesystem UUID: 9bc8f1b0-322c-499d-baed-569e15b8fa17  
Superblock backups stored on blocks:  
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632  
  
Allocating group tables: done  
Writing inode tables: done  
Creating journal (16384 blocks): done  
Writing superblocks and filesystem accounting information: done

**Remarque** …  
L’**option -L** de la commande mkfs.ext4 définit la valeur vol-chiffre comme étiquette (*label*).

**Étape 05  
Démontage**

Il est important de ne pas oublier de fermer proprement la partition chiffrée …  
# cryptsetup close /dev/mapper/secret

## Manipulation d’une partition chiffrée

### Informations sur la partition chiffrée

Pour obtenir les informations sur une partition chiffrée …  
# cryptsetup status <partition>

# cryptsetup status secret  
/dev/mapper/secret is active.

type: LUKS2

cipher: aes-xts-plain64

keysize: 512 bits

key location: keyring

device: /dev/sdb1

sector size: 512

offset: 32768 sectors

size: 20936704 sectors

mode: read/write

Pour voir l'état du conteneur chiffré et les ouvertures (*slots*) utilisées …  
# cryptsetup luksDump <Périphérique>  
# cryptsetup luksDump /dev/sdb1

LUKS header information

Version: 2

Epoch: 4

Metadata area: 16384 [bytes]

Keyslots area: 16744448 [bytes]

UUID: 8fbafdc5-8704-4ecb-8bfe-7088d3a3af05

Label: (no label)

Subsystem: (no subsystem)

Flags: (no flags)

Data segments:

0: crypt

offset: 16777216 [bytes]

length: (whole device)

cipher: aes-xts-plain64

sector: 512 [bytes]

Keyslots:

0: luks2

Key: 512 bits

Priority: normal

Cipher: aes-xts-plain64

Cipher key: 512 bits

PBKDF: argon2i

Time cost: 4

Memory: 402768

Threads: 1

Salt: 72 9b bc d2 3b ef cd e4 15 10 16 c7 27 39 15 c1

d1 04 81 28 2d 94 ac 33 c4 62 07 91 3b a3 8a 5b

AF stripes: 4000

AF hash: sha256

Area offset:32768 [bytes]

Area length:258048 [bytes]

Digest ID: 0

1: luks2

Key: 512 bits

Priority: normal

Cipher: aes-xts-plain64

…

Salt: 32 95 a6 8c ea d7 2b 13 d0 ce be 4e a6 d4 08 44

6c 13 53 85 89 ae f1 89 bf ee 35 17 fe e1 97 ce

AF stripes: 4000

AF hash: sha256

Area offset:290816 [bytes]

Area length:258048 [bytes]

Digest ID: 0

## Pour en savoir plus …

### Type de chiffrement

On invoque la commande **cryptsetup luksFormat -c aes -h sha256 /dev/sdb1** pour formater la partition au type LUKS (initialiser la partition LUKS et définir la clé initiale).   
Le chiffrement sera de type AES avec un algorithme de hachage SHA256.   
Il faudra saisir une phrase de déchiffrement qui va permettre de créer un conteneur standard chiffré à l'aide de la phrase.

### Gestion des clés de chiffrement

LUKS permet d’avoir plusieurs phrases secrètes pour un même volume chiffré.

Le conteneur chiffré de manière standard va stocker la clé de chiffrement maître qui servira à ouvrir le volume chiffré.   
Il est possible d'ajouter jusqu'à 8 clefs supplémentaires dans des ouvertures (*slots*), qui déverrouillent l'accès à la clef maître.   
Il est ainsi possible, à l’aide de cette méthode, d’avoir plusieurs utilisateurs qui vont chacun ouvrir le conteneur chiffré avec leur propre clé, et il vous sera possible de révoquer les clés éventuellement compromises.

Pour ajouter une clé d'accès au conteneur chiffré précédent …  
# cryptsetup luksAddKey <Périphérique>  
# cryptsetup luksAddKey /dev/sdb1  
Entrez une phrase secrète existante :  
Entrez une nouvelle phrase secrète pour l'emplacement de clé :  
Vérifiez la phrase secrète :

**Remarque** …  
Sur d’anciennes versions de cryptsetup, il fallait avoir la partition non déchiffrée pour pouvoir ajouter une phrase secrète.

Pour supprimer une phrase secrète à l’aide de son numéro …

# cryptsetup luksDump <Périphérique>

# cryptsetup luksDump /dev/sdb1

…  
1: luks2

Key: 512 bits

Priority: normal

Cipher: aes-xts-plain64  
…

Puis on la supprime avec la commande …

# cryptsetup luksKillSlot <périrérique> <n°>  
# cryptsetup luksKillSlot /dev/sdb1 1

### Fichiers de phrase secrète

La phrase secrète peut être contenue dans un fichier.

Dans l'exemple qui suit, la phrase secrète est générée à partir de /dev/urandom et possède une longueur de 2048 caractères.

Pour générer un fichier de phrase secrète aléatoire de 2048 caractères et de réserver l'accès à l'utilisateur root …  
# mkdir -p /etc/keys  
# dd if=/dev/urandom of=/etc/clefs/phrasesecrete count=4 bs=512  
# chown 600 /etc/clefs/phrasesecrete

Pour créer un container chiffré à partir d'un fichier de phrase secrète …  
# cryptsetup luksFormat /dev/sdb1 /etc/clefs/phrasesecrete

Pour ajouter la phrase secrète dans l’ouverture (slot) 7 d'un container existant.  
(un mot de passe connu sera demandé) …  
# cryptsetup luksAddKey /dev/sdb1 --key-slot 7 /etc/clefs/phrasesecrete

Pour ouvrir un conteneur à partir d'un fichier de phrase secrète …  
# cryptsetup luksOpen /dev/sdb1 dd1 --key-file /etc/keys/phrasesecrete

### Algorithmes de chiffrement

Afin d’utiliser un algorithme de chiffrement spécifique, il faut le préciser au moment de la création de la partition …

# cryptsetup --verbose --cipher=<Algorithme de chiffrement> --verify-passphrase luksFormat <Péripĥérique>  
# cryptsetup --verbose --cipher=aes-cbc-essiv:sha256 --verify-passphrase luksFormat /dev/sdb1

Le chiffrement aes-cbc-essiv est le chiffrement par défaut de cryptsetup pour les noyaux supérieurs au 2.6.10 car il corrige une vulnérabilité potentielle du chiffrement aes-cbc-plain.   
Une autre méthode de chiffrement utilisée avec l’algorithme AES (Rijndael) est le mode XTS, qui est réputé plus résistant aux attaques par watermarking, mais nécessite un module spécifique (judicieusement nommé xts) et une clef d’initialisation du double de la taille de la clef finale.

Les autres algorithmes dignes d’intérêt sont *twofish* et *serpent*, deux compétiteurs de l’AES face à Rijndael.

### Statistiques (*benchmark*)

On peut vérifier la capacité du processeur avec les différents algorithmes en utilisant la commande cryptsetup benchmark …

# cryptsetup benchmark

# Tests approximatifs en utilisant uniquement la mémoire (pas de stockage E/S).

PBKDF2-sha1 1144733 itérations par seconde pour une clé de 256 bits

PBKDF2-sha256 1444319 itérations par seconde pour une clé de 256 bits

PBKDF2-sha512 824352 itérations par seconde pour une clé de 256 bits

PBKDF2-ripemd160 627138 itérations par seconde pour une clé de 256 bits

PBKDF2-whirlpool 356173 itérations par seconde pour une clé de 256 bits

argon2i 4 itérations, 420103 mémoire, 4 threads parallèles (CPUs) pour une clé de 256 bits (temps de 2000 ms demandé)

argon2id 4 itérations, 406413 mémoire, 4 threads parallèles (CPUs) pour une clé de 256 bits (temps de 2000 ms demandé)

# Algorithme | Clé | Chiffrement | Déchiffrement

aes-cbc 128b 576,1 MiB/s 1804,4 MiB/s

serpent-cbc 128b 67,9 MiB/s 281,7 MiB/s

twofish-cbc 128b 153,2 MiB/s 301,7 MiB/s

aes-cbc 256b 421,1 MiB/s 1407,3 MiB/s

serpent-cbc 256b 73,1 MiB/s 281,9 MiB/s

twofish-cbc 256b 158,4 MiB/s 307,6 MiB/s

aes-xts 256b 1219,0 MiB/s 1245,8 MiB/s

serpent-xts 256b 291,4 MiB/s 280,5 MiB/s

twofish-xts 256b 299,9 MiB/s 303,5 MiB/s

aes-xts 512b 1056,2 MiB/s 1037,8 MiB/s

serpent-xts 512b 293,0 MiB/s 278,1 MiB/s

twofish-xts 512b 297,5 MiB/s 296,0 MiB/s

Cela permet aussi de choisir le chiffrement le plus performant, dans cet exemple aes-xts avec une clé de 256 bits (1056,2 MiB/s) est le plus performant.

### Sauvegarde de l’entête du conteneur

**Sauvegarde d’un entête**Étant donné que toute corruption, altération ou destruction de l’entête LUKS entraîne l’impossibilité d’accéder à l’ensemble du support chiffré, il convient d’en effectuer une sauvegarde.

On peut sauvegarder l’entête d’un conteneur LUKS …

# cryptsetup luksHeaderBackup --header-backup-file <Nom du fichier> <Périphérique>

# cryptsetup luksHeaderBackup --header-backup-file sauvegarde.entete /dev/sdb1

**Remarque** …  
Il faut renouveler la sauvegarde de l’entête en cas de modification des phrases secrètes.

**Vérification d’un entête**

En cas de tentative de restauration d’un entête, il est possible de vérifier que l’entête que l’on souhaite restaurer est le bon …

# cryptsetup -v --header ./sauvegarde.entete open /dev/sdb1 secret

Saisissez la phrase secrète pour /dev/sdb1 :

Emplacement de clé 0 déverrouillé.

Opération réussie.

**Remarque** …  
Le conteneur chiffré doit être fermé.

**Restauration d’un entête**Avec le conteneur fermé, on peut effectuer la restauration d’un entête à partir de son fichier de sauvegarde

Pour restaurer l’entête …

# cryptsetup luksHeaderRestore --header-backup-file <Nom du fichier> <Périphérique>

# cryptsetup luksHeaderRestore --header-backup-file sauvegarde.entete /dev/sdb1

**Remarque** …  
Si on obtient un message de ce type …  
Command failed: Failed to setup dm-crypt key mapping.

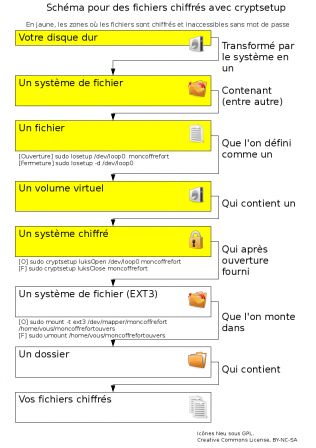
Check kernel for support for the aes-cbc-essiv:sha256 cipher spec and verify that /dev/sda1 contains at least 133 sectors

Cela signifie probablement que le module noyau Device Mapper n’est pas chargé.  
Pour y remédier …  
# modprobe dm-mod

## Zone protégée sur un disque dur

Il est possible de créer un système de fichier chiffré en utilisant un fichier sur un disque dur non chiffré.

Voici un schéma résumant les principales étapes …



**Étape 01  
Génération d’un fichier sur le disque dur**

Comme première étape, on va créer un fichier d’une taille de 50 Mo (avec des données aléatoires) …  
# dd if=/dev/zero bs=1M count=50 of=moncoffrefort

**Étape 02  
Création d’un périphérique**

Par la suite, il va falloir indiquer au noyau que ce fichier devrait être vu comme un périphérique disque dur, et qu’il peut travailler avec comme tel …  
# losetup /dev/loop0 moncoffrefort

Si par hasard un message d’erreur ressemblant à ioctl: LOOP\_SET\_FD: Périphérique ou ressource occupé, c’est que /dev/loop0 est déjà utilisé. Il suffit alors de modifier la valeur 0 en un nombre de 1 à 7.

Remarque …  
Il est possible d’utiliser plusieurs partitions chiffrées en même temps.  
Il faudra seulement faire attention de saisir correctement les prochaines commandes en changeant aussi leur numéros.

**Étape 03  
Initialisation de la partition chiffrée**

Comme pour initialiser un partition physique, il faut avoir recours à la commande cryptsetup …  
# cryptsetup luksFormat -c aes -h sha256 /dev/loop0  
WARNING!  
========  
Cette action écrasera définitivement les données sur /dev/loop1.  
  
Are you sure? (Type uppercase yes): YES  
Saisissez la phrase secrète pour /dev/loop1 :  
Vérifiez la phrase secrète :

Le programme demander d’abord de confirmer l’opération (qui est irréversible).   
Le programme va ensuite demander un mot de passe (*Enter LUKS passphrase*:).

**Étape 04  
Ouverture du volume chiffré**

Il faut par la suite ouvrir le nouveau volume chiffré …  
# cryptsetup luksOpen /dev/loop0 moncoffrefort  
Saisissez la phrase secrète pour /dev/loop1 :

**Remarque** …  
moncoffrefort est cette fois une étiquette.

**Étape 05  
Formatage du volume chiffré**

Après, il faut formater le volume …  
# mkfs.ext4 /dev/mapper/moncoffrefort  
mke2fs 1.44.5 (15-Dec-2018)

Creating filesystem with 9216 1k blocks and 2304 inodes

Filesystem UUID: 63388811-1152-4ebc-932f-6ff668e7705a

Superblock backups stored on blocks:

8193

Allocating group tables: done

Writing inode tables: done

Creating journal (1024 blocks): done

Writing superblocks and filesystem accounting information: done

**Pour la suite …**

Ouverture et montage d’un volume chiffré …  
# losetup /dev/loop0 moncoffrefort  
# cryptsetup luksOpen /dev/loop0 moncoffrefort  
# mkdir ~/ moncoffrefortouvert  
# mount -t ext4 /dev/mapper/moncoffrefort ~/moncoffrefortouvert

Fermeture et démontage d’un volume chiffré …  
# umount /home/vous/moncoffrefortouvert  
# cryptsetup luksClose moncoffrefort  
# losetup -d /dev/loop0

## Montage automatique

La plupart des distributions intègre la gestion des volumes chiffrée LUKS. Ceci permet de gérer de manière automatique le montage et le démontage des volumes (partitions) sécurisées.   
La configuration des paramètres du volume chiffré est dans le fichier **/etc/crypttab** et le montage du volume est de manière classique dans **/etc/fstab**.

Le fichier /etc/crypttab contient la liste des périphériques à déverrouiller automatiquement.   
Chaque ligne du fichier crypttab est de la forme …  
<Nom de mappage> <Périphérique source> <Fichier contenant la clef> <Options>

* <Nom du mappage>  
  Nom à donner au mappage (/dev/mapper/<Nom>), dans le cas présent secret ;
* <Périphérique source>  
  Identifiant du conteneur luks, sous la forme UUID= ;
* <Fichier contenant la clef>  
  Chemin absolu vers le ficher de phrase secrète.   
  Si le déverrouillage doit s'effectuer par saisie d'un mot de passe, indiquer none ;
* <Options>  
  Liste d'options séparées par des virgules, par exemple luks, discard pour un chiffrage luks et autoriser l'utilisation de la commane fstrim ou discard au niveau du container.   
  L'option keyscript= donne la possibilité d'exécuter un script ou une commande avec le chemin vers le fichier de phrase secrète (paramètre password précédent) fourni comme argument.

À chaque périphérique déclaré dans /etc/crypttab est associé une ligne dans /etc/fstab précisant les conditions de montage.   
Chaque ligne de fstab est de la forme …

<Système de fichier> <Point de montage> <Type> <Options> <Dump> <Passe>

* Système de fichier>  
  Nom du conteneur tel que dans crypttab ;
* <Point de montage><Point de montage> <Type> <Options> <Dump> <Passe> correspondant au système de fichier du conteneur (généralement ext4).

**Un exemple de chiffrement du dossier /home**

Il faut modifier le fichier /etc/crypttab …  
# <target name> <source device> <key file> <options>  
home /dev/sdb1 none luks

De plus, on doit modifier le fichier /etc/fstab pour le volume chiffré …  
# /etc/fstab: static file system information.  
#  
# <file system> <mount point> <type> <options> <dump> <pass>  
/dev/mapper/home /home ext4 defaults 0 1

La clé d'ouverture du volume chiffré sera demandée au démarrage du système si la partition est montée automatiquement à l’aide de /etc/fstab.

**Attention** …  
Si une ligne est déjà présente pour le montage de la partition qui héberge le système de fichier chiffré (/dev/sdbq dans l'exemple ci-dessus), il faut la commenter (en ajoutant un # comme premier caractère) afin d’éviter un message d'erreur au démarrage.  
Si le démarrage a lieu avec les **options quiet splash** supprimées, la demande de la phrase de décodification va passer inaperçue… Le démarrage se mettra à attendre la saisie de la clé. Le plus simple est d'appuyer sur la touche entrée afin qu'il demande à nouveau la question.

### Recommandations

Si le disque avait contenu des données en clair, il faudrait penser à effectuer des opérations supplémentaires.  
Même …

* si la table des partitions a été écrasée ;
* si des informations sur le chiffrement ont été écrites pour la partition /dev/sdb1 ;
* si une fois ouvert dans un mappage virtuel arbitrairement appelé /dev/mapper/part-chiffree et la nouvelle partition a été formatée en ext4 …

Il est possible que certaines des anciennes données (voire la plupart) sont malgré tout encore présentes sur le disque.

Voici trois solutions afin de remédier à cette situation …

* Écraser les données avant de mettre en place la partition chiffrée …  
  # dd if=/dev/urandom of=/dev/sdb
* Générer du bruit avant le formatage (soit entre l’étape 3 et l’étape 4) …  
  # dd if=/dev/zero of=/dev/mapper/part-chiffree
* Générer du bruit après le formatage (soit entre l’étape 4 et l’étape 5).  
  On suppose que la partition est montée dans /mnt …  
  # cat /dev/zero > /mnt/zeros; sync; rm /mnt/zero

Afin d’être certain que les données originales soient bien effacées, il vaudrait mieux faire trois passes …  
# dd if=/dev/zero of=/dev/sdb  
# dd if=/dev/urandom of=/dev/sdb  
# dd if=/dev/zero of=/dev/sdb

Les étapes suivantes montrent comment créer un nouveau système de fichiers chiffré à l'aide de la commande cryptsetup.

Pour commencer, il faudra peut-être charger certains modules du noyau …  
# modprobe dm-crypt  
# modprobe des  
ou  
# modprobe aesni-intel  
# modprobe sha256

Il faut ensuite créer un mot de passe au format LUKS sur une nouvelle partition.

Remarque …  
Si on utilise une partition existante, il est préférable de sauvegarder toutes les données et démonter la partition.   
La commande suivante remplace les données de la partition /dev/sdb1 …  
# cryptsetup --verbose --verify-passphrase luksFormat /dev/sdb1  
WARNING!  
========  
Cette action écrasera définitivement les données sur /dev/sdb1.  
  
Are you sure? (Type uppercase yes): YES  
Saisissez la phrase secrète pour /dev/sdb1 :  
Vérifiez la phrase secrète :

Emplacement de clef 0 créé.

Opération réussie.

Dans la sortie précédente de la commande **cryptsetup**, on est invité à fournir un mot de passe complexe. Cette phrase secrète sera nécessaire pour déchiffrer le système de fichiers chaque fois que l’on monter le système de fichiers.

Ensuite, on doit exécuter une commande pour créer un nouveau fichier de périphérique **crypt**.   
Ce qui suit crée un fichier appelé **/dev/mapper/data**, qui sera utilisé ultérieurement pour monter le système de fichiers chiffré.   
La phrase secrète que l’on saisit doit correspondre à celle créée par la commande **cryptsetup** précédente.  
# cryptsetup luksOpen /dev/sdb1 data  
Saisissez la phrase secrète pour /dev/sdb1 :

Maintenant que le système de fichiers est ouvert (on peut considérer cela comme déverrouillé ou déchiffré), il faut créer un système de fichiers sur le nouveau périphérique …  
# mkfs -t ext4 /dev/mapper/data  
mke2fs 1.44.5 (15-Dec-2018)  
En train de créer un système de fichiers avec 5238528 4k blocs et 1310720 i-noeuds.  
UUID de système de fichiers=5a373743-4b56-4d8f-9a25-f9ac4d4337b5  
Superblocs de secours stockés sur les blocs :  
 32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,  
 4096000  
  
Allocation des tables de groupe : complété  
Écriture des tables d'i-noeuds : complété  
Création du journal (32768 blocs) : complété  
Écriture des superblocs et de l'information de comptabilité du système de  
fichiers : complété

**Très important** …  
Dans la commande précédente que le périphérique /dev/mapper/data a été utilisé comme périphérique pour créer le système de fichiers. NE PAS créer le système de fichiers sur la partition (/dev/sdb1 dans cet exemple).

Il faut enfin créer le répertoire du point de montage et les entrées dans les fichiers **/etc/crypttab** et **/etc/fstab**.   
Le fichier **/etc/crypttab** correspond au nouveau périphérique **/dev/mapper/data** avec la partition **/dev/sdb1**.   
Cette association est utilisée par le système pendant le processus de démarrage pour monter /dev/mapper/data dans le répertoire /data.

# mkdir /data  
# echo "data /dev/sdb1 none" >> /etc/crypttab  
# echo "/dev/mapper/data /data ext4 defaults 1 2" >> /etc/fstab

À partir de maintenant, à chaque démarrage du système, ce dernier invitera à entrer le mot de passe complexe pour monter le périphérique **/dev/mapper/data**.

Il est possible d'automatiser ce processus en ne modifiant rien dans le fichier **/etc/crypttab** en un paramètre fournissant le mot de passe, mais en supprimant la raison pour laquelle on a chiffré le système de fichiers.

### Première méthode de chiffrement alternatif

Ceci est un exemple pour configurer un système de fichiers chiffé.   
Tous les modules pertinents doivent être chargés au démarrage …  
# echo aesni-intel >> /etc/modules  
# echo dm\_mod >> /etc/modules  
# echo dm\_crypt >> /etc/modules  
# modprobe -a aesni-intel dm\_mod dm\_crypt

On crée par la suite le périphérique mapper …  
# cryptsetup -y create crypt /dev/sdc1

Il faudra alors saisir une phrase secrète.

Les prochaines étapes sont …  
# echo "crypt /dev/sdc1 none none" >> /etc/crypttab

# echo "/dev/mapper/crypt /crypt reiserfs defaults 0 1" >> /etc/fstab  
# mkfs -t reiserfs /dev/mapper/crypt  
mkfs.reiserfs 3.6.25  
  
Guessing about desired format.. Kernel 4.9.0-9-amd64 is running.  
Format 3.6 with standard journal  
Count of blocks on the device: 5242624  
Number of blocks consumed by mkreiserfs formatting process: 8371  
Blocksize: 4096  
Hash function used to sort names: "r5"  
Journal Size 8193 blocks (first block 18)  
Journal Max transaction length 1024  
inode generation number: 0  
UUID: 1ab714cc-a572-4b90-9e12-7fc365995420  
ATTENTION: YOU SHOULD REBOOT AFTER FDISK!  
ALL DATA WILL BE LOST ON '/dev/mapper/crypt'!  
Continue (y/n):y  
Initializing journal - 0%....20%....40%....60%....80%....100%

Syncing..ok

ReiserFS is successfully created on /dev/mapper/crypt.

**Remarque** …  
Il est possible que reiserfs ne soit pas installé sur le système.   
il faut alors installer le paquet **libguestfs-reiserfs**.

Si nécessaire, comme dernière étape il faut monte le système de fichiers crypté.   
Le mot de passe que l’on a choisi avec **cryptsetup** sera demandé.   
Il sera demandé à chaque démarrage.

# Annexe 01 Chiffrement du répertoire /home

De nombreuses distributions proposent le chiffrement du répertoire personnel à l'installation.  
Toutefois, si on omet le chiffrement lors de la mise en place du système d’exploitation, la façon la plus simple de chiffrer serait de sauvegarder ses données et de réinstaller. Certaine distribution permette de chiffrer son répertoire personnel (avec ecryptfs, moins sûr que luks car les métadonnées du système de fichiers sont accessibles). De plus, certaines distributions sur support externe (cédérom ou clé USB) permettent de chiffrer des partitions (y compris système) avec luks et des volumes logiques (lvm).

Pour commencer, on devra posséder suffisamment d’espace disque pour une nouvelle partition destinée au chiffrement car la procédure détruit totalement les données déjà présentes.   
Avant toute chose, il est prudent d’effectuer une sauvegarde de ses données.   
Il faudra enfin disposer au moins d'une partition système, de partitions swap et /home séparées.

Voici l'hypothèse de départ concernant le partitionnement du disque …

=========================================================================  
| B | R | | S |  
| O | O | /HOME | W |  
| O | O | | A |  
| T | T | | P |  
=========================================================================

sda1 sda2 sda3 sda4

Les partitions peuvent être primaires ou logiques, simples ou être des volumes RAID, cela n'a pas d'importance.   
Si on utilise les volumes logiques (lvm), il est recommandé les mettre en place après création des conteneurs chiffrés, et non l'inverse.

Si on ne souhaite chiffrer que son répertoire personnel sur /home et la partition d'échange swap, une partition /boot séparée n’est pas nécessaire. Toutefois, celle-ci est indispensable si on chiffre la partition système.

**Remarque** …  
il est possible de travailler depuis un système en mode single user (mode de récupération ou init 1, ligne de commande exclusivement), ou depuis un cédérom ou clé USB (environnement graphique disponible, connexion Internet, …).   
Dans tous les cas on doit avoir installé cryptsetup, libpam-mount et, accessoirement, mdadm si les partitions sont situées sur des volumes raid ou encore lvm2 si on souhaite utiliser des volumes logiques.   
Ces paquets doivent être présents sur le système d'origine, ils seront nécessaires au redémarrage, et fournissent des fichiers de configuration qu’il faudra éditer.   
Seul cryptsetup doit également être présent sur le système de travail (cédérom ou clé USB) sinon on n'aura tout simplement pas accès aux commandes nécessaires.

# Annexe 02 Chiffrement de la partition de pagination (*swap*)

Afin de chiffrer la partition de pagination, il existe plusieurs solutions. Celle qui sera adoptée dépendra de son usage de la veille sur disque (hibernation). La mise en veille en mémoire vive est toujours possible, mais elle peut compromettre la sécurité de son système si un attaquant a accès à l'ordinateur alors qu'il est en veille.

Génération d'une clé de chiffrement aléatoire à chaque démarrage

La première solution est la plus sûre, mais elle peut allonger le temps de démarrage et interdit l'hibernation (la mise en veille en mémoire vive).

Pour l'utiliser, il suffit d'ajouter une ligne au fichier /etc/crypttab …  
# swap cryptée  
cryptswap /dev/disk/by-uuid/xxxx-xxx-xxxx-xxxx /dev/urandom swap

Il suffit de remplacer les xxxx par l'UUID de la partition qui reçoit l'espace d'échange swap.   
cryptswap est le nom du mappage qui sera fait pour la partition de pagination,. On peut choisir un autre nom.  
Pour cet exemple, la partition de pagination sera disponible sur /dev/mapper/cryptswap.

Dans un second temps, on doit modifier le fichier /etc/fstab …  
# swap chiffré  
/dev/mapper/cryptswap none swap sw 0 0

Il faudra ajuster l'adresse de la partition en fonction du nom choisit pour le mappage dans /etc/crypttab.   
Une alternative est l’utilisation de l'UUID de la partition de pagination mais il faut prêter une attention particulière à bien utiliser l'identifiant de la partition de pagination une fois le conteneur ouvert et no celle de la partition physique qui supporte le conteneur …  
# swap chiffré  
UUID=xxxxx-xxxxxx-xxxxxx-xxxxx-xxxxx none swap sw 0 0

Pour vérifier que tout fonctionne …  
# invoke-rc.d cryptdisks restart  
# swapon -s

Pour plus d’information …  
<https://www.linuxpedia.fr/doku.php/expert/systeme_chiffre_luks_pam_cryptsetup>

# Annexe 03 Chiffrement de la partition système

Voir la section Chiffrer la partition système du site …  
<https://www.linuxpedia.fr/doku.php/expert/systeme_chiffre_luks_pam_cryptsetup>

# Annexe 04 Utilisation de pam-mount

Pour éviter d'avoir à fournir un mot de passe …

* un premier pour déchiffrer le conteneur /home pendant le démarrage  
  et
* un second lors du login

il est possible de coupler les deux avec pam-mount.

Pour laisser pam-mount gérer le déchiffrage et montage de la partition /home on doit retirer ou commenter la ligne du fichier /etc/fstab qui y fait référence, ainsi que la ligne du fichier /etc/crypttab si on en a déjà ajouté une.

Concrètement ça donne pour le /etc/fstab …  
# /dev/mapper/testcrypt /home ext4 defaults,relatime 0 2  
On identifie la ligne qui devrait figurer pour une utilisation de luks traditionnelle, et le signe dièse # en début de ligne qui la désactive.

Il faut faire de même pour /etc/crypttab …  
# testcrypt UUID=1e1688df-72bc-45d1-824a-59d5f35g20v none luks,tries=2,timeout=10  
La ligne du fichier crypttab est commentée, cryptsetup ne prendra pas en charge ce conteneur au démarrage.

Par la suite, il faudra modifier la configuration PAM en éditant le fichier /etc/security/pam\_mount.conf.xml,   
On ajoute, en fin de fichier, juste avant le </pam\_mount> final …  
<volume user=<Nom d’utilisateur> fstype="crypt" path=</dev/disk/by-uuid/1e1688df-72bc-45d1-824a-59d5f35g20v> mountpoint="/home" options="fsck,relatime" />

On replacera <Nom d’utilisateur> par son nom de connexion, et le chemin /dev/disk/by-uuid/ devra comporter l'UUID correcte de la partition chiffrée /home. L'option de montage “relatime” est facultative.

Il faudra par la suite redémarrer le système, et si tout est bien en place, le démarrage devrait se dérouler sans aucun arrêt, le déchiffrement à la connexion sera totalement transparent.   
Si on rencontre des problèmes, il faudra vérifier soigneusement l'adresse de la partition chiffrée, Il est préférable d’utiliser les UUID dans les fichiers de configuration.

# Annexe 05 Déverrouillage au démarrage

Lors du démarrage, pour chaque périphérique présent dans le fichier /etc/crypttab, l'action de déverrouillage indiquée est effectuée, puis le périphérique est monté conformément aux informations du fichier /etc/fstab.

**Remarque** …  
En l'absence du périphérique, ou en cas d'erreur de mot de passe ou phrase secrète, le démarrage sera interrompu.

### Configurations

Trois situations sont possibles …

* le déverrouillage par saisie de mot de passe ;
* le déverrouillage par fichier de phrase secrète, stocké dans le système ;
* le déverrouillage par fichier de phrase secrète, stocké dans l'initramfs

Ces situations sont applicables respectivement …

* au déverrouillage de la partition racine (/) lorsque la partition /boot n'est pas chiffrée ;
* au déverrouillage d'une partition /home, /swap, /boot, ... lorsque la partition racine est chiffrée ;
* au déverrouillage d'une partition racine, lorsque la partition /boot est chiffrée.

### Déverrouillage par saisie de mot de passe

**Modification des fichiers de configuration**

Dans un premier temps, il faut créer une ligne dans les fichiers /etc/crypttab et /etc/fstab sur les modèles suivants …

# cat /etc/crypttab

# <target name> <source device> <key file> <options>

secret UUID=de53f0d1-e643-4782-bd8d-0655393f3efe none luks

**Remarque** …  
L'UUID de la partition peut être obtenu directement avec …

# blkid /dev/sdb1 -s UUID -o value

8fbafdc5-8704-4ecb-8bfe-7088d3a3af05

# cat /etc/fstab

/dev/mapper/secret /data ext4 defaults 0 0

**Vérification**

Démonter et verrouiller le périphérique et vérifier le déverrouillage selon la configuration définie dans/etc/crypttab …

# cryptdisks\_start secret

[....] Starting crypto disk...[info] secret (starting)...

Please unlock disk secret: \*\*\*\*

[ ok t (started)...done.

Vérifier le montage selon la configuration définie dans /etc/fstab …

# mount /dev/mapper/secret

Démonter et verrouiller le périphérique …

# umount /data

# cryptdisks\_stop secret

[ ok ] Stopping crypto disk...secret (stopping)...done.

**Utilisation**

Au démarrage, le mot de passe sera demandé pour chaque périphérique présent dans /etc/crypttab.   
Une fois le mot de passe saisi, le périphérique sera automatiquement déverrouillé et monté.

### Déverrouillage par phrase secrète

**Note de sécurité** …  
Dans cette configuration, la phrase secrète est stockée dans l'arborescence du système.   
Pour qu'elle ne soit pas accessible lorsque le système est à l'arrêt, la partition système doit donc impérativement être chiffrée.

**Configuration**

Dans ce qui suit, le fichier de phrase secrète sera stocké en /etc/clefs/phrasesecrete

Créer le fichier de phrase de pass, et restreindre les droits de lecture et d'écriture :

# mkdir /etc/clefs

# dd if=/dev/urandom of=/etc/clefs/phrasesecrete bs=512 count=4

# chmod 600 /etc/clefs/phrasesecretE

Ajouter dans un des slots de la partition chiffrée …  
# cryptsetup luksAddKey /dev/sdb1 /etc/clefs/phrasesecrete

Entrez une phrase secrète existante :

Remplacer, dans le fichier /etc/crypttab, none par le chemin absolu du fichier …  
# cat /etc/crypttab

# <target name> <source device> <key file> <options>

secret UUID=8fbafdc5-8704-4ecb-8bfe-7088d3a3af05 /etc/clefs/phrasesecrete luks

La ligne du fichier /etc/fstab est inchangée …

3 cat /etc/fstab

/dev/mapper/secret /data ext4 defaults 0 0

**Vérification**

Vérifier l'ouverture automatique du périphérique ….

# cryptdisks\_start secret

**Utilisation**

Au démarrage, le périphérique est automatiquement déverrouillé et monté.   
Un message confirme le bon déverrouillage.

Remarque …  
Si le périphérique est démonté, verrouillé puis déconnecté lorsque le système est en fonctionnement, il sera automatiquement déverrouillé et monté s'il est reconnecté.

### Déverrouillage depuis l'initramfs

**Note de sécurité** …  
Dans cette configuration, la phrase secrèete est stockée dans l'arborescence du système et dans l'initramfs.   
Pour qu'elle ne soit pas accessible lorsque le système est à l'arrêt, la partition système et la partition dans laquelle est située l'initramfs doivent impérativement être chiffrées.

**Configuration**

Créer une phrase secrète comme dans la configuration précédente et l'ajouter dans un des ouvertures (*slots*) de la partition chiffrée.

Créer le fichier de phrase secrète et restreindre les droits d'accès …

# mkdir /etc/clefs

# dd if=/dev/urandom of=/etc/clefs/phrasesecrete bs=512 count=4

# chmod 600 /etc/clefs/phrasesecrete

Ajouter la phrase de passe dans un des slots de la partition chiffrée …

# cryptsetup luksAddKey /dev/sdb1 /etc/clefs/phrasesecrete

Entrez une phrase secrète existante :

Créer le script qui permettra d'ajouter la phrase secrète à la racine de l'initramfs …

# cat > /etc/initramfs-tools/hooks/addkey <<EOF

#!/bin/sh

cp /etc/clefs/ phrasesecrete "\${DESTDIR}"

EOF

R

endre le script exécutable

# chmod +x /etc/initramfs-tools/hooks/addkey

Restreindre à root seulement, l'accès en lecture à initramfs …

# echo "UMASK=026" >>/etc/initramfs-tools/initramfs.conf

Remplacer, dans le fichier /etc/crypttab, none par le chemin du fichier de phrase secrète dans l'initramfs.

Ajouter l'option keyscript=/bin/cat qui permettra de lire la phrase secrète depuis l'initramfs …

# cat /etc/cryptab

# <target name> <source device> <key file> <options>

secret UUID=8fbafdc5-8704-4ecb-8bfe-7088d3a3af05 /phrasesecrete luks,keyscript=/bin/cat

Mettre à jour l'initramfs …

# update-initramfs -u -k all

On peut vérifier que le fichier phrasesecrete est présent à la racine de l'initramfs …

# lsinitramfs /boot/initrd.img-4.9.0-3-amd64 |grep phrasesecrete

secret\_phrase

La ligne du fichier /etc/fstab est inchangée

# cat /etc/fstab

/dev/mapper/secret /mnt ext4 defaults 0 0

**Utilisation**

Au démarrage, le périphérique est automatiquement déverrouillé et monté.   
Un message confirme le bon déverrouillage.

# Annexe 06 Création d’une clé USB chiffrée via LUKS avec cryptsetup

### Présentation

À l'ère de l’infonuagique et de la dématérialisation des données, certaines de nos informations personnelles ont tout intérêt à rester très proche. Et il est même préférable de prévoir le jour où elles en sortiront par inadvertance.

Ce tutoriel va créer une clé USB chiffrée via le système de chiffrement de disque LUKS (*Linux Unified Key Setup*).

Il serait bien entendu possible de ne chiffrement qu'une partition et non la clé entièrement, mais on va ici créer une partition couvrant l'intégralité d'une clé USB (tutoriel fait avec un clé USB de 16 Go).

|  |
| --- |
| Dans ce qui suit, on utilise les conventions suivantes …   * Identifiant de la partition physique du container chiffré luks … **/dev/sdb1** * Nom du mappage … **secret** * Point de montage … **/data** |

### Installation de cryptsetup

La première étape est d’installer cryptsetup (si cela n’est déjà pas fait).  
Pour la distribution Debian …  
# apt install cryptsetup

Pour la distribution CentOS …  
# yum install cryptsetup-luks

### Création de la partition LUKS et du système de fichier

C'est le moment d'insérer la clé USB (si cela n'est pas déjà fait).

**Remarque** …  
Dans le cadre du tutoriel, la clé USB est chiffrée dans sa totalité et non pas seulement une partition.   
Il est également important de savoir que durant cette procédure, toutes les données existantes de la clé seront effacées.

Pour chiffrer des données qui sont déjà sur la clé …

* Copier les données de la clé vers un répertoire sur un disque dur ;
* Procéder à la création de la partition chiffrée comme cela va être expliqué à la suite de ce tutoriel ;
* Recopier ses données du disque dur vers votre nouvelle partition chiffrée ;
* Supprimer les fichiers mis temporairement sur lee disque dur.

La prochaine étape vise à repérer la clé USB dans la liste des périphériques.  
# cryptsetup -y -v luksFormat /dev/sdb1

On va, comme prochaine étape, procéder au chiffrement.  
Une phrase secrète sera alors demandée, élément permettant d'ouvrir la partition chiffrée.

On pourra ensuite ouvrir la partition LUKS, qui ne sera pas encore tout à faire utilisable.  
La phrase nous sera demandée pour ouvrir notre partition LUKS …  
# cryptsetup luksOpen /dev/sdb1 usb1

On formate alors notre partition pour la rendre toute propre en la remplissant de 0 (du vide si l'on peut dire) …  
# dd if=/dev/zero of=/dev/mapper/usb1

En fonction de la clé, cela peut être assez long (en fonction de la aille et de la puissance du processeur), on peut également utilise la commande pv (non installée par défaut sur la plupart des distributions) qui affiche une petite barre de progression …  
# pv -tpreb /dev/zero | dd of=/dev/mapper/usb1 bs=128M

Afin que la clé USB soit utilisable et que l'on puisse y stocker des fichiers par-dessus LUKS, on créé un système de fichier sur la partition …  
# mkfs.ext4 /dev/mapper/usb1

C'est une partie qui peut sembler déroutante au premier abord, une partition LUKS (qui chiffre tout) et par-dessus un système de fichier ext4 (ou autre si souhaité) qui supportera les fichiers/répertoires. Lorsque l'on utilisera la clé USB, il faudra donc déchiffrer l'ensemble dans un premier temps en saisissant la phrase secrète, puis le syste d’exploitaion sera capable de voir le système de fichier utilisé et pourra le monter pour y lire fichiers et dossiers.

Ici, on précise comme emplacement cible /dev/mappe/usbkey1, pourquoi ?

* /dev/mapper/usb1 est le raccourci qui a été donné à la clé USB (/dev/mapper est générique à Linux pour le mappage, usb1 dépend de ce que l’on a saisi). Il permet donc d'accéder à la partition déchiffrée ;
* /dev/sdb1 est la partition de la clé USB.  
  Ici les informations sont chiffrées grâce à LUKS et un système d’exploitation ne peut monter un système de fichier connu directement ;
* /dev/mapper/\*\*\*\*est le nom du mappage de la partition /dev/sdb1 une fois déchiffré.  
  Ici le contenu est lisible pour un système d’exploitation/système de fichier.

Étant donné que le chiffrement utilisé est un *block-layer encryption* et non un *file-system encryption*, cette étape supplémentaire de mappage est nécessaire.

### Montage et demontage

Avec la distribution Debian, dm-crypt est présent par défaut. Le montage se fait donc dans la plupart des cas directement au branchement de la clé USB. Alors, notre phrase secrète est directement demandée.

Une fois la phrase secrète saisie, le système d’exploitation pourra lire le système de fichier se trouvant sur la clé et pourra ensuite monter le système de fichier, on verra alors la partition montée.

Toutefois, l'interface graphique n'est pas toujours là et dans ce cas travailler en ligne de commande. À nouveau, il faut donc brancher la clé USB et repérer la partition apropriée. La clé USB étant représentée par /dev/sdb (/dev/sda étant le disque dur principal ou la partition racine).

On commence donc par déchiffrer la partition chiffrée à l'aide de crypsetup et de son option luksOpen …  
# cryptsetup luksOpen /dev/sdb usb1

Ensuite, et maintenant que la clé USB est lisible par le système de fichier et le système d’exploitation , on monte son contenu dans un dossier quelconque, par exemple /mnt/usbdata …  
# mount /dev/mapper/usb1 /mnt/usbdata

On pourra donc accéder aux données de la clé USB librement et de façon transparente.

Une fois les opérations terminées, il ne faut pas oublier de fermer tout cela proprement et ne surtout pas arracher la clé USB brusquement.

On commence par démonter le montage du système de fichier effectué …  
# umount /mnt/usbdate

Puis on referme la partition chiffrée …  
# cryptsetup luksClose usb1

On peut maintenant débrancher la clé USB.

**Remarque** …  
Avec une interface graphique, le fait d'éjecter le périphérique effectue ces deux actions simultanément.

## Références

<https://www.linuxpedia.fr/doku.php/expert/systeme_chiffre_luks_pam_cryptsetup>

<https://papy-tux.legtux.org/doc1002/index.html>





# Annexe 01 Manuel de cryptsetup

cryptsetup – Gestionnaire de volumes simples chiffrés dm-crypt et LUKS.

## Synopsis

cryptsetup <options> <action> <arguments pour les actions>

## Description

cryptsetup est utilisé pour configurer facilement les liens de mappeurs de périphériques gérés par dm-crypt.   
Ceux-ci incluent les volumes brut (*plain*) dm-crypt et les volumes LUKS.   
La différence réside dans le fait que LUKS utilise un en-tête de métadonnées et peut donc offrir davantage de fonctionnalités qu’un volume brut dm-crypt.   
Toutefois, l'en-tête LUKSest visible et susceptible d'être endommagé.

En outre, cryptsetup fournit une prise en charge limitée de l’utilisation des volumes AES en boucle et des volumes compatibles TrueCrypt.

### dm-crypt brut ou luks?

Sauf si on comprend bien le contexte cryptographique, l’utilisation de LUKS est suggérée. Plain dm-crypt permet à un certain nombre d'erreurs utilisateur de réduire considérablement la sécurité. Bien que LUKS ne puisse pas tous les résoudre, cela peut en atténuer l’impact pour nombre d’entre eux.

### Avertissements

La FAQ de cryptsetup contient de nombreuses informations utiles sur les risques liés à l'utilisation du stockage chiffré, aux problèmes de traitement et aux aspects de sécurité. Il est important de la lire.  
Néanmoins, certains risques méritent d’être mentionnés ici.

* **Sauvegarde: matrice de support de stockage**  
  Le chiffrement n'a aucune influence sur cela.   
  La sauvegarde est également obligatoire pour les données chiffrées, si celles-ci ont une valeur.
* **Encodage de caractères**  
  Si on saisit une phrase secrète avec des symboles spéciaux, celle-ci peut changer en fonction de l'encodage des caractères.   
  Les paramètres du clavier peuvent également changer, ce qui peut rendre la saisie aveugle difficile, voire impossible.  
  Par exemple, le passage d'une variante ASCII à 8 bits à UTF-8 peut entraîner un codage binaire différent et donc une phrase secrète différente interprétée par cryptsetup, et ce même si ce que l’on voit sur le terminal est exactement identique.  
  Il est donc vivement recommandé de sélectionner les caractères de phrase secrète uniquement en ASCII 7 bits, car le codage pour l’ASCII 7 bits reste le même pour toutes les variantes ASCII et UTF-8.
* **En-tête LUKS**  
  Si l'en-tête d'un volume LUKS est endommagé, toutes les données sont définitivement perdues, sauf si on fait une sauvegarde d'en-tête. Si un emplacement de clé est endommagé, il ne peut être restauré qu'à partir d'une sauvegarde d'en-tête ou si un autre emplacement de clé actif avec une phrase secrète connue n'est pas endommagé.   
  Endommager l'en-tête de LUKS est quelque chose que les gens réussissent à faire avec une fréquence surprenante. Ce risque résulte d'un compromis entre sécurité et sûreté, car LUKS est conçu pour un effacement rapide et sécurisé en écrasant simplement l'en-tête et la zone des emplacements de clés.
* **Partitions précédemment utilisées**  
  Si une partition a déjà été utilisée, il est très judicieux d’effacer les signatures, les données, ... du système de fichiers avant de créer un conteneur LUKS ou un simple conteneur dm-crypt.   
  Pour supprimer rapidement les signatures du système de fichiers, on peut utilise la commande wipefs.  
  Il est important de veiller toutefois à ce que cela ne supprime pas tout. En particulier, les signatures RAID MD à la fin d'un périphérique peuvent survivre. Cela ne supprime pas non plus les données.   
  Pour un effacement complet, il est préférable d’écrase toute la partition avant la création du conteneur. Si vous ne savez pas comment faire cela, la FAQ de cryptsetup décrit plusieurs options.BASIC COMMANDS

### Commandes de base

The following are valid actions for all supported device types.

open <device> <name> --type <device\_type>

Opens (creates a mapping with) <name> backed by device <device>.

Device type can be plain, luks (default), luks1, luks2, loopaes or tcrypt.

For backward compatibility there are open command aliases:

create (argument-order <name> <device>): open --type plain

plainOpen: open --type plain

luksOpen: open --type luks

loopaesOpen: open --type loopaes

tcryptOpen: open --type tcrypt

<options> are type specific and are described below for individ‐

ual device types. For create, the order of the <name> and <de‐

vice> options is inverted for historical reasons, all other

aliases use the standard <device> <name> order.

close <name>

Removes the existing mapping <name> and wipes the key from ker‐

nel memory.

For backward compatibility there are close command aliases: re‐

move, plainClose, luksClose, loopaesClose, tcryptClose (all be‐

haves exactly the same, device type is determined automatically

from active device).

<options> can be [--deferred]

status <name>

Reports the status for the mapping <name>.

resize <name>

Resizes an active mapping <name>.

If --size (in 512-bytes sectors) is not specified, the size is

computed from the underlying device. For LUKS it is the size of

the underlying device without the area reserved for LUKS header

(see data payload offset in luksDump command). For plain crypt

device, the whole device size is used.

Note that this does not change the raw device geometry, it just

changes how many sectors of the raw device are represented in

the mapped device.

If cryptsetup detected volume key for active device loaded in

kernel keyring service, resize action would first try to re‐

trieve the key using a token and only if it failed it'd ask for

a passphrase to unlock a keyslot (LUKS) or to derive a volume

key again (plain mode). The kernel keyring is used by default

for LUKS2 devices.

With LUKS2 device additional <options> can be [--token-id, --to‐

ken-only, --key-slot, --key-file, --keyfile-size, --keyfile-off‐

set, --timeout, --disable-locks, --disable-keyring].

refresh <name>

Refreshes parameters of active mapping <name>.

Updates parameters of active device <name> without need to deac‐

tivate the device (and umount filesystem). Currently it supports

parameters refresh on following devices: LUKS1, LUKS2 (including

authenticated encryption), plain crypt and loopaes.

Mandatory parametrs are identical to those of an open action for

respective device type.

You may change following parameters on all devices

--perf-same\_cpu\_crypt, --perf-submit\_from\_crypt\_cpus and --al‐

low-discards.

Refreshing device without any optional parameter will refresh

the device with default setting (respective to device type).

LUKS2 only:

--integrity-no-journal parameter affects only LUKS2 devices with

underlying dm-integrity device.

Adding option --persistent stores any combination of device pa‐

rameters above in LUKS2 metadata (only after successful refresh

operation).

--disable-keyring parameter refreshes a device with volume key

passed in dm-crypt driver.

PLAIN MODE

Plain dm-crypt encrypts the device sector-by-sector with a single, non-

salted hash of the passphrase. No checks are performed, no metadata is

used. There is no formatting operation. When the raw device is mapped

(opened), the usual device operations can be used on the mapped device,

including filesystem creation. Mapped devices usually reside in

/dev/mapper/<name>.

The following are valid plain device type actions:

open --type plain <device> <name>

create <name> <device> (OBSOLETE syntax)

Opens (creates a mapping with) <name> backed by device <device>.

<options> can be [--hash, --cipher, --verify-passphrase, --sec‐

tor-size, --key-file, --keyfile-offset, --key-size, --offset,

--skip, --size, --readonly, --shared, --allow-discards, --re‐

fresh]

Example: 'cryptsetup open --type plain /dev/sda10 e1' maps the

raw encrypted device /dev/sda10 to the mapped (decrypted) device

/dev/mapper/e1, which can then be mounted, fsck-ed or have a

filesystem created on it.

LUKS EXTENSION

LUKS, the Linux Unified Key Setup, is a standard for disk encryption.

It adds a standardized header at the start of the device, a key-slot

area directly behind the header and the bulk data area behind that. The

whole set is called a 'LUKS container'. The device that a LUKS con‐

tainer resides on is called a 'LUKS device'. For most purposes, both

terms can be used interchangeably. But note that when the LUKS header

is at a nonzero offset in a device, then the device is not a LUKS de‐

vice anymore, but has a LUKS container stored in it at an offset.

LUKS can manage multiple passphrases that can be individually revoked

or changed and that can be securely scrubbed from persistent media due

to the use of anti-forensic stripes. Passphrases are protected against

brute-force and dictionary attacks by PBKDF2, which implements hash it‐

eration and salting in one function.

LUKS2 is a new version of header format that allows additional exten‐

sions like different PBKDF algorithm or authenticated encryption. You

can format device with LUKS2 header if you specify --type luks2 in

luksFormat command. For activation, the format is already recognized

automatically.

Each passphrase, also called a key in this document, is associated with

one of 8 key-slots. Key operations that do not specify a slot affect

the first slot that matches the supplied passphrase or the first empty

slot if a new passphrase is added.

The <device> parameter can also be specified by a LUKS UUID in the for‐

mat UUID=<uuid>. Translation to real device name uses symlinks in

/dev/disk/by-uuid directory.

To specify a detached header, the --header parameter can be used in all

LUKS commands and always takes precedence over the positional <device>

parameter.

The following are valid LUKS actions:

luksFormat <device> [<key file>]

Initializes a LUKS partition and sets the initial passphrase

(for key-slot 0), either via prompting or via <key file>. Note

that if the second argument is present, then the passphrase is

taken from the file given there, without the need to use the

--key-file option. Also note that for both forms of reading the

passphrase from a file you can give '-' as file name, which re‐

sults in the passphrase being read from stdin and the safety-

question being skipped.

You can only call luksFormat on a LUKS device that is not

mapped.

To use LUKS2, specify --type luks2.

<options> can be [--hash, --cipher, --verify-passphrase,

--key-size, --key-slot, --key-file (takes precedence over op‐

tional second argument), --keyfile-offset, --keyfile-size,

--use-random | --use-urandom, --uuid, --master-key-file,

--iter-time, --header, --pbkdf-force-iterations, --force-pass‐

word, --disable-locks].

For LUKS2, additional <options> can be [--integrity, --integ‐

rity-no-wipe, --sector-size, --label, --subsystem, --pbkdf,

--pbkdf-memory, --pbkdf-parallel, --disable-locks, --dis‐

able-keyring, --luks2-metadata-size, --luks2-keyslots-size,

--keyslot-cipher, --keyslot-key-size].

WARNING: Doing a luksFormat on an existing LUKS container will

make all data the old container permanently irretrievable unless

you have a header backup.

open --type luks <device> <name>

luksOpen <device> <name> (old syntax)

Opens the LUKS device <device> and sets up a mapping <name> af‐

ter successful verification of the supplied passphrase.

First, the passphrase is searched in LUKS tokens. If it's not

found in any token and also the passphrase is not supplied via

--key-file, the command prompts for it interactively.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--readonly, --test-passphrase, --allow-discards, --header,

--key-slot, --master-key-file, --token-id, --token-only, --dis‐

able-keyring, --disable-locks, --type, --refresh].

luksSuspend <name>

Suspends an active device (all IO operations will block and ac‐

cesses to the device will wait indefinitely) and wipes the en‐

cryption key from kernel memory. Needs kernel 2.6.19 or later.

After this operation you have to use luksResume to reinstate the

encryption key and unblock the device or close to remove the

mapped device.

WARNING: never suspend the device on which the cryptsetup binary

resides.

<options> can be [--header, --disable-locks].

luksResume <name>

Resumes a suspended device and reinstates the encryption key.

Prompts interactively for a passphrase if --key-file is not

given.

<options> can be [--key-file, --keyfile-size, --header, --dis‐

able-keyring, --disable-locks, --type]

luksAddKey <device> [<key file with new key>]

Adds a new passphrase. An existing passphrase must be supplied

interactively or via --key-file. The new passphrase to be added

can be specified interactively or read from the file given as

positional argument.

NOTE: with --unbound option the action creates new unbound LUKS2

keyslot. The keyslot cannot be used for device activation. If

you don't pass new key via --master-key-file option, new random

key is generated. Existing passphrase for any active keyslot is

not required.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--new-keyfile-offset, --new-keyfile-size, --key-slot, --mas‐

ter-key-file, --iter-time, --force-password, --header, --dis‐

able-locks, --unbound, --type, --keyslot-cipher,

--keyslot-key-size].

luksRemoveKey <device> [<key file with passphrase to be removed>]

Removes the supplied passphrase from the LUKS device. The

passphrase to be removed can be specified interactively, as the

positional argument or via --key-file.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--header, --disable-locks, --type]

WARNING: If you read the passphrase from stdin (without further

argument or with '-' as an argument to --key-file), batch-mode

(-q) will be implicitly switched on and no warning will be given

when you remove the last remaining passphrase from a LUKS con‐

tainer. Removing the last passphrase makes the LUKS container

permanently inaccessible.

luksChangeKey <device> [<new key file>]

Changes an existing passphrase. The passphrase to be changed

must be supplied interactively or via --key-file. The new

passphrase can be supplied interactively or in a file given as

positional argument.

If a key-slot is specified (via --key-slot), the passphrase for

that key-slot must be given and the new passphrase will over‐

write the specified key-slot. If no key-slot is specified and

there is still a free key-slot, then the new passphrase will be

put into a free key-slot before the key-slot containing the old

passphrase is purged. If there is no free key-slot, then the

key-slot with the old passphrase is overwritten directly.

WARNING: If a key-slot is overwritten, a media failure during

this operation can cause the overwrite to fail after the old

passphrase has been wiped and make the LUKS container inaccessi‐

ble.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--new-keyfile-offset, --new-keyfile-size, --key-slot,

--force-password, --header, --disable-locks, --type,

--keyslot-cipher, --keyslot-key-size].

luksConvertKey <device>

Converts an existing LUKS2 keyslot to new pbkdf parameters. The

passphrase for keyslot to be converted must be supplied interac‐

tively or via --key-file. If no --pbkdf parameters are specified

LUKS2 default pbkdf values will apply.

If a keyslot is specified (via --key-slot), the passphrase for

that keyslot must be given. If no keyslot is specified and there

is still a free keyslot, then the new parameters will be put

into a free keyslot before the keyslot containing the old param‐

eters is purged. If there is no free keyslot, then the keyslot

with the old parameters is overwritten directly.

WARNING: If a keyslot is overwritten, a media failure during

this operation can cause the overwrite to fail after the old pa‐

rameters have been wiped and make the LUKS container inaccessi‐

ble.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--key-slot, --header, --disable-locks, --iter-time, --pbkdf,

--pbkdf-force-iterations, --pbkdf-memory, --pbkdf-parallel,

--keyslot-cipher, --keyslot-key-size].

luksKillSlot <device> <key slot number>

Wipe the key-slot number <key slot> from the LUKS device. Except

running in batch-mode (-q) a remaining passphrase must be sup‐

plied, either interactively or via --key-file. This command can

remove the last remaining key-slot, but requires an interactive

confirmation when doing so. Removing the last passphrase makes a

LUKS container permanently inaccessible.

<options> can be [--key-file, --keyfile-offset, --keyfile-size,

--header, --disable-locks, --type].

WARNING: If you read the passphrase from stdin (without further

argument or with '-' as an argument to --key-file), batch-mode

(-q) will be implicitly switched on and no warning will be given

when you remove the last remaining passphrase from a LUKS con‐

tainer. Removing the last passphrase makes the LUKS container

permanently inaccessible.

NOTE: If there is no passphrase provided (on stdin or through

--key-file argument) and batch-mode (-q) is active, the key-slot

is removed without any other warning.

erase <device>

luksErase <device>

Erase all keyslots and make the LUKS container permanently inac‐

cessible. You do not need to provide any password for this op‐

eration.

WARNING: This operation is irreversible.

luksUUID <device>

Print the UUID of a LUKS device.

Set new UUID if --uuid option is specified.

isLuks <device>

Returns true, if <device> is a LUKS device, false otherwise.

Use option -v to get human-readable feedback. 'Command success‐

ful.' means the device is a LUKS device.

By specifying --type you may query for specific LUKS version.

luksDump <device>

Dump the header information of a LUKS device.

If the --dump-master-key option is used, the LUKS device master

key is dumped instead of the keyslot info. Together with --mas‐

ter-key-file option, master key is dumped to a file instead of

standard output. Beware that the master key cannot be changed

without reencryption and can be used to decrypt the data stored

in the LUKS container without a passphrase and even without the

LUKS header. This means that if the master key is compromised,

the whole device has to be erased to prevent further access. Use

this option carefully.

To dump the master key, a passphrase has to be supplied, either

interactively or via --key-file.

<options> can be [--dump-master-key, --key-file, --keyfile-off‐

set, --keyfile-size, --header, --disable-locks, --mas‐

ter-key-file, --type].

WARNING: If --dump-master-key is used with --key-file and the

argument to --key-file is '-', no validation question will be

asked and no warning given.

luksHeaderBackup <device> --header-backup-file <file>

Stores a binary backup of the LUKS header and keyslot area.

Note: Using '-' as filename writes the header backup to a file

named '-'.

WARNING: This backup file and a passphrase valid at the time of

backup allows decryption of the LUKS data area, even if the

passphrase was later changed or removed from the LUKS device.

Also note that with a header backup you lose the ability to se‐

curely wipe the LUKS device by just overwriting the header and

key-slots. You either need to securely erase all header backups

in addition or overwrite the encrypted data area as well. The

second option is less secure, as some sectors can survive, e.g.

due to defect management.

luksHeaderRestore <device> --header-backup-file <file>

Restores a binary backup of the LUKS header and keyslot area

from the specified file.

Note: Using '-' as filename reads the header backup from a file

named '-'.

WARNING: Header and keyslots will be replaced, only the

passphrases from the backup will work afterward.

This command requires that the master key size and data offset

of the LUKS header already on the device and of the header

backup match. Alternatively, if there is no LUKS header on the

device, the backup will also be written to it.

token <add|remove|import|export> <device>

Action add creates new keyring token to enable auto-activation

of the device. For the auto-activation, the passphrase must be

stored in keyring with the specified description. Usually, the

passphrase should be stored in user or user-session keyring.

The token command is supported only for LUKS2.

For adding new keyring token, option --key-description is manda‐

tory. Also, new token is assigned to key slot specified with

--key-slot option or to all active key slots in the case

--key-slot option is omitted.

To remove existing token, specify the token ID which should be

removed with --token-id option.

WARNING: The action token remove removes any token type, not

just keyring type from token slot specified by --token-id op‐

tion.

Action import can store arbitrary valid token json in LUKS2

header. It may be passed via standard input or via file passed

in --json-file option. If you specify --key-slot then success‐

fully imported token is also assigned to the key slot.

Action export writes requested token json to a file passed with

--json-file or to standard output.

<options> can be [--header, --token-id, --key-slot, --key-de‐

scription, --disable-locks, --disable-keyring, --json-file].

convert <device> --type <format>

Converts the device between LUKS1 and LUKS2 format (if possi‐

ble). The conversion will not be performed if there is an addi‐

tional LUKS2 feature or LUKS1 has unsupported header size.

Conversion (both directions) must be performed on inactive de‐

vice. There must not be active dm-crypt mapping established for

LUKS header requested for conversion.

--type option is mandatory with following accepted values: luks1

or luks2.

WARNING: The convert action can destroy the LUKS header in the

case of a crash during conversion or if a media error occurs.

Always create a header backup before performing this operation!

<options> can be [--header, --type].

config <device>

Set permanent configuration options (store to LUKS header). The

config command is supported only for LUKS2.

The permanent options can be --priority to set priority (normal,

prefer, ignore) for keyslot (specified by --key-slot) or --label

and --subsystem.

<options> can be [--priority, --label, --subsystem, --key-slot,

--header].

loop-AES EXTENSION

cryptsetup supports mapping loop-AES encrypted partition using a com‐

patibility mode.

open --type loopaes <device> <name> --key-file <keyfile>

loopaesOpen <device> <name> --key-file <keyfile> (old syntax)

Opens the loop-AES <device> and sets up a mapping <name>.

If the key file is encrypted with GnuPG, then you have to use

--key-file=- and decrypt it before use, e.g. like this:

gpg --decrypt <keyfile> | cryptsetup loopaesOpen --key-file=-

<device> <name>

WARNING: The loop-AES extension cannot use the direct input of

key file on real terminal because the keys are separated by end-

of-line and only part of the multi-key file would be read.

If you need it in script, just use the pipe redirection:

echo $keyfile | cryptsetup loopaesOpen --key-file=- <device>

<name>

Use --keyfile-size to specify the proper key length if needed.

Use --offset to specify device offset. Note that the units need

to be specified in number of 512 byte sectors.

Use --skip to specify the IV offset. If the original device used

an offset and but did not use it in IV sector calculations, you

have to explicitly use --skip 0 in addition to the offset param‐

eter.

Use --hash to override the default hash function for passphrase

hashing (otherwise it is detected according to key size).

<options> can be [--key-file, --key-size, --offset, --skip,

--hash, --readonly, --allow-discards, --refresh].

See also section 7 of the FAQ and http://loop-aes.sourceforge.net for

more information regarding loop-AES.

TCRYPT (TrueCrypt-compatible and VeraCrypt) EXTENSION

cryptsetup supports mapping of TrueCrypt, tcplay or VeraCrypt (with

--veracrypt option) encrypted partition using a native Linux kernel

API. Header formatting and TCRYPT header change is not supported,

cryptsetup never changes TCRYPT header on-device.

TCRYPT extension requires kernel userspace crypto API to be available

(introduced in Linux kernel 2.6.38). If you are configuring kernel

yourself, enable "User-space interface for symmetric key cipher algo‐

rithms" in "Cryptographic API" section (CRYPTO\_USER\_API\_SKCIPHER .con‐

fig option).

Because TCRYPT header is encrypted, you have to always provide valid

passphrase and keyfiles.

Cryptsetup should recognize all header variants, except legacy cipher

chains using LRW encryption mode with 64 bits encryption block (namely

Blowfish in LRW mode is not recognized, this is limitation of kernel

crypto API).

To recognize a VeraCrypt device use the --veracrypt option. VeraCrypt

is just extension of TrueCrypt header with increased iteration count so

unlocking can take quite a lot of time (in comparison with TCRYPT de‐

vice).

To open a VeraCrypt device with a custom Personal Iteration Multiplier

(PIM) value, additionally to --veracrypt use either the --ver‐

acrypt-pim=<PIM> option to directly specify the PIM on the command-

line or use --veracrypt-query-pim to be prompted for the PIM.

The PIM value affects the number of iterations applied during key deri‐

vation. Please refer to https://www.veracrypt.fr/en/Personal%20Itera‐

tions%20Multiplier%20%28PIM%29.html for more detailed information.

NOTE: Activation with tcryptOpen is supported only for cipher chains

using LRW or XTS encryption modes.

The tcryptDump command should work for all recognized TCRYPT devices

and doesn't require superuser privilege.

To map system device (device with boot loader where the whole encrypted

system resides) use --tcrypt-system option. You can use partition de‐

vice as the parameter (parameter must be real partition device, not an

image in a file), then only this partition is mapped.

If you have the whole TCRYPT device as a file image and you want to map

multiple partition encrypted with system encryption, please create

loopback mapping with partitions first (losetup -P, see losetup(8) man

page for more info), and use loop partition as the device parameter.

If you use the whole base device as a parameter, one device for the

whole system encryption is mapped. This mode is available only for

backward compatibility with older cryptsetup versions which mapped

TCRYPT system encryption using the whole device.

To use hidden header (and map hidden device, if available), use

--tcrypt-hidden option.

To explicitly use backup (secondary) header, use --tcrypt-backup op‐

tion.

NOTE: There is no protection for a hidden volume if the outer volume is

mounted. The reason is that if there were any protection, it would re‐

quire some metadata describing what to protect in the outer volume and

the hidden volume would become detectable.

open --type tcrypt <device> <name>

tcryptOpen <device> <name> (old syntax)

Opens the TCRYPT (a TrueCrypt-compatible) <device> and sets up a

mapping <name>.

<options> can be [--key-file, --tcrypt-hidden, --tcrypt-system,

--tcrypt-backup, --readonly, --test-passphrase, --allow-dis‐

cards, --veracrypt, --veracrypt-pim, --veracrypt-query-pim].

The keyfile parameter allows a combination of file content with

the passphrase and can be repeated. Note that using keyfiles is

compatible with TCRYPT and is different from LUKS keyfile logic.

WARNING: Option --allow-discards cannot be combined with option

--tcrypt-hidden. For normal mapping, it can cause the destruc‐

tion of hidden volume (hidden volume appears as unused space for

outer volume so this space can be discarded).

tcryptDump <device>

Dump the header information of a TCRYPT device.

If the --dump-master-key option is used, the TCRYPT device mas‐

ter key is dumped instead of TCRYPT header info. Beware that the

master key (or concatenated master keys if cipher chain is used)

can be used to decrypt the data stored in the TCRYPT container

without a passphrase. This means that if the master key is com‐

promised, the whole device has to be erased to prevent further

access. Use this option carefully.

<options> can be [--dump-master-key, --key-file, --tcrypt-hid‐

den, --tcrypt-system, --tcrypt-backup].

The keyfile parameter allows a combination of file content with

the passphrase and can be repeated.

See also https://en.wikipedia.org/wiki/TrueCrypt for more information

regarding TrueCrypt.

Please note that cryptsetup does not use TrueCrypt code, please report

all problems related to this compatibility extension to the cryptsetup

project.

MISCELLANEOUS

repair <device>

Tries to repair the device metadata if possible. Currently sup‐

ported only for LUKS device type.

This command is useful to fix some known benign LUKS metadata

header corruptions. Only basic corruptions of unused keyslot are

fixable. This command will only change the LUKS header, not any

key-slot data. You may enforce LUKS version by adding --type op‐

tion.

WARNING: Always create a binary backup of the original header

before calling this command.

benchmark <options>

Benchmarks ciphers and KDF (key derivation function). Without

parameters, it tries to measure few common configurations.

To benchmark other ciphers or modes, you need to specify --ci‐

pher and --key-size options or --hash for KDF test.

NOTE: This benchmark is using memory only and is only informa‐

tive. You cannot directly predict real storage encryption speed

from it.

For testing block ciphers, this benchmark requires kernel

userspace crypto API to be available (introduced in Linux kernel

2.6.38). If you are configuring kernel yourself, enable "User-

space interface for symmetric key cipher algorithms" in "Crypto‐

graphic API" section (CRYPTO\_USER\_API\_SKCIPHER .config option).

<options> can be [--cipher, --key-size, --hash].

OPTIONS

--verbose, -v

Print more information on command execution.

--debug or --debug-json

Run in debug mode with full diagnostic logs. Debug output lines

are always prefixed by '#'. If --debug-json is used, additional

LUKS2 JSON data structures are printed.

--type <device-type>

Specifies required device type, for more info read BASIC COM‐

MANDS section.

--hash, -h <hash-spec>

Specifies the passphrase hash for open (for plain and loopaes

device types).

Specifies the hash used in the LUKS key setup scheme and volume

key digest for luksFormat. The specified hash is used as hash-

parameter for PBKDF2 and for the AF splitter.

The specified hash name is passed to the compiled-in crypto

backend. Different backends may support different hashes. For

luksFormat, the hash algorithm must provide at least 160 bits of

output, which excludes, e.g., MD5. Do not use a non-crypto hash

like "crc32" as this breaks security.

Values compatible with old version of cryptsetup are "ripemd160"

for open --type plain and "sha1" for luksFormat.

Use cryptsetup --help to show the defaults.

--cipher, -c <cipher-spec>

Set the cipher specification string.

cryptsetup --help shows the compiled-in defaults. The current

default in the distributed sources is "aes-cbc-essiv:sha256" for

plain dm-crypt and "aes-xts-plain64" for LUKS.

If a hash is part of the cipher specification, then it is used

as part of the IV generation. For example, ESSIV needs a hash

function, while "plain64" does not and hence none is specified.

For XTS mode you can optionally set a key size of 512 bits with

the -s option. Key size for XTS mode is twice that for other

modes for the same security level.

XTS mode requires kernel 2.6.24 or later and plain64 requires

kernel 2.6.33 or later. More information can be found in the

FAQ.

--verify-passphrase, -y

When interactively asking for a passphrase, ask for it twice and

complain if both inputs do not match. Advised when creating a

regular mapping for the first time, or when running luksFormat.

Ignored on input from file or stdin.

--key-file, -d name

Read the passphrase from file.

If the name given is "-", then the passphrase will be read from

stdin. In this case, reading will not stop at newline charac‐

ters.

With LUKS, passphrases supplied via --key-file are always the

existing passphrases requested by a command, except in the case

of luksFormat where --key-file is equivalent to the positional

key file argument.

If you want to set a new passphrase via key file, you have to

use a positional argument to luksAddKey.

See section NOTES ON PASSPHRASE PROCESSING for more information.

--keyfile-offset value

Skip value bytes at the beginning of the key file. Works with

all commands that accept key files.

--keyfile-size, -l value

Read a maximum of value bytes from the key file. The default is

to read the whole file up to the compiled-in maximum that can be

queried with --help. Supplying more data than the compiled-in

maximum aborts the operation.

This option is useful to cut trailing newlines, for example. If

--keyfile-offset is also given, the size count starts after the

offset. Works with all commands that accept key files.

--new-keyfile-offset value

Skip value bytes at the start when adding a new passphrase from

key file with luksAddKey.

--new-keyfile-size value

Read a maximum of value bytes when adding a new passphrase from

key file with luksAddKey. The default is to read the whole file

up to the compiled-in maximum length that can be queried with

--help. Supplying more than the compiled in maximum aborts the

operation. When --new-keyfile-offset is also given, reading

starts after the offset.

--master-key-file

Use a master key stored in a file.

For luksFormat this allows creating a LUKS header with this spe‐

cific master key. If the master key was taken from an existing

LUKS header and all other parameters are the same, then the new

header decrypts the data encrypted with the header the master

key was taken from.

Action luksDump together with --dump-master-key option: The vol‐

ume (master) key is stored in a file instead of being printed

out to standard output.

WARNING: If you create your own master key, you need to make

sure to do it right. Otherwise, you can end up with a low-en‐

tropy or otherwise partially predictable master key which will

compromise security.

For luksAddKey this allows adding a new passphrase without hav‐

ing to know an existing one.

For open this allows one to open the LUKS device without giving

a passphrase.

--dump-master-key

For luksDump this option includes the master key in the dis‐

played information. Use with care, as the master key can be used

to bypass the passphrases, see also option --master-key-file.

--json-file

Read token json from a file or write token to it. See token ac‐

tion for more information. --json-file=- reads json from stan‐

dard input or writes it to standard output respectively.

--use-random

--use-urandom

For luksFormat these options define which kernel random number

generator will be used to create the master key (which is a

long-term key).

See NOTES ON RANDOM NUMBER GENERATORS for more information. Use

cryptsetup --help to show the compiled-in default random number

generator.

WARNING: In a low-entropy situation (e.g. in an embedded sys‐

tem), both selections are problematic. Using /dev/urandom can

lead to weak keys. Using /dev/random can block a long time, po‐

tentially forever, if not enough entropy can be harvested by the

kernel.

--key-slot, -S <0-7>

For LUKS operations that add key material, this options allows

you to specify which key slot is selected for the new key. This

option can be used for luksFormat, and luksAddKey.

In addition, for open, this option selects a specific key-slot

to compare the passphrase against. If the given passphrase

would only match a different key-slot, the operation fails.

--key-size, -s <bits>

Sets key size in bits. The argument has to be a multiple of 8.

The possible key-sizes are limited by the cipher and mode used.

See /proc/crypto for more information. Note that key-size in

/proc/crypto is stated in bytes.

This option can be used for open --type plain or luksFormat.

All other LUKS actions will use the key-size specified in the

LUKS header. Use cryptsetup --help to show the compiled-in de‐

faults.

--size, -b <number of 512 byte sectors>

Set the size of the device in sectors of 512 bytes. This option

is only relevant for the open and resize actions.

--offset, -o <number of 512 byte sectors>

Start offset in the backend device in 512-byte sectors. This

option is only relevant for the open action with plain or

loopaes device types or for LUKS devices in luksFormat.

For LUKS, the --offset option sets the data offset (payload) of

data device and must be be aligned to 4096-byte sectors (must be

multiple of 8). This option cannot be combined with

--align-payload option.

--skip, -p <number of 512 byte sectors>

Start offset used in IV calculation in 512-byte sectors (how

many sectors of the encrypted data to skip at the beginning).

This option is only relevant for the open action with plain or

loopaes device types.

Hence, if --offset n, and --skip s, sector n (the first sector

of the encrypted device) will get a sector number of s for the

IV calculation.

--readonly, -r

set up a read-only mapping.

--shared

Creates an additional mapping for one common ciphertext device.

Arbitrary mappings are supported. This option is only relevant

for the open --type plain action. Use --offset, --size and

--skip to specify the mapped area.

--pbkdf <PBKDF spec>

Set Password-Based Key Derivation Function (PBKDF) algorithm for

LUKS keyslot. The PBKDF can be: pbkdf2 (for PBKDF2 according to

RFC2898), argon2i for Argon2i or argon2id for Argon2id (see

https://www.cryptolux.org/index.php/Argon2 for more info).

For LUKS1, only PBKDF2 is accepted (no need to use this option).

The default PBKDF2 for LUKS2 is set during compilation time and

is available in cryptsetup --help output.

A PBKDF is used for increasing dictionary and brute-force attack

cost for keyslot passwords. The parameters can be time, memory

and parallel cost.

For PBKDF2, only time cost (number of iterations) applies. For

Argon2i/id, there is also memory cost (memory required during

the process of key derivation) and parallel cost (number of

threads that run in parallel during the key derivation.

Note that increasing memory cost also increases time, so the fi‐

nal parameter values are measured by a benchmark. The benchmark

tries to find iteration time (--iter-time) with required memory

cost --pbkdf-memory. If it is not possible, the memory cost is

decreased as well. The parallel cost --pbkdf-parallel is con‐

stant, is is checked against available CPU cores (if not avail‐

able, it is decreased) and the maximum parallel cost is 4.

You can see all PBKDF parameters for particular LUKS2 keyslot

with luksDump command.

NOTE: If you do not want to use benchmark and want to specify

all parameters directly, use --pbkdf-force-iterations with

--pbkdf-memory and --pbkdf-parallel. This will override the

values without benchmarking. Note it can cause extremely long

unlocking time. Use only is specified cases, for example, if you

know that the formatted device will be used on some small embed‐

ded system. In this case, the LUKS PBKDF2 digest will be set to

the minimum iteration count.

--iter-time, -i <number of milliseconds>

The number of milliseconds to spend with PBKDF passphrase pro‐

cessing. This option is only relevant for LUKS operations that

set or change passphrases, such as luksFormat or luksAddKey.

Specifying 0 as parameter selects the compiled-in default.

--pbkdf-memory <number>

Set the memory cost for PBKDF (for Argon2i/id the number repre‐

sents kilobytes). Note that it is maximal value, PBKDF bench‐

mark or available physical memory can decrease it. This option

is not available for PBKDF2.

--pbkdf-parallel <number>

Set the parallel cost for PBKDF (number of threads, up to 4).

Note that it is maximal value, it is decreased automatically if

CPU online count is lower. This option is not available for

PBKDF2.

--pbkdf-force-iterations <num>

Avoid PBKDF benchmark and set time cost (iterations) directly.

It can be used for LUKS/LUKS2 device only. See --pbkdf option

for more info.

--batch-mode, -q

Suppresses all confirmation questions. Use with care!

If the -y option is not specified, this option also switches off

the passphrase verification for luksFormat.

--progress-frequency <seconds>

Print separate line every <seconds> with wipe progress.

--timeout, -t <number of seconds>

The number of seconds to wait before timeout on passphrase input

via terminal. It is relevant every time a passphrase is asked,

for example for open, luksFormat or luksAddKey. It has no ef‐

fect if used in conjunction with --key-file.

This option is useful when the system should not stall if the

user does not input a passphrase, e.g. during boot. The default

is a value of 0 seconds, which means to wait forever.

--tries, -T

How often the input of the passphrase shall be retried. This

option is relevant every time a passphrase is asked, for example

for open, luksFormat or luksAddKey. The default is 3 tries.

--align-payload <number of 512 byte sectors>

Align payload at a boundary of value 512-byte sectors. This op‐

tion is relevant for luksFormat.

If not specified, cryptsetup tries to use the topology info pro‐

vided by the kernel for the underlying device to get the optimal

alignment. If not available (or the calculated value is a mul‐

tiple of the default) data is by default aligned to a 1MiB

boundary (i.e. 2048 512-byte sectors).

For a detached LUKS header, this option specifies the offset on

the data device. See also the --header option.

WARNING: This option is DEPRECATED and has often unexpected im‐

pact to the data offset and keyslot area size (for LUKS2) due to

the complex rounding. For fixed data device offset use --offset

option instead.

--uuid=UUID

Use the provided UUID for the luksFormat command instead of gen‐

erating a new one. Changes the existing UUID when used with the

luksUUID command.

The UUID must be provided in the standard UUID format, e.g.

12345678-1234-1234-1234-123456789abc.

--allow-discards

Allow the use of discard (TRIM) requests for the device. This

option is only relevant for open action.

WARNING: This command can have a negative security impact be‐

cause it can make filesystem-level operations visible on the

physical device. For example, information leaking filesystem

type, used space, etc. may be extractable from the physical de‐

vice if the discarded blocks can be located later. If in doubt,

do not use it.

A kernel version of 3.1 or later is needed. For earlier kernels,

this option is ignored.

--perf-same\_cpu\_crypt

Perform encryption using the same cpu that IO was submitted on.

The default is to use an unbound workqueue so that encryption

work is automatically balanced between available CPUs. This op‐

tion is only relevant for open action.

NOTE: This option is available only for low-level dm-crypt per‐

formance tuning, use only if you need a change to default dm-

crypt behaviour. Needs kernel 4.0 or later.

--perf-submit\_from\_crypt\_cpus

Disable offloading writes to a separate thread after encryption.

There are some situations where offloading write bios from the

encryption threads to a single thread degrades performance sig‐

nificantly. The default is to offload write bios to the same

thread. This option is only relevant for open action.

NOTE: This option is available only for low-level dm-crypt per‐

formance tuning, use only if you need a change to default dm-

crypt behaviour. Needs kernel 4.0 or later.

--test-passphrase

Do not activate the device, just verify passphrase. This option

is only relevant for open action (the device mapping name is not

mandatory if this option is used).

--header <device or file storing the LUKS header>

Use a detached (separated) metadata device or file where the

LUKS header is stored. This option allows one to store cipher‐

text and LUKS header on different devices.

This option is only relevant for LUKS devices and can be used

with the luksFormat, open, luksSuspend, luksResume, status and

resize commands.

For luksFormat with a file name as the argument to --header, the

file will be automatically created if it does not exist. See

the cryptsetup FAQ for header size calculation.

For other commands that change the LUKS header (e.g. luksAdd‐

Key), specify the device or file with the LUKS header directly

as the LUKS device.

If used with luksFormat, the --align-payload option is taken as

absolute sector alignment on ciphertext device and can be zero.

WARNING: There is no check whether the ciphertext device speci‐

fied actually belongs to the header given. In fact, you can

specify an arbitrary device as the ciphertext device for open

with the --header option. Use with care.

--header-backup-file <file>

Specify file with header backup for luksHeaderBackup or luk‐

sHeaderBackup actions.

--force-password

Do not use password quality checking for new LUKS passwords.

This option applies only to luksFormat, luksAddKey and

luksChangeKey and is ignored if cryptsetup is built without

password quality checking support.

For more info about password quality check, see the manual page

for pwquality.conf(5) and passwdqc.conf(5).

--deferred

Defers device removal in close command until the last user

closes it.

--disable-locks

Disable lock protection for metadata on disk. This option is

valid only for LUKS2 and ignored for other formats.

WARNING: Do not use this option unless you run cryptsetup in a

restricted environment where locking is impossible to perform

(where /run directory cannot be used).

--disable-keyring

Do not load volume key in kernel keyring but use store key di‐

rectly in the dm-crypt target. This option is supported only

for the LUKS2 format.

--key-description <text>

Set key description in keyring for use with token command.

--priority <normal|prefer|ignore>

Set a priority for LUKS2 keyslot. The prefer priority marked

slots are tried before normal priority. The ignored priority

means, that slot is never used, if not explicitly requested by

--key-slot option.

--token-id

Specify what token to use in actions token, open or resize. If

omitted, all available tokens will be checked before proceeding

further with passphrase prompt.

--token-only

Do not proceed further with action (any of token, open or re‐

size) if token activation failed. Without the option, action

asks for passphrase to proceed further.

--sector-size <bytes>

Set sector size for use with disk encryption. It must be power

of two and in range 512 - 4096 bytes. The default is 512 bytes

sectors. This option is available only in the LUKS2 mode.

Note that if sector size is higher than underlying device hard‐

ware sector and there is not integrity protection that uses data

journal, using this option can increase risk on incomplete sec‐

tor writes during a power fail.

If used together with --integrity option and dm-integrity jour‐

nal, the atomicity of writes is guaranteed in all cases (but it

cost write performance - data has to be written twice).

Increasing sector size from 512 bytes to 4096 bytes can provide

better performance on most of the modern storage devices and

also with some hw encryption accelerators.

--persistent

If used with LUKS2 devices and activation commands like open,

the specified activation flags are persistently written into

metadata and used next time automatically even for normal acti‐

vation. (No need to use cryptab or other system configuration

files.) Only --allow-discards, --perf-same\_cpu\_crypt,

--perf-submit\_from\_crypt\_cpus and --integrity-no-journal can be

stored persistently.

--refresh

Refreshes an active device with new set of parameters. See ac‐

tion refresh description for more details.

--label <LABEL>

--subsystem <SUBSYSTEM> Set label and subsystem description for

LUKS2 device, can be used in config and format actions. The la‐

bel and subsystem are optional fields and can be later used in

udev scripts for triggering user actions once device marked by

these labels is detected.

--integrity <integrity algorithm>

Specify integrity algorithm to be used for authenticated disk

encryption in LUKS2.

WARNING: This extension is EXPERIMENTAL and requires dm-integ‐

rity kernel target (available since kernel version 4.12). For

native AEAD modes, also enable "User-space interface for AEAD

cipher algorithms" in "Cryptographic API" section (CON‐

FIG\_CRYPTO\_USER\_API\_AEAD .config option).

For more info, see AUTHENTICATED DISK ENCRYPTION section.

--luks2-metadata-size <size>

This option can be used to enlarge the LUKS2 metadata (JSON)

area. The size includes 4096 bytes for binary metadata (usable

JSON area is smaller of the binary area). According to LUKS2

specification, only these values are valid: 16, 32, 64, 128,

256, 512, 1024, 2048 and 4096 kB The <size> can be specified

with unit suffix (for example 128k).

--luks2-keyslots-size <size>

This option can be used to set specific size of the LUKS2 binary

keyslot area (key material is encrypted there). The value must

be aligned to multiple of 4096 bytes with maximum size 128MB.

The <size> can be specified with unit suffix (for example 128k).

--keyslot-cipher <cipher-spec>

This option can be used to set specific cipher encryption for

the LUKS2 keyslot area.

--keyslot-key-size <bits>

This option can be used to set specific key size for the LUKS2

keyslot area.

--integrity-no-journal

Activate device with integrity protection without using data

journal (direct write of data and integrity tags). Note that

without journal power fail can cause non-atomic write and data

corruption. Use only if journalling is performed on a different

storage layer.

--integrity-no-wipe

Skip wiping of device authentication (integrity) tags. If you

skip this step, sectors will report invalid integrity tag until

an application write to the sector.

NOTE: Even some writes to the device can fail if the write is

not aligned to page size and page-cache initiates read of a sec‐

tor with invalid integrity tag.

--unbound

Creates new LUKS2 unbound keyslot. See luksAddKey action for

more details.

--tcrypt-hidden

--tcrypt-system --tcrypt-backup Specify which TrueCrypt on-disk

header will be used to open the device. See TCRYPT section for

more info.

--veracrypt

Allow VeraCrypt compatible mode. Only for TCRYPT extension. See

TCRYPT section for more info.

--veracrypt-pim

--veracrypt-query-pim Use a custom Personal Iteration Multiplier

(PIM) for VeraCrypt device. See TCRYPT section for more info.

--version

Show the program version.

--usage

Show short option help.

--help, -?

Show help text and default parameters.

RETURN CODES

Cryptsetup returns 0 on success and a non-zero value on error.

Error codes are: 1 wrong parameters, 2 no permission (bad passphrase),

3 out of memory, 4 wrong device specified, 5 device already exists or

device is busy.

NOTES ON PASSPHRASE PROCESSING FOR PLAIN MODE

Note that no iterated hashing or salting is done in plain mode. If

hashing is done, it is a single direct hash. This means that low-en‐

tropy passphrases are easy to attack in plain mode.

From a terminal: The passphrase is read until the first newline, i.e.

'\n'. The input without the newline character is processed with the

default hash or the hash specified with --hash. The hash result will

be truncated to the key size of the used cipher, or the size specified

with -s.

From stdin: Reading will continue until a newline (or until the maximum

input size is reached), with the trailing newline stripped. The maximum

input size is defined by the same compiled-in default as for the maxi‐

mum key file size and can be overwritten using --keyfile-size option.

The data read will be hashed with the default hash or the hash speci‐

fied with --hash. The hash result will be truncated to the key size of

the used cipher, or the size specified with -s.

Note that if --key-file=- is used for reading the key from stdin,

trailing newlines are not stripped from the input.

If "plain" is used as argument to --hash, the input data will not be

hashed. Instead, it will be zero padded (if shorter than the key size)

or truncated (if longer than the key size) and used directly as the bi‐

nary key. This is useful for directly specifying a binary key. No

warning will be given if the amount of data read from stdin is less

than the key size.

From a key file: It will be truncated to the key size of the used ci‐

pher or the size given by -s and directly used as a binary key.

WARNING: The --hash argument is being ignored. The --hash option is

usable only for stdin input in plain mode.

If the key file is shorter than the key, cryptsetup will quit with an

error. The maximum input size is defined by the same compiled-in de‐

fault as for the maximum key file size and can be overwritten using

--keyfile-size option.

NOTES ON PASSPHRASE PROCESSING FOR LUKS

LUKS uses PBKDF2 to protect against dictionary attacks and to give some

protection to low-entropy passphrases (see RFC 2898 and the cryptsetup

FAQ).

From a terminal: The passphrase is read until the first newline and

then processed by PBKDF2 without the newline character.

From stdin: LUKS will read passphrases from stdin up to the first new‐

line character or the compiled-in maximum key file length. If --key‐

file-size is given, it is ignored.

From key file: The complete keyfile is read up to the compiled-in maxi‐

mum size. Newline characters do not terminate the input. The --key‐

file-size option can be used to limit what is read.

Passphrase processing: Whenever a passphrase is added to a LUKS header

(luksAddKey, luksFormat), the user may specify how much the time the

passphrase processing should consume. The time is used to determine the

iteration count for PBKDF2 and higher times will offer better protec‐

tion for low-entropy passphrases, but open will take longer to com‐

plete. For passphrases that have entropy higher than the used key

length, higher iteration times will not increase security.

The default setting of one or two seconds is sufficient for most prac‐

tical cases. The only exception is a low-entropy passphrase used on a

device with a slow CPU, as this will result in a low iteration count.

On a slow device, it may be advisable to increase the iteration time

using the --iter-time option in order to obtain a higher iteration

count. This does slow down all later luksOpen operations accordingly.

INCOHERENT BEHAVIOR FOR INVALID PASSPHRASES/KEYS

LUKS checks for a valid passphrase when an encrypted partition is un‐

locked. The behavior of plain dm-crypt is different. It will always

decrypt with the passphrase given. If the given passphrase is wrong,

the device mapped by plain dm-crypt will essentially still contain en‐

crypted data and will be unreadable.

NOTES ON SUPPORTED CIPHERS, MODES, HASHES AND KEY SIZES

The available combinations of ciphers, modes, hashes and key sizes de‐

pend on kernel support. See /proc/crypto for a list of available op‐

tions. You might need to load additional kernel crypto modules in order

to get more options.

For the --hash option, if the crypto backend is libgcrypt, then all al‐

gorithms supported by the gcrypt library are available. For other

crypto backends, some algorithms may be missing.

NOTES ON PASSPHRASES

Mathematics can't be bribed. Make sure you keep your passphrases safe.

There are a few nice tricks for constructing a fallback, when suddenly

out of the blue, your brain refuses to cooperate. These fallbacks need

LUKS, as it's only possible with LUKS to have multiple passphrases.

Still, if your attacker model does not prevent it, storing your

passphrase in a sealed envelope somewhere may be a good idea as well.

NOTES ON RANDOM NUMBER GENERATORS

Random Number Generators (RNG) used in cryptsetup are always the kernel

RNGs without any modifications or additions to data stream produced.

There are two types of randomness cryptsetup/LUKS needs. One type

(which always uses /dev/urandom) is used for salts, the AF splitter and

for wiping deleted keyslots.

The second type is used for the volume (master) key. You can switch be‐

tween using /dev/random and /dev/urandom here, see --use-random and

--use-urandom options. Using /dev/random on a system without enough en‐

tropy sources can cause luksFormat to block until the requested amount

of random data is gathered. In a low-entropy situation (embedded sys‐

tem), this can take a very long time and potentially forever. At the

same time, using /dev/urandom in a low-entropy situation will produce

low-quality keys. This is a serious problem, but solving it is out of

scope for a mere man-page. See urandom(4) for more information.

AUTHENTICATED DISK ENCRYPTION (EXPERIMENTAL)

Since Linux kernel version 4.12 dm-crypt supports authenticated disk

encryption.

Normal disk encryption modes are length-preserving (plaintext sector is

of the same size as a ciphertext sector) and can provide only confiden‐

tiality protection, but not cryptographically sound data integrity pro‐

tection.

Authenticated modes require additional space per-sector for authentica‐

tion tag and use Authenticated Encryption with Additional Data (AEAD)

algorithms.

If you configure LUKS2 device with data integrity protection, there

will be an underlying dm-integrity device, which provides additional

per-sector metadata space and also provide data journal protection to

ensure atomicity of data and metadata update. Because there must be

additional space for metadata and journal, the available space for the

device will be smaller than for length-preserving modes.

The dm-crypt device then resides on top of such a dm-integrity device.

All activation and deactivation of this device stack is performed by

cryptsetup, there is no difference in using luksOpen for integrity pro‐

tected devices. If you want to format LUKS2 device with data integrity

protection, use --integrity option.

Some integrity modes requires two independent keys (key for encryption

and for authentication). Both these keys are stored in one LUKS

keyslot.

WARNING: All support for authenticated modes is experimental and there

are only some modes available for now. Note that there are a very few

authenticated encryption algorithms that are suitable for disk encryp‐

tion.

NOTES ON LOOPBACK DEVICE USE

Cryptsetup is usually used directly on a block device (disk partition

or LVM volume). However, if the device argument is a file, cryptsetup

tries to allocate a loopback device and map it into this file. This

mode requires Linux kernel 2.6.25 or more recent which supports the

loop autoclear flag (loop device is cleared on the last close automati‐

cally). Of course, you can always map a file to a loop-device manually.

See the cryptsetup FAQ for an example.

When device mapping is active, you can see the loop backing file in the

status command output. Also see losetup(8).

LUKS2 header locking

The LUKS2 on-disk metadata is updated in several steps and to achieve

proper atomic update, there is a locking mechanism. For an image in

file, code uses flock(2) system call. For a block device, lock is per‐

formed over a special file stored in a locking directory (by default

/run/lock/cryptsetup). The locking directory should be created with

the proper security context by the distribution during the boot-up

phase. Only LUKS2 uses locks, other formats do not use this mechanism.

DEPRECATED ACTIONS

The reload action is no longer supported. Please use dmsetup(8) if you

need to directly manipulate with the device mapping table.

The luksDelKey was replaced with luksKillSlot.

REPORTING BUGS

Report bugs, including ones in the documentation, on the cryptsetup

mailing list at <dm-crypt@saout.de> or in the 'Issues' section on LUKS

website. Please attach the output of the failed command with the --de‐

bug option added.

AUTHORS

cryptsetup originally written by Jana Saout <jana@saout.de>

The LUKS extensions and original man page were written by Clemens Fruh‐

wirth <clemens@endorphin.org>.

Man page extensions by Milan Broz <gmazyland@gmail.com>.

Man page rewrite and extension by Arno Wagner <arno@wagner.name>.

COPYRIGHT

Copyright © 2004 Jana Saout

Copyright © 2004-2006 Clemens Fruhwirth

Copyright © 2012-2014 Arno Wagner

Copyright © 2009-2019 Red Hat, Inc.

Copyright © 2009-2019 Milan Broz

This is free software; see the source for copying conditions. There is

NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR

PURPOSE.

SEE ALSO

The LUKS website at https://gitlab.com/cryptsetup/cryptsetup/

The cryptsetup FAQ, contained in the distribution package and online at

https://gitlab.com/cryptsetup/cryptsetup/wikis/FrequentlyAskedQuestions

The cryptsetup mailing list and list archive, see FAQ entry 1.6.

The LUKS on-disk format specification available at https://git‐

lab.com/cryptsetup/cryptsetup/wikis/Specification

cryptsetup January 2019

# eCryptfs

<https://help.ubuntu.com/lts/serverguide/ecryptfs.html>

<https://wiki.archlinux.fr/Encryption_avec_eCryptfs>

<https://wiki.archlinux.org/index.php/ECryptfs>

<https://doc.ubuntu-fr.org/ecryptfs>

|  |  |
| --- | --- |
| Fichier | Description |
| apache2.conf | **Fichier de configuration principal Apache2** Contient des paramètres globaux pour Apache2 |
| httpd.conf | **Fichier de configuration principal Apache2**, nommé d'après le démon httpd (Historiquement)  **Fichier inexistant** (Maintenant) Ce fichier peut être présent mais vide, car toutes les options de configuration ont été déplacées vers les répertoires référencés ci-dessous |
| /conf-available/ | **Répertoire contient les fichiers de configuration disponibles** Tous les fichiers qui se trouvaient auparavant dans /etc/apache2/conf.d doivent être déplacés vers /etc/apache2/conf-available |
| /conf-enabled/ | **Répertoire contenant les liens symboliques vers les fichiers de**  **/etc/apache2/conf-available**  Lorsqu'un fichier de configuration est lié symboliquement, il sera activé lors du prochain redémarrage d'apache2 |
| envvars | **Fichier dans lequel les variables d'environnement Apache2 sont définies** |
| /mods-available/ | **Répertoire contenant les fichiers de configuration des deux modules de chargement et de configuration** Cependant, tous les modules n'auront pas de fichiers de configuration spécifiques |
| /mods-enabled/ | **Répertoire contenant les liens symboliques vers les fichiers dans** **/etc/apache2/mods-available** Lorsqu’un fichier de configuration de module est lié symboliquement, il sera activé lors du prochain redémarrage d'apache2 |
| ports.conf | **Fichier contenant les directives qui déterminent les ports TCP** sur lesquels Apache2 écoute |
| /sites-available/ | **Répertoire contenant des fichiers de configuration pour les hôtes virtuels Apache2** Les hôtes virtuels permettent de configurer Apache2 pour plusieurs sites dotés de configurations distinctes |
| /sites-enabled/ | Répertoire contenant des liens symboliques vers le répertoire  /etc/apache2/sites-available (tout comme mods-enabled)  De même, lorsqu'un fichier de configuration de sites-available est lié symboliquement, le site configuré par celui-ci sera actif une fois Apache2 redémarré |
| magic | **Fichier contenant les instructions pour déterminer le type MIME** en fonction des premiers octets d'un fichier |