Verschlüsseltes Home-Filesystem unter Red Hat, Debian und Gentoo

Lese-Schutz

Ganze Filesysteme verschlüsseln schützt deren Daten auch, wenn jemand die Festplatte klaut oder den Rechner von Diskette bootet. Während Suse-Anwender bereits bei der Installation die Verschlüsselung aktivieren können, ist bei anderen Distributionen etwas Handarbeit angesagt. Christian Ney

Laptop-Anwender haben ihre Daten immer dabei – das ist meist praktisch, aber schlecht, wenn der mobile Computer gestohlen wird. Die darauf gespeicherten Files geraten dann schnell in falsche Hände. Selbst wer das Gerät nur zur Reparatur einschickt, muss damit rechnen, dass ein neugieriger Techniker gerade zu viel Zeit hat und in den privaten Dateien schnüffelt.

Für den Schutz sensibler Daten bietet Kryptographie einen Ausweg. Das gelingt ohne großen Aufwand: Der GNU Privacy Guard (GPG, [1]) verschlüsselt auf Kommando einzelne Dateien. Um viele Files zu schützen ist dieses Vorgehen aber zu unhandlich. Bequemer ist es, ganze Dateisysteme zu verschlüsseln. Suse zeigt schon seit Jahren (siehe Abbildung 1), wie benutzerfreundlich sich das umsetzen lässt. Bereits bei der Installation legt es auf Wunsch Krypto-Filesysteme an. Bei anderen Distributionen lässt sich dies mit etwas Handarbeit nachbilden und in kleinen Teilen sogar noch verbessern. Besonders die Metadistribution Gentoo macht es dem Anwender sehr einfach.

Verschlüsseltes Home

Als Beispiel dient im Folgenden das »/home«-Filesystem. Es soll trotz Verschlüsselung beim Systemstart automatisch gemountet werden, der Schlüssel soll auf einem USB-Stick liegen. Dieser Key dient als Ersatz für das sonst übliche Passwort, das bei der Initialisierung des Cryptoloop-Device als Basis für die Verschlüsselung dient. Der Vorteil: Der Schlüssel besteht aus Zufallswerten und ist um einiges länger als die meisten Passwörter, die sonst recht Erfolg versprechenden Dictionary-Attacken scheitern daran. Ein Angreifer müsste wirklich alle möglichen Schlüssel probieren (Brute-Force-Angriff), das Gelingen ist bei ausreichender Schlüssellänge praktisch ausgeschlossen.

Unter Linux gibt es mehrere Möglichkeiten, komplette Dateisysteme zu verschlüsseln. Die Loopback-Encryption, die sich der kryptographischen Funktionen des Kernels bedient (Crypto-API [2], früher International Patch), bietet die meisten Algorithmen zur Auswahl und ist im Vergleich zu anderen Lösungen sehr komfortabel zu bedienen.

Zwischenschicht

Die Loopdevices arbeiten als Zwischenschicht zwischen dem Blockdevice (zum Beispiel »/dev/hda1«) und dem darauf enthaltenen Filesystem. Im Normalfall greift das Filesystem direkt auf das Blockdevice zu (Abbildung 2, oben). Das Loopback-Device verhält sich zwar wie ein Blockdevice, es gibt die Daten aber an ein normales Device oder an eine Datei weiter (Abbildung 2, Mitte). Dieses Verfahren kennt jeder, der ein ISO-Image schon mal mit der »loop«-Option gemountet hat.

Loopback-Devices sind auch in der Lage, die durchgereichten Daten zu ändern. Bei Cryptoloop greift der Kernel in jeden Schreib- und Lesevorgang ein, um die Daten zu ver- beziehungsweise zu entschlüsseln (Abbildung 2, unten). Der Anwender bemerkt – bis auf eine initiale Passwortabfrage – nichts. Auch die Passwortabfrage lässt sich vermeiden, wenn der Schlüssel auf einem USB-Stick untergebracht ist. Leider sind die erforderlichen Patches nicht standardmäßig in jedem Kernel enthalten.

- Unter Debian Woody ist es nötig, den Kernel zusammen mit dem Paket »kernel-patch-int« neu zu bauen.
- Red Hat 8 unterstützt bereits Cryptoloop-Filesysteme, es stellt per Default aber ausschließlich das AES-Modul zur Verfügung.
- Der Standardkernel von Gentoo »gentoo-sources« enthält seit längerem das International Patch, beim Kompilieren müssen die nötigen Optionen aber ausgewählt sein (siehe Abbildung 3). Der aktuellen 2.4.22-Version fehlt allerdings noch das »cryptoloop«-Patch.

Ob eine Distribution die benötigten Patches funktionsfähig mitbringt, ist leicht zu erkennen: eine verschlüsselte Datei erzeugen und versuchen, sie als normales Filesystem einzubinden. Dazu erstellt der Admin zunächst mit Hilfe von »dd« eine Datei, die später das verschlüsselte Filesystem enthalten wird. Sie sollte für den Test 20 MByte groß sein.

Loopdevice anlegen

Das »losetup«-Kommando erzeugt aus der Datei ein Loopdevice. Es fragt dabei nach einem Verschlüsselungs-Passwort. Für den Test ist ein Passwort ausreichend, später kommt ein zufälliger Key zum Einsatz. Beim Eingeben des Passworts ist Vorsicht geboten: »losetup« fragt es nur genau ein Mal ab. Wer sich hier vertippt, kommt später nicht mehr an seine verschlüsselten Daten. Zum Abschluss muss der Tester noch ein Filesystem auf dem Loopdevice anlegen, das Dateisystem mounten und ein paar Files darauf ablegen:

```
dd if=/dev/zero of=/root/cryptfile ➥
  bs=1024k count=20
losetup -e aes -k 128 /dev/loop0 ➥
  /root/cryptfile
mkfs -t ext2 -m 0 /dev/loop0
mount -t ext2 /dev/loop0 /mnt
cp /bin/{mv,rm} /mnt
```

Bringt bereits das »losetup«-Kommando eine Fehlermeldung, ist der Kernel vermutlich noch nicht auf die Gegebenheiten verschlüsselter Filesysteme eingerichtet. Ihm fehlt Crypto-API: [3] erklärt, wie der Kernel zu dieser Funktionalität kommt. Es ist dazu nicht nötig, den kompletten Kernel neu zu übersetzen, denn Crypto-API besteht nur aus Modulen, die der Kernel im laufenden Betrieb lädt. Auch für den Betrieb mit USB-Sticks muss der Kernel gewappnet sein: Die entsprechenden Optionen sind »USB-Mass-Storage«, »SCSI-Disk-Support« und »SCSI emulation support«. Die Kernel der meisten aktuellen Distributionen enthalten die entsprechenden Module bereits, das erneute Kompilieren kann damit entfallen.

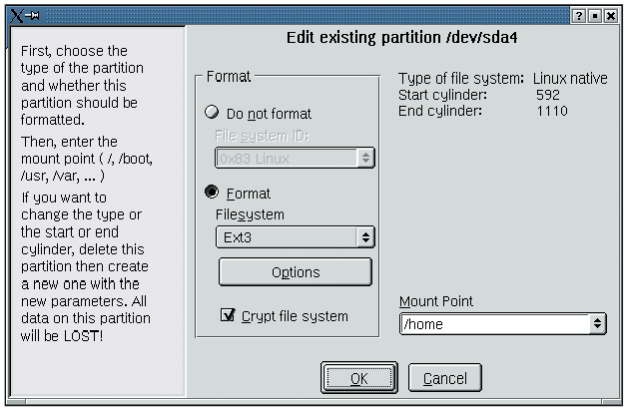


Abbildung 1: Bei Suse (hier Version 7.3) ist ein verschlüsseltes Home-Filesystem nur eine Frage des richtigen Häkchens im Yast.

Userspace-Tools

Über die im Kernel ablaufende Ver- und Entschlüsselung hinaus müssen Userspace-Programme die Krypto-Filesysteme vorbereiten und später mounten. Dazu sind »losetup« und »mount« (aus dem Paket »util-linux«) entsprechend einzurichten.

- Das Util-Linux-Paket von Debian Woody ist bereits für verschlüsselte Filesysteme vorbereitet, ein Patch ist nicht nötig.
- Red Hat 8 bringt ein ungepatchtes Util-Linux-Paket mit, hier ist Handarbeit angesagt. Die Schritte sind auf [4] sehr gut erklärt.
- Gentoo fügt dem Util-Linux-Paket bei gesetztem Use-Flag »+crypt« die Funktionen hinzu, um mit Krypto-Filesystemen zu arbeiten.

Der erste Schritt vor dem Praxiseinsatz ist ein Backup der Daten. Da man im weiteren Ablauf das komplette »/home«-Filesystem unwiderruflich löscht, wären die darauf enthaltenen Daten sonst verloren. Eine gute Gelegenheit, sein Heim mal wieder aufzuräumen.

Ein verschlüsseltes Filesystem ist aber nur dann ein echter Sicherheitsgewinn, wenn nicht noch Reste der alten Partition auf der Platte liegen. Also gilt es, die Daten gründlich von der Platte zu putzen: Einfaches Formatieren oder Löschen genügt nicht. Die Originalfiles liegen im (hoffentlich geprüften) Backup und werden später in das neue Krypto-Filesystem eingespielt.

Das alte Filesystem unbrauchbar machen geht recht einfach mit Zufallswerten: Der Admin unmountet das »/home«-Filesystem und entleert den Zufallszahlengenerator des Kernels in die Partition. In den folgenden Beispielen liegt »/home« auf »/dev/hda6«. Wer Wert auf beste Zufälligkeit legt und viel Zeit hat, sollte statt »/dev/urandom«

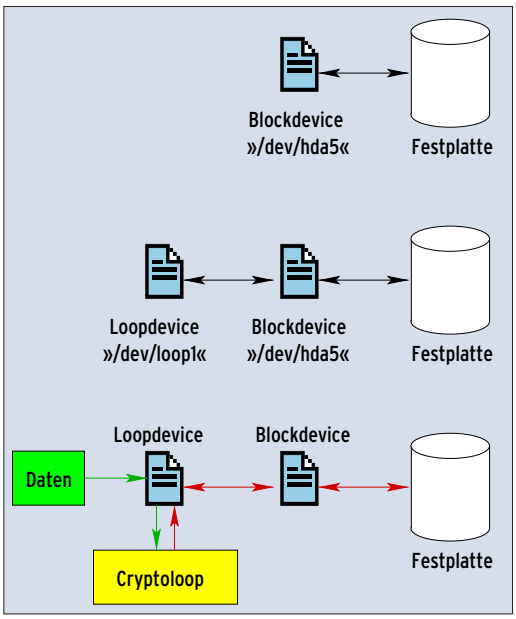


Abbildung 2: Der Kernel gibt Zugriffe auf ein Blockdevice an die Festplatte weiter (oben). Das Loopdevice (Mitte) kommt noch vor dem Blockdevice zum Zug. Mit Loopback-Verschlüsselung (unten) landen keine Klartextdaten mehr auf der Festplatte.

besser den Zufallsgenerator »/dev/random« verwenden (siehe Kasten „Guter Zufall braucht Zeit“).

```
dd if=/dev/urandom of=/dev/hda6 bs=4k ➥
  conv=notrunc
```

Dieses Kommando überschreibt das gesamte Blockdevice mit zufälligen Werten. Damit ist es später schwerer, zwischen Datenmüll und den verschlüsselten Daten zu unterscheiden. Je nachdem, wie viel Entropie im Zufallszahlengenerator übrig ist, und je nach Größe des Filesystems kann das Überschreiben viel Zeit in Anspruch nehmen.

Diese Vorgehensweise reicht allerdings noch nicht aus, um professionelle Datenretter davon abzuhalten, die alten Daten wiederherzustellen. Es ist immer noch genug Restmagnetismus auf der Festplatte vorhanden [5]. Wer paranoid genug ist, sollte den Vorgang also mehrfach wiederholen.

Schlüssel zum Filesystem

Als Nächstes gilt es, die Schlüsseldatei zu erzeugen. Dieser Key dient zum Ver- und Entschlüsseln der Daten im Krypto-Filesystem. Um zu vermeiden, dass sich der Schlüssel per Dictionary-Attacke knacken lässt, sollte er möglichst lang und zufällig sein. Nur so kann er seine Vorteile gegenüber den meist erheblich kürzeren Passwörtern ausspielen. Da die Datei auf dem USB-Stick landen soll, muss dieser gemountet sein, im Folgenden auf »/mnt/usb/«:

```
head -c 2880 /dev/random | uuencode -m -|➥
  head -n 65 | tail -n 64 | ➥
  gpg --symmetric -a > /mnt/usb/key.gpg
```

Dieser Aufruf liest zunächst 2880 Byte aus »/dev/random« und benutzt dann »uuencode«, um die Zufallswerte Base64-konform zu kodieren. Die Kommandos »head« und »tail« entfernen die Klartextausgaben von Uuencode.

Das Ergebnis hat schon recht gute Zufallseigenschaften, die abschließende Verschlüsselung mit GPG verbessert die Qualität weiter. Dabei verlangt GPG eine Passphrase, die als symmetrischer Schlüssel dient. Nur mit diesem Key sind die Daten zu entschlüsseln. Geht die Schlüsseldatei verloren, sind auch die Daten auf der Festplatte wertlos. Es ist also erforderlich, ein Backup des Keys sicher aufzubewahren.

Die zusätzliche Verschlüsselung mit GPG eignet sich auch dazu, den Schlüssel vor unbefugten Benutzern zu schützen: Statt die verschlüsselte Version zu verwenden, könnte man das Keyfile mit Hilfe der Passphrase wieder entschlüsseln und die resultierenden Daten als Schlüssel nutzen. Ein Angreifer müsste dann nicht nur den USB-Stick in seinen Besitz bringen, sondern zusätzlich die GPG-Passphrase kennen, um an die Daten zu gelangen.

Die Vorbereitungen sind damit abgeschlossen, als Nächstes ist das Loopdevice einzurichten. Dazu sind ein paar Überlegungen zu Algorithmus und Schlüssellänge angebracht. Das folgende Beispiel nutzt Twofish [6], dieser Algorithmus kommt seit einigen Jahren in vielen Produkten zum Einsatz und wurde von vielen Krypto-Experten untersucht, er gilt als ausgreift (siehe Kasten „Wahl des Algorithmus“).

```
losetup -e twofish -k 128 /dev/loop0 ➥
  /dev/hda6 -p 0 < /mnt/usb/key.gpg
```

Diese Kommando initialisiert das Loopdevice »/dev/loop0« mit dem Twofish-Algorithmus (Schlüssellänge: 128 Bit). Das Loopdevice legt die verschlüsselten Daten im Blockdevice »/dev/hda6« ab. Als Basis für den Schlüssel erwartet das Programm ein Passwort, das es in diesem Aufruf nicht von der Tastatur, sondern von der Standardeingabe erhält: »-p 0 < /mnt/usb/key.gpg«.

Testphase

Das Loopdevice ist nun bereit ein beliebiges Filesystem aufzunehmen. Journaling-Filesysteme können aber in ungünstigen Fällen Probleme bereiten: Eventuell schreibt das Loopdevice die Daten in einer anderen Reihenfolge auf die Platte, als es das Filesystem erwartet. Das Journal ist dann weitgehend nutzlos und kann nach Abstürzen zu üblen Filesystem-Inkonsistenzen führen.

```
mkfs -t ext2 /dev/loop0
mount /dev/loop0 /mnt
```

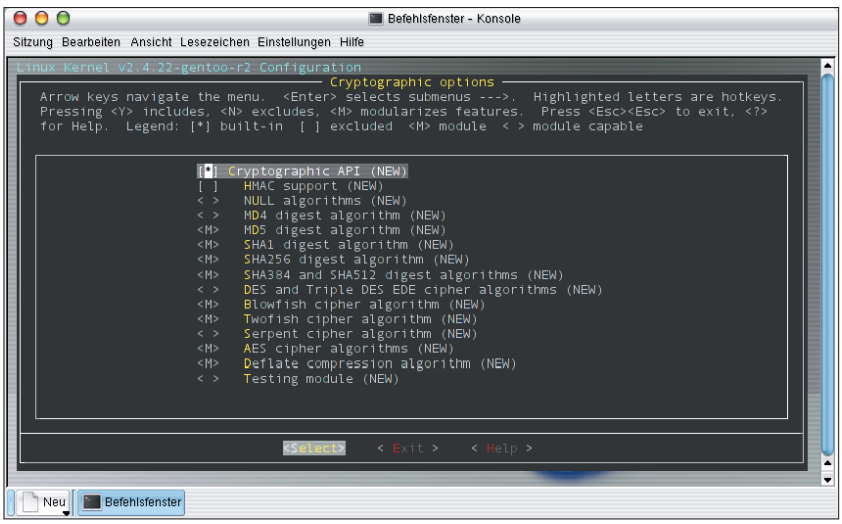


Abbildung 3: Neuerdings enthält bereits der Standardkernel (ohne zusätzliche Patches) Verschlüsselungsoptionen. Dazu sind »Cryptographic API« und die gewünschten Algorithmen auszuwählen.

Guter Zufall braucht Zeit

Die beiden Devicefiles »/dev/random« und »/dev/urandom« sind Schnittstellen zum Zufallszahlengenerator des Kernels. Er sammelt Ereignisse mit möglichst hohem Grad an Zufall und fügt diese Daten zu einem Entropiepool hinzu. Jeder Zugriff auf »/dev/random« reduziert die Menge der im Pool vorhandenen Entropie.

Falls nicht mehr genügend Entropie vorhanden ist, blockiert der Kernel den Lesevorgang, bis er wieder ausreichend Zufälligkeit gesammelt hat. Daher liefert das Device Zufallswerte mit hoher Qualität, viele kryptographische Funktionen (etwa OpenSSL, GPG und andere) nutzen diese Werte.

Das Device »/dev/urandom« hingegen gibt die gesamte angeforderte Datenmenge zurück, ohne zu prüfen, ob ausreichend Entropie vorhanden ist. Es kann daher passieren, dass Angreifer Rückschlüsse auf die zurückgegebenen Daten ziehen können. Für einfache Aufgaben genügen die Werte aber.

Bei diesem Beispiel kommt Ext 2 zum Einsatz. Nach dem Mounten ist es ratsam, ein paar Daten testweise auf dem neuen Filesystem abzulegen. Danach das Filesystem wieder entladen und das Loopdevice deaktivieren:

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

Um späteren bösen Überraschungen vorzubeugen, sollte man das Aufsetzen des Loopdevice und das Mounten des Filesystems mehrmals testen. Noch liegen keine wichtigen Daten im Krypto-Filesystem, im Zweifelsfall kann man erneut von vorn anfangen.

Alles startbereit

Abschließend sind noch einige Änderungen am System nötig, da der gesamte Vorgang inklusive Mounten des USB-Sticks automatisiert ablaufen soll. Einige Distributionen sind bereits für einen solchen Einsatz vorbereitet. Beispielsweise liefert Gentoo das Init-Skript »cryptoloop« mit. Wie es zu verwenden ist, erklären die Kommentare in diesem Skript. Die folgende generische Lösung ist zwar nicht übertrieben elegant, sie sollte aber auf den meisten verbreiteten Distributionen laufen. Die einfachste Variante benutzt ein vorhandenes Init-Skript, meist ist »/etc/init.d/rc.local« eine gute Wahl. Eleganter wäre ein eigenes Init-Skript, wobei allerdings die Startreihenfolge zu beachten ist.

Ein Beispiel einer »rc.local« für die Kernelversionen ab 2.4.22 ist in Listing 1 zu sehen. Die Namen der Algorithmen-Module haben sich mit Erscheinen des 2.4.22er Kernels geändert, für ältere Versionen ist dem Namen ein »cipher-« voranzustellen. Zeile 9 lautet dann »/sbin /modprobe cipher-twofish«.

Das Mounten des USB-Sticks wäre mit Hilfe des Automounters »supermount« [7] noch komfortabler. Hierzu ist ein weiteres Kernelmodul erforderlich, das aber recht einfach zu installieren ist. Zusätzlich ist ein spezieller Eintrag in »/etc/fstab« nötig:

```
/mnt/usb  /mnt/usb  supermount ➥
  sync,rw,fs=auto,dev=/dev/sda1 0 0
```

Das spart das explizite Mounten im Init-Skript. Supermount übernimmt auf Wunsch zum Beispiel auch das automatische Einbinden von CD-ROMs.

Schotten dicht

Der ganze Aufwand ist aber nutzlos, wenn vertrauliche Daten im Klartext an anderen Stellen auf der Platte zu finden sind. Im schlimmsten Fall lagert der Kernel genau jenen Speicherbereich, in dem der Schlüssel liegt, in den Swapbereich aus. Eine weitere Schwachstelle wäre ein Coredump, der unter Umständen den Speicherbereich eines Prozesses nach einem massiven Fehler auf die Festplatte schreibt.

Beide Löcher sind aber recht einfach zu stopfen. Im Swapbereich liegen im Normalfall keine Daten, die nach einem Neustart noch benötigt werden. Es ist daher ohne weiteres möglich, diesen Bereich transparent zu verschlüsseln, am besten sogar mit einem zufälligen Schlüssel, den niemand kennt.

Der Nachteil dieser Methode ist eine zusätzliche minimale Verzögerung beim Booten: Da das System den Swap nach jedem Booten mit einem anderen Schlüssel verschlüsselt, muss es ihn jedes Mal neu initialisieren. Hierzu ist das

Wahl des Algorithmus

Ein guter Anhaltspunkt für die Auswahl eines geeigneten Algorithmus bietet der Vergleich [8] der AES-Finalisten (Advanced Encryption Standard), geschrieben unter anderem vom Krypto-Papst Bruce Schneier. Er zeigt, dass Rijandel und Twofish die schnellsten Algorithmen sind, sie werden auch am häufigsten benutzt. Gerade Rijandel verliert aber mit zunehmender Schlüssellänge an Geschwindigkeit.

Twofish bietet ein Quäntchen mehr Sicherheit [9]. Rijandel ist zwar etwas jünger, aber es gibt bereits theoretische Angriffe gegen diesen Algorithmus. Trotzdem sollte auch der AES-Gewinner Rijandel völlig ausreichenden Schutz bieten.

Ein weiterer Punkt ist die Auswahl einer sinnvollen Schlüssellänge. Zur Wahl stehen 128, 192 oder 256 Bit. Je größer die Länge, umso sicherer ist der Algorithmus gegen Brute-Force-Angriffe, allerdings sinkt die Geschwindigkeit. Werden grundlegende Schwächen in der verwendeten Verschlüsselung gefunden, nützt jedoch auch die größte Schlüssellänge wenig [10]. 128 oder 192 Bit gelten daher als ausreichend.

Init-Skript anzupassen, das den Swapbereich einbindet. Ein »grep swapon /etc/init.d/*« ermittelt das zu ändernde Skript. Vor der Zeile, die »swapon« enthält, sind zwei Zeilen einzufügen:

```
/bin/dd if=/dev/urandom bs=1 count=16 | ➥
  /sbin/losetup -e twofish -k 128 -p 0 ➥
  /dev/loop1 /dev/hda2
/sbin/mkswap /dev/loop1
/sbin/swapon -a
```

Diese Kommandos erzeugen ein weiteres Loopdevice. Das Passwort besteht hier aus 16 zufälligen Bits. Da sich dieser Schlüssel bei jedem Neustart ändert, ist der Swapbereich jedes Mal durch »mkswap« neu zu initialisieren.
Zusätzlich ist in »/etc/fstab« der Eintrag für das Swap-Blockdevice (in diesem Fall »/dev/hda2«) durch das entsprechende Loopdevice (hier »/dev/loop1«) zu ersetzen. Wer auf Nummer Sicher gehen will, deaktiviert nun kurzzeitig per »swapoff -a« den Swap, überschreibt ihn mit Hilfe von »dd« und »/dev/urandom« und führt danach die drei eben beschriebenen Schritte manuell aus.

Kein Coredump

Bei fast allen Distributionen ist das Schreiben von Coredumps standardmäßig deaktiviert. Überprüfen lässt sich das Verhalten per »ulimit -a | grep core«: Es ermittelt die maximale Anzahl an Blöcken, die ein Core-File belegen darf. Wenn der ausgegebene Wert eine »0« ist, verzichtet das System auf Coredumps. Ist der Wert größer als »0«, lässt er sich global in »/etc/security/limits.conf« deaktivieren. Der passende Eintrag lautet:

```
*  soft  core  0
```

Er ist in den meisten Fällen schon vorhanden, aber auskommentiert.

Manchmal inkompatibel

Wer den frisch erschienenen Kernel 2.6 einsetzen will, sollte die gesamte Prozedur erst dann durchführen, wenn er den neuen Kernel ausschließlich nutzt. Die mit Kernel 2.4 erstellten Cryptoloop-Devices lassen sich mit dem neuen Kernel leider nicht mehr mounten.
Selbst ein Update innerhalb der 2.4er Kernelserie kann dazu führen, dass der neue Kernel die verschlüsselten Daten nicht mehr entschlüsselt. Am kritischsten ist ein Update auf Versionen ab 2.4.22: Seit dieser Version enthält der offizielle Kernel auch Krypto-Code, das International Patch ist seither unnötig. Leider betreffen die Änderungen nicht nur die Namen der jeweiligen Module (beispielsweise »aes« statt »cipher-aes«), sondern sind so tiefgreifend, dass die alten Filesysteme nicht mehr lesbar sind. Glücklicherweise scheint ein Umstieg vom 2.4.22er Kernel auf nachfolgende Versionen der 2.4-Reihe keine Probleme mehr zu bereiten.

Durch die Flexibilität des Cryptoloop-Verfahrens sind Erweiterungen denkbar, die das System noch flexibler oder sicherer machen. Wer auf dem USB-Stick zur Kompatibilität mit Windows ein FAT-Dateisystem betreiben muss, könnte zusätzlich ein eigenes kleines Ext-2-Filesystem anlegen, auf dem er das Keyfile ablegt. Damit ist der Schlüssel vor versehentlichen Änderungen geschützt.

Schnell genug

Trotz der Ver- und Entschlüsselung ist die Geschwindigkeit der Lese- und Schreibvorgänge auf üblichen Laptops kaum von einem nicht verschlüsselten Filesystem zu unterscheiden. Nur die etwas erhöhte CPU-Last deutet auf den hohen Rechenaufwand hin. *(fjl)* ■

Listing 1: Startskript

```
01 #!/bin/sh -e
02
03 case "$1" in
04   start)
05     /sbin/modprobe sd_mod
06     /sbin/modprobe usb-storage
07     /sbin/modprobe cryptoapi
08     /sbin/modprobe cryptoloop
09     /sbin/modprobe twofish
10     /bin/mount -t vfat /dev/sda1 /mnt/usb
11     /sbin/losetup -e twofish -k 128 /dev/loop0 /home
   -p 0 < /mnt/usb/key.gpg
12     /bin/mount /dev/loop0 /home
13     /bin/umount /dev/sda1
14     ;;
15   stop)
16     /sbin/losetup -d /dev/loop0
17     /bin/umount /dev/loop0
18     ;;
19 esac
```

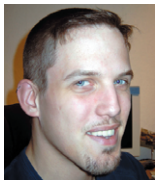
Mögliche Alternative: Loop-AES

Loop-AES [11] wurde eigens für verschlüsselte Filesystemen entwickelt. Es ist in der Lage, auch den Swapbereich ohne besondere Schritte zu verschlüsseln. Loop-AES nutzt im Normalfall den Gewinner der AES-Ausschreibung Rijandel, verwendet mittlerweile auf Wunsch aber auch andere Algorithmen (Blowfish, Twofish und Serpent).
Wie bei Crypto-API ist es auch bei Loop-AES nicht erforderlich, einen neuen Kernel zu bauen, die benötigten Funktionen lassen sich als Modul einbinden. Das Util-Linux-Paket braucht in den meisten Distributionen aber ein Patch. Beides ist sehr gut unter [12] beschrieben.
Trotz der Flexibilität der Crypto-API-Lösung gibt es ein paar Bereiche, in denen Loop-AES das Vorhaben vereinfacht. Spätestens wenn ein Admin verschiedenen Usern den Zugriff per GPG-Key ermöglichen will, sollte er einen Blick auf Loop-AES werfen.

Infos

[1] GPG: [http://www.gnupg.org]
[2] Crypto-API: [http://www.kerneli.org/index.phtml]
[3] Crypto-API installieren: [http://www.kerneli.org/howto/node2.php]
[4] Patchen von »util-linux«: [http://www.linuxsecurity.com/docs/HOWTO/Encryption-HOWTO/Encryption-HOWTO-4.html#losetup]
[5] Peter Gutmann, „Secure Deletion of Data from Magnetic and Solid-State Memory“: [http://www.cs.auckland.ac.nz/~pgut001/pubs/secure_del.html]
[6] Twofish-Dokumentation: [http://www.schneier.com/paper-twofish-paper.pdf]
[7] Supermount: [http://supermount-ng.sourceforge.net]
[8] Vergleich der AES-Finalisten: [http://www.schneier.com/paper-aes-performance.pdf]
[9] Twofish und Rijandel: [http://www.schneier.com/paper-twofish-final.pdf]
[10] Schlüssellängen erklärt: [http://mitglied.lycos.de/cthoeing/crypto/keylen.htm]
[11] Loop-AES: [http://loop-aes.sf.net]
[12] Loop-AES-Readme: [http://loop-aes.sourceforge.net/loop-AES.README]

Der Autor



Christian Ney arbeitet als Unix- und Firewall-Administrator bei einer Regionalfluggesellschaft und wirkt in seiner Freizeit in mehreren Open-Source-Projekten mit.