

Daten verschlüsseln

mit DM-Crypt für Linux

3. April 2011

Inhaltsverzeichnis

1	Einleitung	1
2	DM-Crypt für Linux	3
2.1	Gedanken zum Passwort	4
2.2	Verschlüsselten Container erstellen	4
2.3	Passwörter verwalten	6
2.4	Verschlüsselten Container öffnen/schließen	6
2.5	Debian GNU/Linux komplett verschlüsseln	9
2.6	HOME-Verzeichnis verschlüsseln	9
2.7	SWAP und /tmp verschlüsseln	9

1 Einleitung

Dass die Verschlüsselung von Daten der Erhaltung einer Privatsphäre dient, bemerkt man spätestens, wenn ein USB-Stick verloren geht. Wird ein Laptop gestohlen, möchte man die Fotosammlung sicher nicht im Internet sehen.

Investigative Journalisten, Rechtsanwälte und auch Priester haben das Recht und die Pflicht, ihre Informanten bzw. Klienten zu schützen. Sie sollten sich frühzeitig Gedanken über ein Konzept zur Verschlüsselung machen.

Die kurzen Beispiele zeigen, dass unterschiedliche Anforderungen an eine Verschlüsselung bestehen können. Bevor man wild anfängt, alles irgendwie zu verschlüsseln, sollte man sich Gedanken über die Bedrohung machen, gegen die man sich schützen will:

1. **Schutz sensibler Daten** wie z.B. Passwortlisten, Revocation Certificates o.ä. erfordert die Speicherung in einem Container oder verschlüsselten Archiv, welches auch im normalen Betrieb geschlossen ist.
2. **Schutz aller persönlichen Daten** bei Verlust oder Diebstahl von Laptop oder USB-Stick erfordert eine Software, die transparent arbeitet ohne den Nutzer zu behindern und bei korrekter Anmeldung möglichst automatisch den Daten-Container öffnet (beispielsweise TrueCrypt für WINDOWS oder DM-Crypt für Linux).

3. **Backups auf externen Medien** enthalten in der Regel die wichtigen privaten Daten und sollten ebenfalls verschlüsselt sein. Dabei sollte die Wiederherstellung auch bei totalem Datenverlust möglich sein. Es ist nicht sinnvoll, die Daten mit einem PGP-Schlüssel zu chiffrieren, der nach einem Crash nicht mehr verfügbar ist.
4. Wer eine **Manipulation der Sytemdaten** befürchtet, kann seinen Rechner komplett verschlüsseln (mit Truecrypt für WINDOWS, DM-Crypt für Linux oder GELI für FreeBSD) und von einem sauberen USB-Stick booten.
5. Zur **Herausgabe von Schlüsseln** im Fall einer Beschlagnahme des Rechners oder verschlüsselten Datenträgers gibt es immer wieder Missverständnisse. In Deutschland gelten folgende gesetzlichen Regelungen:
 - Richten sich die Ermittlungen gegen den Besitzer des Rechners oder Datenträgers muss man grundsätzlich keine Keys herausgeben.
 - Richten sich die Ermittlungen gegen Dritte, kann man die Herausgabe von Keys verweigern, wenn man sich auf das Recht zur Zeugnisverweigerung berufen oder glaubhaft(!) versichern kann, dass man sich damit selbst belasten würde. Im Zweifel sollte man einen Anwalt konsultieren.

In Großbritannien ist es bereits anders. Gemäß dem dort seit Oktober 2007 geltendem RIPA-Act können Nutzer von Verschlüsselung unter Strafandrohung zur Herausgabe der Schlüssel gezwungen werden. Es drohen bis zu 2 Jahre Gefängnis oder Geldstrafen. Das die Anwendung des Gesetzes nicht auf die bösen Terroristen beschränkt ist, kann man bei [Heise](#) nachlesen. Es wurde als ersten gegen eine Gruppe von Tierschützern angewendet.

2 DM-Crypt für Linux

DM-Crypt ist seit Version 2.6.4 fester Bestandteil des Linux-Kernels und somit in allen aktuellen Distributionen enthalten. Es nutzt den Device-Mapper. Folgende Software wird außerdem benötigt:

- Das Tool **cryptsetup** (mit LUKS-Support) kann zum Erstellen, Öffnen und Schließen der verschlüsselten Container eingesetzt werden. Aktuelle Distributionen enthalten es: Debian GNU/Linux im Paket *cryptsetup*, SuSE-Linux im Paket *util-linux-crypto*.

Einige Distributionen installieren das Tool unter dem Namen *cryptsetup-luks*. Die im Folgenden beschriebenen Befehle sind dann entsprechend anzupassen. Besser wäre es, einen Link zu erstellen. Dann funktionieren auch die Skripte *mount.crypt* und *umount.crypt* aus der Sammlung *pam-mount*.

```
# ln -s /usr/sbin/cryptsetup-luks /sbin/cryptsetup
```

- Das Paket **pmount** enthält einen Wrapper für das *mount*-Kommando, welcher automatisch verschlüsselte Laufwerke erkennt und vor dem Einbinden das Passwort abfragt. Aktuelle Debian-Distributionen verwenden es standardmäßig.
- Die Sammlung **pam-mount** enthält weitere Skripte, das das Öffnen und Schließen verschlüsselter Container vereinfachen. Die Skripte ermöglichen beispielsweise das Öffnen eines Containers automatisch beim Login. Unter Debian installiert man die Tools wie üblich mit

```
# aptitude install libpam-mount.
```

- Das Kernelmodul **dm_crypt** muss vor der Verwendung der oben genannten Skripte geladen werden. In Abhängigkeit von der bevorzugten Distribution und der Installationsvariante wird das Modul bereits beim Booten geladen oder ist statisch in *initrd.img* eingebunden. Einfach probieren.

Sollte beim Erstellen oder Öffnen eines verschlüsselten Containers die folgende Fehlermeldung auftreten:

```
Command failed: Failed to setup dm-crypt key mapping.  
Check kernel for support for the aes-cbc-essiv:sha256 cipher
```

ist das Kernel-Modul *dm_crypt* zu laden:

```
# modprobe dm_crypt
```

Außerdem sollte das Modul in die Liste der beim Systemstart zu ladenen Module eingefügt werden. In der Datei */etc/modules* ist die Zeile *dm_crypt* anzuhängen.

2.1 Gedanken zum Passwort

An Stelle von *Password* sollte man vielleicht die Bezeichnung *Passphrase* bevorzugen. Sie suggeriert, dass es auch ein wenig länger sein darf und dass Leerzeichen durchaus erlaubt sind.

Eine gute Passphrase sollte leicht merkbar aber schwer zu erraten sein. Außer Buchstaben sollte sie auch Zahlen und Sonderzeichen enthalten und etwa 20 Zeichen lang sein. Soetwas schüttelt man nicht einfach aus dem Ärmel. Wie wäre es mit folgender Phrase:

das geht nur %mich% _AN_

Zusätzlich zur Passphrase können auch Keyfiles als Schlüssel genutzt werden. Damit ist es möglich, eine Zwei-Faktor-Authentifizierung aufzubauen: eine Passphrase, die man im Kopf hat, und ein Keyfile, welches man in der Hand hat. Ein Angreifer müsste beides erlangen.

Die LUKS-Erweiterung von *cryptsetup* erlaubt es, bis zu 8 Passphrasen und Keyfiles zum Öffnen eines Containers zu nutzen. Damit ist es möglich, mehreren Nutzern den Zugriff mit einem eigenen Passwort zu erlauben.

Soll ein verschlüsselter Container mit dem Login eines Nutzers automatisch geöffnet werden, muss eines der 8 möglichen Passwörter mit dem Login-Passwort des Nutzers identisch sein. Login-Manager wie KDM oder GDM können das eingegebene Passwort an das pam-mount Modul weiterreichen. Dieses Feature kann beispielsweise für ein verschlüsseltes */home* Verzeichnis genutzt werden.

WICHTIG: bei Änderung des Login-Passwortes muss auch das Passwort für den Container geändert werden. Sie werden nicht automatisch synchronisiert.

2.2 Verschlüsselten Container erstellen

Alle folgenden Schritte sind als *root* auszuführen. Zum Aufwärmen soll zuerst die Partition */dev/hda4* verschlüsselt werden. Debian und Ubuntu enthalten das Skript `luksformat`, dass alle Aufgaben erledigt.

```
# luksformat -t ext3 /dev/hda4
```

Das ist alles. Der Vorgang dauert ein wenig und es wird 3x die Passphrase abgefragt. Ein Keyfile kann dieses Script nicht nutzen! Um einen USB-Stick komplett zu verschlüsseln, wählt man */dev/sdb1* oder */dev/sda1*. Es ist vor(!) Aufruf des Kommandos zu prüfen, unter welchem Device der Stick zur Verfügung steht.

Verschlüsselten Container erstellen für Genießer

Am Beispiel einer verschlüsselten Containerdatei werden die einzelnen Schritte beschrieben, welche das Script *luksformat* aufruft. Soll eine Partition (Festplatte oder USB-Stick) verschlüsselt werden, entfallen die Schritte 1 und 8. Das als Beispiel genutzte Device */dev/loop5* ist durch die Partition zu ersetzen, beispielsweise */dev/hda5* oder */dev/sdb1*.

1. Zuerst ist eine leere Imagedatei zu erstellen. Im Beispiel wird es unter dem Dateinamen *geheim.luks* im aktuellen Verzeichnis erstellt. Der Parameter *count* legt die Größe in MByte fest. Anschließend ist das Image als Loop-Device einzubinden. Das Kommando *losetup -f* ermittelt das nächste freie Loop-Device (Ergebnis: *loop0*).

```
# dd if=/dev/zero of=geheim.luks bs=1M count=100
# losetup -f
/dev/loop0
# losetup /dev/loop0 geheim.luks
```

2. Die ersten 2 MByte sind mit Zufallswerten zu füllen. Das Füllen der gesamten Datei würde sehr lange dauern und ist nicht nötig:

```
# dd if=/dev/urandom of=/dev/loop0 bs=1M count=2
```

3. Anschließend erfolgt die LUKS-Formatierung mit der Festlegung der Verschlüsselung. Die Option *-y* veranlaßt eine doppelte Abfrage des Passwortes, das *keyfile* ist optional

```
# cryptsetup luksFormat -c aes-cbc-essiv:sha256 -s 256 -y
/dev/loop0 [ keyfile ]
```

4. Das formatierte Device wird dem Device-Mapper unterstellt. Dabei wird das zuvor eingegebene Passwort abgefragt. Das Keyfile ist nur anzugeben, wenn es auch im vorherigen Schritt verwendet wurde. Der *<name>* kann frei gewählt werden. Unter */dev/mapper/<name>* wird später auf den verschlüsselten Container zugegriffen:

```
# cryptsetup luksOpen /dev/loop0 <name> [ keyfile ]
```

5. Wer paranoid ist, kann das verschlüsselte Volume mit Zufallszahlen füllen. Der Vorgang kann in Abhängigkeit von der Größe der Containerdatei sehr lange dauern:

```
# dd if=/dev/urandom of=/dev/mapper/<name>
```

6. Ein Dateisystem wird auf dem Volume angelegt:

```
# mkfs.ext3 /dev/mapper/<name>
```

7. Das Volume ist nun vorbereitet und wird wieder geschlossen:

```
# cryptsetup luksClose <name>
```

8. Die Containerdatei wird ausgehängt:

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

2.3 Passwörter verwalten

Mit root-Rechten ist es möglich, bis zu 7 zusätzliche Passwörter für das Öffnen eines Containers festzulegen oder einzelne Passwörter wieder zu löschen.

Für das Hinzufügen eines Passwortes zu der verschlüsselten Imagedatei *geheim.img* im aktuellen Verzeichnis ist diese zuerst einzuhängen, beispielsweise als */dev/loop5*. Dieser Schritt entfällt für Partitionen:

```
# losetup /dev/loop5 geheim.luks
```

Das Hinzufügen eines Passwortes und damit eines neuen Keyslots erfolgt mit folgendem Kommando, wobei als *<device>* beispielsweise */dev/loop5* für die eingebundene Imagedatei oder */dev/hda5* für eine Festplattenpartition anzugeben ist. Das Keyfile ist optional.

```
# cryptsetup luksAddKey <device> [ keyfile ]
```

Ein Keyslot und das zugehörige Passwort können mit folgendem Kommando wieder entfernt werden:

```
# cryptsetup luksKillSlot <device> <slot>
```

Als *<slot>* ist die Nummer des Keyslots anzugeben, eine Zahl von 0 bis 7. Es ist also nötig, sich zu merken, welches Passwort auf welchen Keyslot gelegt wurde. Eine Übersicht, welche Keyslots belegt und welche noch frei sind, liefert *luksDump*:

```
# cryptsetup luksDump <device>
LUKS header information for <device>
...
Key Slot 0: DISABLED
Key Slot 1: ENABLED
    Iterations:
    Salt:

    Key material offset:
    AF stripes:
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

2.4 Verschlüsselten Container öffnen/schließen

Aktuelle Distributionen wie Debian oder Ubuntu erkennen verschlüsselte Partitionen auf Festplatten und USB-Sticks automatisch und fragen die Passphrase ab, sobald das Gerät erkannt wird. Einfach Anschließen, auf den Passwort-Dialog wie im Bild 1 warten - fertig.



Abbildung 1: Passwort-Abfrage für verschlüsselten USB-Stick

Auf der Kommandozeile

Sollte es mit dem automatischem Öffnen des verschlüsselten USB-Sticks nicht funktionieren, kann man auf der Kommandozeile nachhelfen. *pmount* arbeitet mit User-Privilegien und bindet die Partition unter */media* ein. *pmount* kann keine Containerdateien öffnen.

```
> pmount /dev/sda1  
Enter LUKS passphrase:
```

Geschlossen wird der Container mit *pumount*:

```
> pumount /dev/sda1
```

Die Sammlung *pam-mount* enthält zwei weitere Scripte, welche die Arbeit mit verschlüsselten Containerdateien vereinfachen. Wurde außerdem *sudo* entsprechend konfiguriert, stehen die folgenden Kommandos jedem Nutzer zur Verfügung. Eine verschlüsselte Partition (beispielsweise der USB-Stick unter */dev/sda1*) kann mit folgendem Kommando geöffnet und im Verzeichnis */mnt* eingebunden werden:

```
> sudo /sbin/mount.crypt /dev/sda1 /mnt  
Enter LUKS passphrase:
```

Das folgende Kommando öffnet die verschlüsselte Imagedatei *geheim.luks* aus dem aktuellen Verzeichnis und hängt sie unter */mnt* in das Dateisystem ein:

```
> sudo /sbin/mount.crypt geheim.luks /mnt -o loop  
Enter LUKS passphrase:
```

Geschlossen wird der Container mit folgendem Kommando:

```
> sudo /sbin/umount.crypt /mnt
```

Für häufig genutzte Container könnte man einen Menüeintrag oder ein Desktop-Icon anlegen. Dabei ist zu beachten, dass die Option *Im Terminal ausführen* aktiviert wird! Anderenfalls kann man keine Passphrase eingeben.

Für jene, die es genau wissen wollen

Das Öffnen einer Containerdatei auf der Komandozeile erfordert drei Schritte als *root*. Als erstes ist die verschlüsselte Imagedatei einzuhängen. Dieser Schritt entfällt für Partitionen. Im zweiten Schritt ist das verschlüsselte Device dem Device-Mapper zu unterstellen. Der Name kann dabei frei gewählt werden. Im dritten Schritt kann es mit *mount* in das Dateisystem eingehängt werden, beispielsweise nach */mnt*.

```
# losetup /dev/loop5 geheim.luks
# cryptsetup luksOpen /dev/loop5 <name> [ keyfile ]
# mount /dev/mapper/<name> /mnt
```

Das Schließen des Containers erfolgt in umgekehrter Reihenfolge:

```
# umount /mnt
# cryptsetup luksClose <name>
# losetup -d /dev/loop5
```

Komfortabel beim Login

Mit Hilfe des Modules *pam-mount* ist es möglich, das Anmeldepasswort zu nutzen, um standardmäßig beim Login einen oder mehrere Container zu öffnen. Insbesondere für verschlüsselte */home* Partitionen ist dies sinnvoll und komfortabel.

Folgende Konfigurationen sind für einen Crypto-Login anzupassen:

1. **PAM-Konfiguration:** Dem PAM-Dämon ist mitzuteilen, dass er das Modul *mount* zu verwenden hat und das Login-Passwort zu übergeben ist. Gut vorbereitete Distributionen wie Debian und aktuelle Ubuntu(s) benötigen nur einen Eintrag in den Dateien */etc/pam.d/login*, */etc/pam.d/kdm* und */etc/pam.d/gdm*:

```
@include common-pammount
```

2. **pam-mount Modul:** Das Modul wird konfiguriert in der XML-Datei */etc/security/pam_mount.conf.xml*. Am Anfang der Datei findet man eine Section für Volumes, die beim Login geöffnet werden sollen. Im ersten Beispiel wird bei allen Logins die verschlüsselte Partition */dev/hda4* als */home* eingebunden:

```
<volume fstype="crypt" path="/dev/hda4" mountpoint="/home" />
```

Das zweite Beispiel zeigt die Einbindung einer verschlüsselten Containerdatei */geheim.luks* als HOME für den User *pitschie*. Die Containerdatei wird nur geöffnet, wenn *Pitschie* sich anmeldet.

```
<volume user="pitschie" fstype="crypt" path="/geheim.luks"
mountpoint="/home/pitschie" options="loop" />
```

3. **fstab:** Da beim Booten keine Partition nach */home* gemountet werden soll, ist evtl. der entsprechende Eintrag in der Datei */etc/fstab* zu löschen.

2.5 Debian GNU/Linux komplett verschlüsseln

In einem komplett verschlüsselten System sind sowohl die Daten als auch die Systemkonfiguration und Software verschlüsselt. Debian ab Version 4.0r1 (etch) bietet bereits beim Installieren die Option, ein komplett verschlüsseltes System unter Ausnutzung der gesamten Festplatte zu installieren. Lediglich für */boot* bleibt ein kleiner unverschlüsselter Bereich.

Um diese einfache Variante zu nutzen, wählt man im Installations-Dialog *Festplatte partitionieren* die Option *Geführt - gesamte Platte mit verschlüsseltem LVM*. Im folgenden Schritt ist die Passphrase einzugeben, welche das System sichert. Diese Passphrase wird später bei jedem Bootvorgang abgefragt.

Partitionsmethode:

```
Geführt - verwende vollständige Festplatte
Geführt - gesamte Platte verwenden und LVM einrichten
> Geführt - gesamte Platte mit verschlüsseltem LVM
Manuell
```

Ubuntu-Nutzer können die **alternate desktop cd** nutzen, die kein Live-System enthält, dafür aber mehr Optionen für die Installation bietet. Die Standard-Edition von Ubuntu bietet dieses Feature nicht!

Ein vollständig verschlüsseltes System macht es böswilligen Buben sehr schwer, bei einem *heimlichen Hausbesuch* die Software zu manipulieren und einen Trojaner zu installieren. Es ist jedoch nicht unmöglich. Wer noch einen Schritt weiter gehen will, erstellt nach der Installation eine bootfähige CD-ROM mit einer Kopie des sauberen Verzeichnis */boot* und bootet in Zukunft immer von der CD. (Oder man geht zum Psychater und lässt seine Paranoia behandeln.)

Man sollte nicht aus Zeitgründen auf ein Überschreiben der alten Daten mit Zufallszahlen verzichten. Um die Position verschlüsselter Daten auf der Platte zu verstecken und Daten der alten Installation zu vernichten, bietet die Installationsroutine die Option, den Datenträger mit Zufallszahlen zu überschreiben. Das dauert zwar einige Zeit, ist aber ein sinnvolles Feature.

2.6 HOME-Verzeichnis verschlüsseln

Die Verschlüsselung der persönlichen Daten im \$HOME-Verzeichnis bieten alle Linux-Distributionen bei der Installation an. Wer keine Kompletterschlüsselung nutzen möchte, sollte zumindest diese Option aktivieren. Der Container mit den verschlüsselten Daten wird beim Login automatisch geöffnet. Die Nutzung ist vollständig transparent. Bei Verlust des Laptops sind die Daten jedoch geschützt.

2.7 SWAP und /tmp verschlüsseln

Das */tmp*-Verzeichnis und der SWAP Bereich können unter Umständen persönliche Informationen enthalten, die im Verlauf der Arbeit ausgelagert wurden.

Wenn eine komplette Verschlüsselung des Systems nicht möglich ist, sollte man verhindern, dass lesbare Datenrückstände in diesen Bereichen verbleiben.

Das Verzeichnis */tmp* kann man im RAM des Rechners ablegen, wenn dieser hinreichend groß dimensioniert ist. Mit dem Ausschalten des Rechners sind alle Daten verloren. Um diese Variante zu realisieren bootet man den Rechner im abgesicherten Mode, beendet die grafische Oberfläche (X-Server) und löscht alle Dateien in */tmp*. In der Datei */etc/fstab* wird folgender Eintrag ergänzt:

```
tmpfs /tmp tmpfs defaults,size=256m 0 0
```

Die Bereiche SWAP und */tmp* können im Bootprozess als verschlüsselte Partitionen mit einem zufälligen Passwort initialisiert und eingebunden werden. Mit dem Ausschalten des Rechners ist das Passwort verloren und ein Zugriff auf diese Daten nicht mehr möglich.

Achtung: Suspend-to-RAM und Suspend-to-Disk funktionieren mit einer verschlüsselten SWAP-Partition noch nicht.

Debian GNU/Linux

Debian und Ubuntu enthalten ein Init-Script, welches eine einfache Verschlüsselung von SWAP und */tmp* ermöglicht, wenn diese auf einer eigenen Partition liegen.

In der Datei */etc/crypttab* sind die folgenden Zeilen einzufügen, wobei */dev/hda5* und */dev/hda8* durch die jeweils genutzten Partitionen zu ersetzen sind:

```
cryptswp /dev/hda5 /dev/urandom swap
crypttmp /dev/hda8 /dev/urandom tmp
```

In der Datei */etc/fstab* sind die Einträge für swap und */tmp* anzupassen:

```
/dev/mapper/cryptswp none swap sw 0 0
/dev/mapper/crypttmp /tmp ext2 defaults 0 0
```

Anschließend ist der Rechner neu zu booten und beide Partitionen sind verschlüsselt.

Achtung: Die Partition für */tmp* darf kein Dateisystem enthalten! Soll eine bereits verwendete */tmp*-Partition verschlüsselt werden, ist diese erst einmal nach dem Beenden des X-Servers(!) zu dismounten und zu überschreiben:

```
# umount /tmp
# dd if=/dev/zero of=/dev/hda8
```