Festplattenverschlüsselung

Sebastian Berschneider Robby Zippel

Seminar Konzepte von Betriebssystemkomponenten
Lehrstuhl 4 des Department Informatik
Universität Erlangen-Nürnberg
21. Juli 2010

Inhaltsverzeichnis

1 Einleitung

Mit steigender Verbreitung von mobilen Endgeräten wie Laptops, Tablet-PCs und Netbooks wurde das mobile Leben revolutioniert. Man hat seine Daten überall dabei, kann im Park oder in einem Café an seinem Projekt für die Firma oder Universität arbeiten. Sollte dieses Gerät aber einmal gestohlen werden, ist der Ärger groß. Während man sich natürlich über die verlorene Hardware und den damit verbundenen Kostenfaktor ärgert, wird man eventuell erst im zweiten Moment merken, dass der Dieb physischen Zugang zu dem Laptop und damit auch Zugang zu sämtlichen Daten auf der Festplatte hat.

Viele denken dabei, dass der Dieb niemals an seine privaten Daten kommen kann, denn man hat ja in weiser Vorraussicht ein Login-Passwort vergeben. Dieses kann allerdings ohne Probleme z.B. durch den Einsatz einer Linux-Live-Distribution, also eines von einer CD oder DVD gestarteten Betriebssystems, ausgehebelt werden. Genauso verhält es sich mit eventuell vergebenen Dateiberechtigungen. Sollte man seine privaten Daten verschlüsselt haben (z.B. mit GPG, PGP etc.), so sind zwar die eigentlichen privaten Daten verschlüsselt, der zum Entschlüsseln nötige Private Key liegt allerdings trotzdem schutzlos auf der Festplatte. Genauso verhält es sich mit SSH-Keys zum Einloggen auf fremden Servern.

Selbst verschlüsselte Daten können unverschlüsselt noch im Auslagerungsspeicher liegen, da die meisten Programme nur entschlüsselte Dateien öffnen können und diese dann im Auslagerungsspeicher zwischenlagern. Ein Angreifer könnte also den Auslagerungsspeicher auslesen und so ebenfalls Zugriff auf private Daten erlangen.

Im Folgenden sollen deswegen zwei Programme vorgestellt werden, die diese Problematik lösen sollen. TrueCrypt [2] zum Einen verschlüsselt eine komplette Partition oder Festplatte blockweise, eCryptFS [3] dagegen verschlüsselt einen speziellen Ordner und die Dateien, die sich darin befinden. Beide Verfahren haben ihre Vor- und Nachteile, auf die nun genauer eingegangen werden soll.

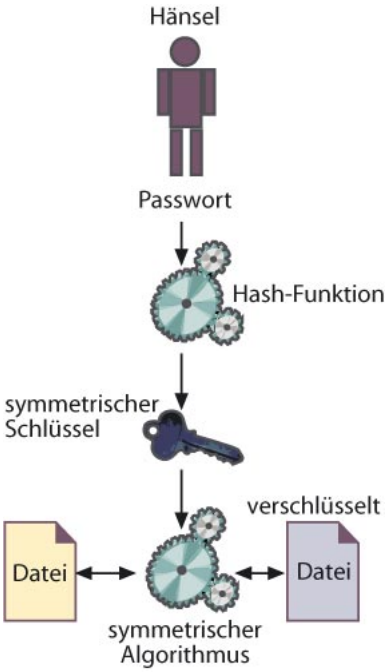

Abbildung 1: Authentifizierung [1]

2 Authentifizierung

Um die Funktionen der beiden Tools TrueCrypt und eCryptFS besser zu verstehen, ist es hilfreich, zuerst die Authentifizierung [1] als gemeinsame Grundlage zu beleuchten.

Für das Verschlüsseln muss meist ein Passwort angegeben werden, das vom Benutzer frei gewählt werden kann. Dieses Passwort selbst ist allerdings nicht der Festplattenschlüssel, der für die Verschlüsselung verwendet wird. Das angegebene Passwort wird zuerst mit einer Hash-Funktion (z.B. SHA-1, MD5) in eine Zeichenfolge umgewandelt, welche dann der symmetrische Schlüssel für einen symmetrischen Algorithmus darstellt (für ein besseres Verständnis siehe auch Abbildung 1). Zusätzlich zu diesem einfachen Authentifizierungs-Verfahren gibt es auch die Möglichkeit über Zertifikatsverfahren. Dies ermöglicht einen flexibleren Zugriff auf die verschlüsselten Daten u.a. in Firmen durch die asymmetrische Verwaltung der Festplattenschlüssel. Durch einen Public Key wird dadurch die Verschlüsselung durchgeführt, zum entschlüsseln ist ein Private Key notwendig. Dieses Verfahren ist zum Beispiel innerhalb einzelner Abteilungen einer Firma sehr nützlich.

3 TrueCrypt

Die folgenden Informationen sind gänzlich aus der TrueCrypt-Dokumentation entnommen und aufgearbeitet worden. [2]

3.1 Methoden

Eine verschlüsselte Partition wird von TrueCrypt blockweise ver- und entschlüsselt, was bedeutet, dass bei jedem Lese- und Schreibzugriff auf die Festplatte CPU-Last erzeugt wird. Die zu entschlüsselnde Partition wird dann als virtuelles Laufwerk in das Betriebssystem eingebunden. TrueCrypt bietet die Möglichkeit, eine komplette Festplatte oder eine Partition zu verschlüsseln. Dazu muss man anmerken, dass diese Festplatte bzw. Partition nicht die System-Partition sein darf, die das Betriebssystem enthält. Diese kann nur mit einem speziellen Modus verschlüsselt werden (System-Verschlüsselung/Pre-Boot-Authentication). TrueCrypt kann außerdem sogenannte Container verwalten, welche wie eine verschlüsselte TrueCrypt-Partition aufgebaut sind und ebenfalls wie eine Partition als virtuelles Laufwerk eingebunden werden, aber als eine einzelne Datei auf der Festplatte gespeichert werden. Man kann also auf seiner unverschlüsselten Partition neben unsensiblen Daten einen Container speichern, der die sensiblen Daten enthält.

TrueCrypt bietet die Verschlüsselungs-Algorithmen AES, Twofish und Serpent an. Zusätzlich sind verschiedene Konkatenationen der einzelnen Algorithmen möglich. Das bedeutet, dass die Partition z.B. zuerst mit AES, dann mit Twofish, dann mit Serpent verschlüsselt werden kann, um eine extrem hohe Sicherheit zu gewährleisten. Allerdings leidet die Geschwindigkeit bei der Entschlüsselung dieser Daten merklich darunter.

Zusätzlich gibt es noch Hash-Algorithmen als Misch-Funktionen, um eine noch größere Sicherheit zu erreichen. Diese sind RIPEMD-160, SHA-512 und Whirlpool.

3.2 Pre-Boot-Authentication

Um die System-Partition mit dem Betriebssystem darauf verschlüsseln zu können, muss man das spezielle Pre-Boot-Authentication-Verfahren verwenden. Dabei wird die komplette System-Partition verschlüsselt (bis auf den Master-Boot-Record und den Bootloader) und ein spezieller Bootloader installiert, welcher dann für das Entschlüsseln der Systempartition zuständig ist.

Man schaltet den Rechner also ein, wird vom TrueCrypt-Bootloader nach dem Passwort gefragt und nach erfolgreicher Authentifizierung wird die System-Partition entschlüsselt und das installierte Betriebssystem startet. Großer Vorteil bei dieser Methode ist, dass wirklich das komplette Betriebssystem, die Konfigurations-Dateien, die Auslagerungs-Partition (bei Windows wäre dies dann die Auslagerungs-

Datei) und auch alle anderen Daten der System-Partition verschlüsselt werden, sodass auch keine unverschlüsselten Restdaten mehr von einem Angreifer ausgelesen werden können.

Diese System-Verschlüsselung ist momentan noch nicht ganz fehlerfrei. Es existiert ein Tool mit dem Namen *stoned* [4], welches sich zwischen unverschlüsselten Bootloader und die verschlüsselte System-Partition hängt und nach dem Starten des Betriebssystems erheblichen Schaden anrichten kann, da es sämtliche Sicherheitsvorkehrungen des Betriebssystems umgehen kann. Dabei wird es von gängigen Virenscannern nicht entdeckt. [5] Dieses Tool ist frei verfügbar und sollte eigentlich die TrueCrypt-Entwickler auf die Schwachstelle aufmerksam machen. Bis jetzt ist dieser Fehler allerdings immer noch enthalten.

3.3 Plausible Deniability

Sollte man, aus welchen Gründen auch immer, einmal gezwungen werden, seine verschlüsselte TrueCrypt-Partition zu entschlüsseln, obwohl man das selbst gar nicht will, hat man zwei Möglichkeiten: Man beugt sich und gibt seine privaten Daten frei oder man weigert sich und riskiert (wenn die Justiz im Spiel ist) eine hohe Strafe. TrueCrypt bietet aber noch eine dritte Möglichkeit: Man gibt Daten frei, aber nicht seine sensiblen Daten, sondern Pseudo-Daten, die man vorher angelegt hat.

Dies funktioniert folgendermaßen: Beim Verschlüsseln der Partition gibt man zwei Passwörter an, eines zum Entschlüsseln der echten Partition mit den sensiblen Daten, ein weiteres Passwort zum Entschlüsseln der Pseudo-Daten. Das heißt aus einer verschlüsselten Partition können zwei verschiedene entschlüsselte virtuelle Partitionen eingebunden werden. In TrueCrypt gibt es zwei verschiedene Möglichkeiten:

- Hidden Volume: Versteckt eine Partition mit sensiblen Daten innerhalb einer Partition mit unsensiblen Daten (beide sind verschlüsselt) (Abbildung 2)
- Hidden Operating System (nur unter Windows): Das Betriebssystem, das man verstecken möchte, befindet sich versteckt auf einer zweiten Partition, während das Betriebssystem mit den unsensiblen Daten sich auf der ersten Partition befindet. Beim Booten entscheidet der TrueCrypt-Bootloader anhand der Wahl des Passworts, welches Betriebssystem geladen werden soll (Abbildung 3)

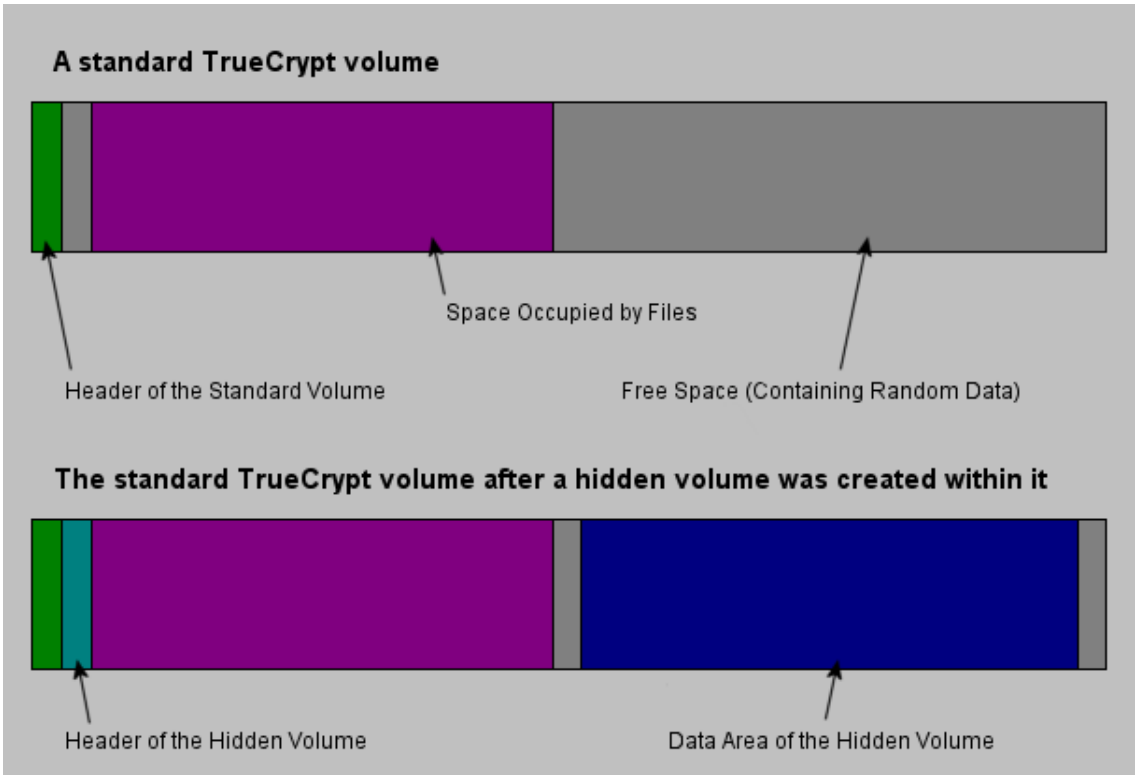

Abbildung 2: Hidden Volume [2]

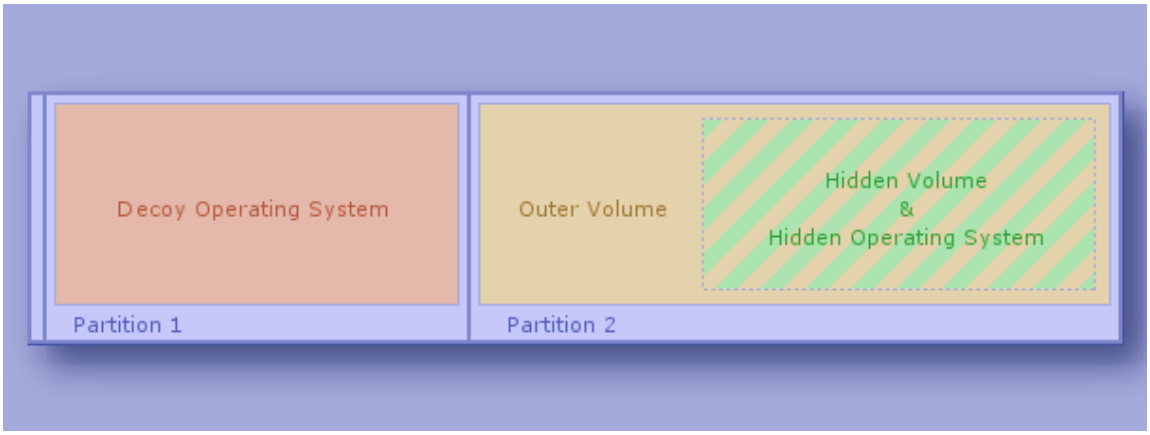

Abbildung 3: Hidden Operating System [2]

3.4 Header

3.4.1 Header-Backup

Ein weiteres wichtiges Merkmal von TrueCrypt ist, dass zum Einen vom Benutzer, zum Anderen von TrueCrypt selbst nicht erkennbar ist, dass es sich um eine TrueCrypt-Partition oder einen TrueCrypt-Container handelt. Deswegen kann man bei der Wahl der Datei, die man einbinden möchte, auch jede auswählen, aber natürlich ist das Einbinden nur erfolgreich, wenn es sich um ein TrueCrypt-Volume handelt und das Passwort richtig eingegeben wurde.

Die sensiblen Daten werden mit einem bei der Erstellung angelegten Master-Key verschlüsselt, der im Header liegt. Das beudetet, das Passwort entschlüsselt nur den Header, der den Master-Key enthält und damit werden wiederum die eigentlichen Daten entschlüsselt. Dadurch ist es auch ohne Weiteres möglich, das Passwort zu ändern, denn es muss nur der Header neu verschlüsselt werden, nicht die eigentlichen Daten, da sich der Master-Key nicht ändert.

3.4.2 Schema

In Abbildung 4 sieht man den Aufbau des Headers. Die Bytes 0-65535 sind dabei für das normale Volume, Bytes 65536-131071 für das Hidden Volume (wenn kein Hidden Volume existiert, stehen hier einfach nur Zufallszahlen). Ab Byte 131072 stehen dann die eigentlichen verschlüsselten Daten. Die letzten 65536 Bytes des Volumes beinhaltet ein Backup des Headers des Hidden Volumes, die 65536 Bytes davor ein Backup des normalen Volumes. Auch hier gilt: sollte es kein Hidden Volume geben, stehen hier nur Zufallszahlen.

Im Folgenden soll nun darauf eingegangen werden, wie genau der Vorgang der Entschlüsselung abläuft.

- Zuerst werden die Bytes 0-512 in den Arbeitsspeicher gelesen (das sind die Daten des Headers des normalen Volumes).
- Anschließend werden die Bytes 65536-66047 in den Arbeitsspeicher gelesen (das sind die Daten des Headers des Hidden Volumes).
- Da ja nirgends gespeichert wird, wie die Daten und der Header verschlüsselt wurden (deswegen ist ein TrueCrypt-Volume auch nicht als solches erkennbar), muss TrueCrypt die eingelesenen Bytes 0-512 nun nach dem TTrial and ErrorPrinzip entschlüsseln. Dies funktioniert so, dass eine Zusammensetzung aus gespeichertem Salt, eingegebenem Passwort, einer der drei Mischalgorithmen, den drei Verschlüsselungs-Algorithmen und deren Konkatenationen, den verschiedenen Operationsmodi und der/die Schlüsselgröße/n die Bytes 64-67 entschlüsseln kann. Kommt dabei der ASCII String TRUE

heraus und stimmen die Checksummen der entschlüsslten Bytes mit denen der Bytes 72-75 und 252-255 überein, war diese Kombination korrekt, das Volume kann eingebunden und die eigentlichen Daten entschlüsselt werden.

- Sollten alle Kombinationen fehlschlagen, wird das gleiche Trial and Error-Verfahren auf die Bytes 65536-66047 angewandt und versucht, das Hidden Volume zu entschlüsseln.

- Sollten auch hier alle Kombinationen fehlschlagen, wird das Mounting abgebrochen. Schuld am Fehlschlagen ist entweder ein falsch eingegebenes Passwort oder ein defekter Header.

Ein großes Problem, das diese Header-Konstruktion mit sich bringt, ist, dass wenn der Header einmal einen Fehler aufweist oder zerstört wurde, die komplette Partition unbrauchbar ist und sich auch nicht wieder rekonstruieren lassen würde. Um dem entgegenzuwirken erstellt TrueCrypt in jeder TrueCrypt-Partition ein Backup des Headers am Ende der Partition. Dieses Backup ist allerdings keine 1:1-Kopie, sondern hat einen anderen Salt und dadurch auch andere Checksummen. Wenn der Header einmal defekt sein sollte, kann man das Header-Backup einspielen und somit wieder auf seine Daten zugreifen. Im Anschluss daran werden wieder neue Header-Backups erstellt.

Neben diesem eingebetteten Header-Backups gibt es auch die Möglichkeit, den Header extern zu sichern. Dies gewährt zusätzlichen Schutz vor defekten Headern. Dabei sei allerdings zu beachten, dass, wenn man ein externes Backup erstellt, danach das Passwort bei der TrueCrypt-Partition ändert und dann wiederum irgendwann später das externe Header-Backup einspielt, das Passwort wieder das alte zum Zeitpunkt der Header-Sicherung ist.

Offset (bytes)	Size (bytes)	Encryption Status†	Description
0	64	Unencrypted§	Salt
64	4	Encrypted	ASCII string "TRUE"
68	2	Encrypted	Volume header format version
70	2	Encrypted	Minimum program version required to open the volume
72	4	Encrypted	CRC-32 checksum of the (decrypted) bytes 256-511
76	16	Encrypted	Reserved (set to zero)
92	8	Encrypted	Size of hidden volume (for normal/outer volumes, set to zero)
100	8	Encrypted	Size of volume
108	8	Encrypted	Byte offset of the start of the master key scope
116	8	Encrypted	Size of the encrypted area within the master key scope
124	4	Encrypted	Flag bits (bit 0 set: system encryption; bit 1 set: non-system in-place-encrypted volume; bits 2–31 are reserved)
128	124	Encrypted	Reserved (set to zero)
252	4	Encrypted	CRC-32 checksum of the (decrypted) bytes 64-251
256	*Var.*	Encrypted	Concatenated primary and secondary master keys**
512	65024	Encrypted	Reserved (for system encryption, this item is omitted‡‡)
65536	65536	Encrypted / Unencrypted§	Area for hidden volume header (if there is no hidden volume within the volume, this area contains random data††). For system encryption, this item is omitted.‡‡ See bytes 0–65535.
131072	*Var.*	Encrypted	Data area (master key scope). For system encryption, offset may be different (depending on offset of system partition).
S-131072‡	65536	Encrypted / Unencrypted§	Backup header (encrypted with a different header key derived using a different salt). For system encryption, this item is omitted.‡‡ See bytes 0–65535.
S-65536‡	65536	Encrypted / Unencrypted§	Backup header for hidden volume (encrypted with a different header key derived using a different salt). If there is no hidden volume within the volume, this area contains random data.†† For system encryption, this item is omitted.‡‡ See bytes 0–65535.

Abbildung 4: Header einer TrueCrypt-Partition [2]

4 Stackbasierte Festplattenverschlüsselung

Im Gegensatz zu blockbasierter Verschlüsselung wird hier auf einem bereits existierenden Dateisystem gearbeitet. Dafür wird ein Verschlüsselungdateisystem als eine Schicht über dem logischen Dateisystem (z.B. FAT oder ext2) geladen. Für die Entwicklung solcher Dateisysteme steht das Framework[1] FiST[6] zur Verfügung, welches im Grundlagenteil dieses Kapitels erklärt wird. Ein Ergebnis dieser Entwicklung ist eCryptFS[7][8], anhand dessen die Funktionsweise stackbasierter Verschlüsselung erläutert werden soll.

4.1 Grundlagen

4.1.1 VFS - Virtual File System

Das *virtual file system* ist eine Abstraktionsschicht über dem konkreten Dateisystem unter Linux[9][11]. Das Ziel des VFS ist es, eine transparente Nutzung der logischen Dateisysteme für den Benutzer zu ermöglichen. Wenn ein neuer Ordner angelegt werden soll, so wird aus Benutzersicht die gleiche Operation ausgeführt, ganz gleich, ob es sich zum Beispiel um ein FAT- oder ext2-Dateisystem handelt. Die VFS-Schicht definiert eine gemeinsame Schnittstelle für jegliche Dateioperationen und übernimmt die konkrete Anweisung an das darunterliegende Dateisystem. Sie verarbeitet also alle Systemaufrufe in Bezug auf ein Linux Dateisystem und leitet diese an konkrete Dateisysteme weiter. Ebenso ist die Interoperabilität zwischen verschiedenen Dateisystemen gegeben. So kann das VFS beispielsweise die transparente Arbeit zwischen einem lokalen und einem Netzwerkspeicher gewährleisten, wenn dieser ebenfalls über das VFS eingebunden ist.

4.1.2 FiST - File System Translator

FiST ist eine Entwicklungsumgebung für Dateisysteme, mit der diese auf hohem Abstraktionsgrad mit wenig Leistungseinbußen entwickelt werden können. Es können nur geschichtete Dateisysteme[2] implementiert werden. Eines der Ziele von FiST ist es, ein plattformunabhängiges Framework zur Verfügung zu stellen.

[1] Ein Framework ist ein Programmiergerüst, welches einen Rahmen für die Entwicklung zur Verfügung stellt. Beispiele sind QT bei C++ und Zend bei PHP. Es ist kein fertiges Programm.

[2] Geschichtete Dateisysteme operieren auf anderen Dateisystemen und nicht auf Containern (z.B. Partitionen). Die Informationen werden 1:1 weitergegeben

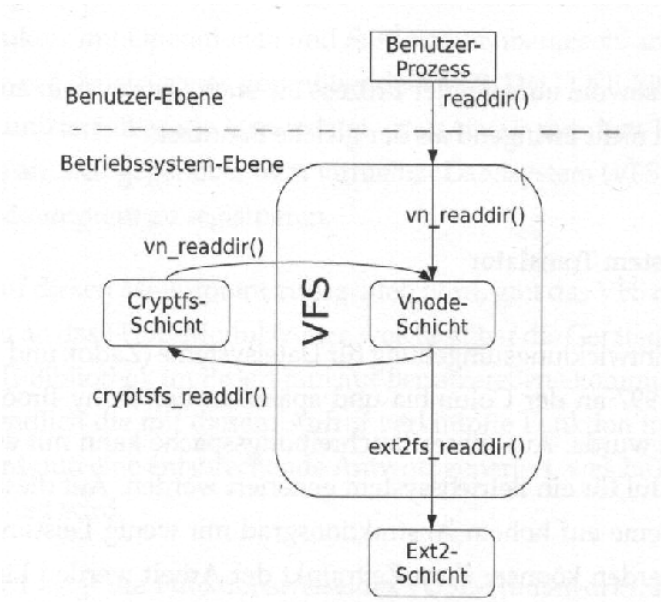

Abbildung 5: Vnode-stacking am Beispiel CryptFS [12]

4.1.2.1 Vnode-stacking

Da FiST ursprünglich für Linux entwickelt wurde, benutzt es die Möglichkeit mehrere Ebenen im VFS übereinander zu stapeln. Dies nennt sich *Vnode-stacking*. Dabei repräsentiert der Vnode den Inode bzw. Dateideskriptor im Hauptspeicher. Das Prinzip soll in Abbildung 5 am Beispiel von CryptFS verdeutlicht werden.

Beispiel Führt ein Benutzer *readdir()* aus, nimmt das VFS diesen Befehl entgegen. Die Vnode-Schicht ruft daraufhin im VFS die mit dieser Vnode verknüpfte Funktion auf. Das wäre in diesem Fall die readdir-Funktion der CryptFS-Treiberschicht. Diese analysiert im nächsten Schritt, welche Daten sie benötigt und fordert sie von der Vnode-Schicht an. Diesmal jedoch erfolgt der Aufruf mit der Vnode des darunterliegenden Dateisystems. Abschließend ruft die Vnode-Schicht nun die entsprechende Funktion des darunterliegenden Dateisystems auf. So wird sich durch die einzelnen Schichten für den Benutzer transparent durchgearbeitet, bis die gewünschte Operation schließlich auf dem logischen Dateisystem ausgeführt wird.

4.1.2.2 Aufbau

Im Allgemeinen besteht der Aufbau einer FiST-Datei aus vier Teilen. C-Deklarationen, FiST-Deklarationen, FiST-Regeln und zusätzlichem C-Code. Damit können für Dateisysteme typische Operationen implementiert werden. Darunter fallen bei-

spielsweise das Öffnen und Erzeugen einer Datei, sowie das Durchsuchen von Ordnern.

4.2 eCryptFS

eCryptFS ist ein unternehmenstaugliches natives Verschlüsselungsdateisystem für Linux. Warum es gerade für Unternehmen geeignet ist, wird im Folgenden aufgezeigt werden. eCryptFS ist eine Weiterentwicklung[3] von CryptFS, welches bereits 1998 von Erez Zadok aus FiST entwickelt wurde. Seit der Version 2.6.19 ist das POSIX-konforme eCryptFS im Betriebssystemkern als *Stand-alone Kernel Modul* verankert. Das bedeutet, dass keinerlei Modifikationen am Linux-Kernel notwenig sind und auch keine Patches oder ähnliches installiert werden müssen. Aktivierung erfolgt durch das Laden des eCryptFS-Moduls, welches daraufhin im *Kernelspace*, also im Adressraum des Betriebssystems, arbeitet.

4.2.1 Funktionsweise

Mounten. Beim Einbinden einer eCryptFS-Partition kann man sich durch verschiedenste Methoden Authentifizieren. Von der einfachen Passphrase über Private / Public-Key-Paar bis hin zu TPM[4]. Nach einer erfolgreichen Authentifzierung erstellt der *Mount-Helper* - ein daraufhin ausgeführtes Helfer-Skript - den *authentication token* für die Passphrase des Benutzers. Dabei greift eCryptFS auf den *Linux kernel keyring service*, einen im Linux-Kernel implementierten Schlüsselring, der für jede Benutzer-Session die Schlüssel speichert und verwaltet. Der *authentication token* wird in eben diesem Schlüsselring des Benutzers gespeichert. eCryptFS nutzt den Token-Inhalt zur Erstellung neuer und zum Öffnen existierender, verschlüsselter Daten.

Ablage der Daten im Dateisystem. Die Verschlüsselungsinformationen werden im Header[5] der Zieldatei , welcher unverschlüsselt bleibt, gespeichert. Auch die Metadaten bleiben in Klartext vorhanden[13], was zum Nachteil hat, dass die Größe, die Zugriffsrechte und der Dateiname lesbar bleiben, was wiederum Anhaltspunkte auf den Inhalt der Datei gibt. Allerdings hat diese Methode einen bedeutenden Vorteil. Die Dateien bleiben überaus portabel und können deshalb in andere Ordner, Partitionen oder gar auf andere Computer verschoben und ver- und entschlüsselt[6] werden, ganz im Gegensatz zu Blockbasierter Verschlüsselung,

[3] 2003 - 2006 in IBMs Linux Technology Center

[4] *Trusted Platform Module*: Bindung von Daten an Geräte

[5] Der Header entspricht den ersten Bytes einer Datei. Dies wird in Abbildung 6 und 7 verdeutlicht.

[6] Vorraussetzung ist natürlich, dass das eCryptFS-Modul auf dem Computer vorhanden und aktiviert ist

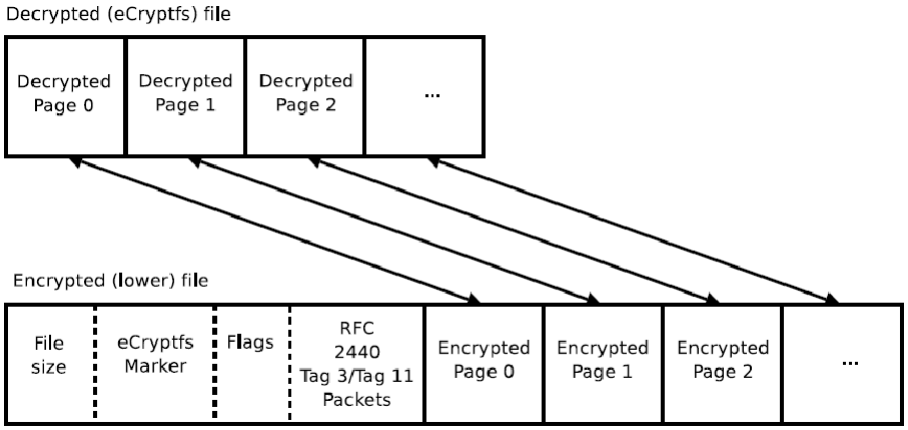

Abbildung 6: Ver- und Entschlüsselte Datei [8]

bei der man immer eine komplette Blocksequenz[7] verschieben müsste. Ein Anwendungsgebiet, in dem die Portabilität eine große Rolle spielt, ist das Anlegen von Datensicherungen. Möchte man beispielsweise eine wenige Kilobyte große Datei verschieben und hat diese mit blockbasierter Verschlüsselung in einem 500 Megabyte großen Container, dann wäre man gezwungen die kompletten 500 Megabyte zu verschieben, um diese Datei zu bewegen, und dabei ihre Verschlüsselung zu erhalten. Dies zeigt den Vorteil der unverschlüsselten Header.
Die Nutzdaten werden in sogenannte *Extents*, welche standardmäßig die gleiche Größe haben wie die *Pages*[8], aufgespaltet und jedes *Extent* wird einzeln ver- und entschlüsselt[9]. eCryptFS verwendet dabei als Standard den AES-128 Algortihmus. Dies lässt sich aber durch Änderung der Einstellungen auch mit Blowfish, Twofish oder DES3 realisieren.
Abbildung 6 zeigt sowohl die entschlüsselte (*decrypted file*) als auch die verschlüsselte Datei (*Encrypted (lower) file*), welche den bereits auf Seite 13 beschriebenen Header vor den Nutzdaten (hier beginnend bei *Encrypted Page 0*) enthält. Aus der Abbildung sehr gut erkenntlich wird die Abarbeitung einzelner *Extents* und nicht der kompletten Nutzdaten mit einmal. Der Vorteil dabei ist, dass, wenn beispielsweise nur das letzte Bit geändert werden soll, man nicht den kompletten Block entschlüsseln, dann ändern und im Anschluss wieder verschlüsseln muss, sondern man die Arbeit auf das *Extent* beschränken kann, welches geändert werden soll.

[7] Ein Block ist in diesem Sinne ein Container wie in Kapitel 3 (TrueCrypt) beschrieben. Hier verwendet, um die Unterschiede zu blockbasierter Verschlüsselung aufzuzeigen

[8] Eine Page (dt. Seite) ist ein Speicherblock fester Größe, der zur Abbildung von virtuellem auf physikalischen Speicher verwendet wird. Die Größe ist in gängigen Betriebssytemen 4 Kilobyte

[9] Bilden der *Extents* mit Blockalgorithmus im CBC-Modus

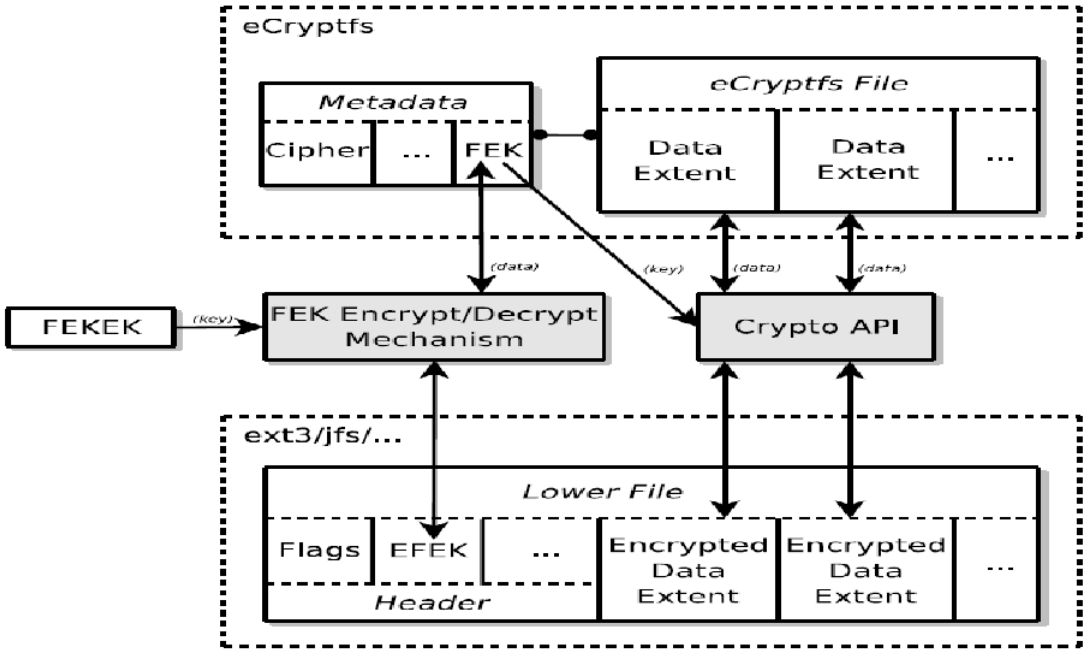

Abbildung 7: Prinzip der Ver- und Entschlüsselung [14]

Das spart Aufwand und somit Zeit.

Lesen und Schreiben von Dateien. Beide Operationen funktionieren vom Prinzip her gleich. Im Allgemeinen öffnet eCryptFS das *lower file*[10] und liest den Header der Datei. Dabei wird untersucht, ob sich der zum Entschlüsseln benötigte *authentication token* im Benutzer-Schlüsselring befindet. Ist dies der Fall, kann die Datei ent- und später auch wieder verschlüsselt werden.
Abbildung 7 verdeutlicht den Vorgang. Im Header der *lower file* befindet sich der sogenannte *encrypted file encryption key (EFEK)*, der codierte Schlüssel, der für das Ver- und Entschlüsseln nötig ist. Dieser wird durch den *FEK Encrypt/Decrypt Mechanism* in Klartext übersetzt. Dafür benötigt es aber den *File encryption key encryption key (FEKEK)* aus dem *authentication token* des Schlüsselrings des Benutzers. Nun können die einzelnen *Extends* via Crypto API[11] mittels des *file encryption key (FEK)* ver- und entschlüsselt werden.
Die Passphrase des Benutzers ist also nicht der Schlüssel selbst, sondern notwendig, um den benötigten Schlüssel zu dechiffrieren.

[10] Durch Vnode-Stacking, welches in Kapitel 4.1.2.1 erklärt wird
[11] Die Crypto API ist ein kryptographisches Framework im Linux-Kernel. Es enthält alle essentiell wichtigen Verschlüsseulngsalgorithmen und Hash-Funktionen.

4.2.2 User Access Control

Ein bedeutender Vorteil und dadurch eine weitere Rechtfertigung für die Unternehmenstauglichkeit von eCryptFS ist die feingranulare Mehrbenutzerfähigkeit.[10]
In großen Firmen ist es nicht unüblich, an mehreren Projekten gleichzeitig beteiligt zu sein. Da die Daten eines solchen Projekts den Anspruch erfüllen sollen, sicher vor anderen Unbeteiligten zu sein, müssen diese verschlüsselt werden. Hier stößt man auf ein Problem. Da man höchstwahrscheinlich nicht alleine arbeitet, müssten pro Projektgruppe ein gemeinsames Geheimnis ausgemacht werden, welches sich jeder Teilnehmer merken muss. Wenn man nun an mehreren Projekten arbeitet, muss man sich im Umkehrschluss also auch mehrere solcher Geheimnisse merken. Hier setzt eCryptFS an und ermöglicht eine für den Benutzer vollkommen transparente Benutzerverwaltung für diese Dateien. Jeder kann die Dateien lesen und schreiben, für die er die Berechtigung hat. Wie wird das realisiert? Jede Datei besitzt eine Inode, welche wiederum ein Array besitzt, das die Signaturen der *authentication token* von denjenigen Benutzern enthält, welche berechtigt sind, diese Datei zu entschlüsseln.
Dieses Methode bringt einen weiteren Vorteil mit sich: Selbst wenn das Betriebssystem manipuliert („*gehackt*“) wird und dadurch das Dateiberechtigungssystem des Betriebssystems versagt, kann der Angreifer immer noch nicht die verschlüsselten Daten entschlüsseln, da, wie gerade aufgezeigt, eCryptFS vollkommen unabhängig vom Betriebssystem die Dateiberechtigungen verwaltet.
Möchte ein Benutzer nun eine mit eCryptFS verschlüsselte Datei in Klartext übersetzen, dann wird nachgesehen, ob die Signatur seines *authentication token* im Array der Inode vorhanden ist.

4.3 Spezifischer Nachteil

Durch die Verwendung von CBC, bringt eCryptFS eben dessen Schwachstellen mit. Zum Einen gab es bereits viele bekannte Angriffe auf CBC und zum Anderen ist keine Parallelisierung des CBC-Moduls möglich, was zu Leistungseinbußen führt[15].

5 Fazit

Nachdem nun die zwei verschiedenen Prinzipien der Festplattenverschlüsselung aufgezeigt wurden, soll nun auf allgemeine Probleme, sowie die Leistung eingegangen werden. Schließlich werden die beiden Möglichkeiten zusammenfassend gegenübergestellt.

5.1 Fallstricke

In der heutigen Zeit erwartet der Benutzer höchsten Komfort bei der Verwendung von Festplattenverschlüsselung. So sollen Dateien im Hintergrund automatisch ver- und entschlüsselt werden, ohne dass ein Passwort wiederholt eingegeben werden muss. Um diesen Anforderungen nachzukommen, wird der dechiffrierte Festplattenschlüssel im Speicher gehalten und die bereits entschlüsselten Daten liegen bis zum Aushängen des Dateisystems im Klartext vor. Selbst wenn der Computer in den Ruhezustand geht, wird der Festplattenschlüssel im sogenannten *Hibernation-File* gespeichert. Durch diese Funktionsweise treten natürlich Sicherheitsrisiken auf. Im Folgenden wird davon ausgegangen, dass der Angreifer, der auf die sensiblen Daten zugreifen möchte, physischen Zugriff auf das Gerät hat.

Beispiel für einen Angriff - Cold boot attack Da die meisten Benutzer, wenn sie ihre Arbeit unterbrechen, ihren Laptop nicht ausschalten, sondern nur in einen gesperrten Zustand überführen, bleibt der Festplattenschlüssel auch weiterhin im Speicher. Es wird fälschlicherweise oft davon ausgegangen, dass Arbeitsspeicher ihre Daten sofort verlieren, wenn der Computer ausgeschaltet wird. Dies ist aber nicht der Fall. Durch den Aufbau mit Kondensatoren dauert es eine gewisse Zeit (bis zu zwei Minuten), bis sich die Daten verflüchtigen. Angreifer nutzen diese physikalische Eigenschaft der Speicher aus. Der Computer wird neu gestartet und ein spezielles Programm gebootet. Mit diesem ist es möglich, den Speicher auszulesen und den Inhalt beispielsweise auf einen USB-Stick zu übertragen. Schafft der Angreifer dies, bevor die Daten vom Speicher gelöscht sind, dann kann er den USB-Stick nach Festplattenschlüsseln durchsuchen. Das kann beispielsweise durch ein Linux-Live-System geschehen.
Es gibt auch eine Möglichkeit die Zeit bis zur Verflüchtigung der Daten zu verlängern. Durch Zufuhr von Kälte – z.B. durch das Besprühen des Arbeitsspeicher mit Stickstoff – ist der Inhalt auch 10 Minuten nach Abschalten des Computers noch lesbar.

Verlust des Passworts impliziert den Verlust der Daten. Festplattenverschlüsselung ist wie die meisten Sicherungssysteme darauf ausgelegt, dass nur derjenige Benutzer, der die notwendige Berechtigung hat, für bestimmte Sachen

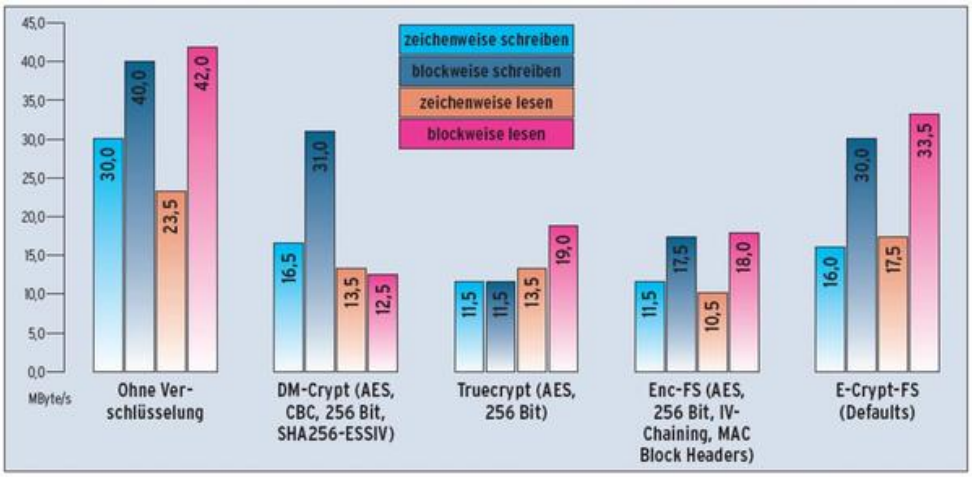

Abbildung 8: Geschwindigkeitsbenchmark diverser Verschlüsselungssysteme [16]

autorisiert ist. In diesem Fall wäre das beispielsweise das Lesen einer verschlüsselten Datei. Verliert nun der Benutzer sein Passwort, verliert er auch seine Daten. Das ist vergleichbar mit dem Verlust eines Haustürschlüssels, nur dass die Daten auf der Festplatte irreversibel verloren sind, da niemand ohne Berechtigung an die Dateien kommt.

5.2 Performance

So wie jede Datenbewegung Rechenzeit verbraucht, ist es nur logisch, dass Verschlüsselungssysteme noch mehr Zeit kosten. Daten müssen nicht nur bewegt sondern auch berechnet werden.

In Abbildung 8 wird das Ergebnis eines Geschwindigkeitstest verschiedener Verschlüsselungssysteme gezeigt. Hierbei fällt auf, dass sich durch den Einsatz solcher Systeme die Leistung verringert.
Das schnellste getestete System ist hierbei eCryptFS. Zwar ist DM-Crypt beim Schreiben um 0,5 bis 1,0 Megabytes pro Sekunde schneller, aber eCryptFS hat die mit Abstand besten Werte beim zeichen- und blockweisen Lesen. Verwendet man das in diesem Test schnellste System, so hat man beim Lesen Einbußen von ungefähr 20 bis 25%. Dabei ist der Unterschied zwischen dem Leistungsverlust von zeichen- und blockweisem Lesen sehr gering. Anders ist es beim Schreiben: Hier erfährt man beim zeichenweisen Schreiben eine Minderung des Durchsatzes von ca. 45%, während die Leistung beim blockweisen Schreiben um „nur“ 25% reduziert wird.
Allerdings ist die aktuelle Hardware mittlerweile so leistungsstark, dass die Verwendung von Verschlüsselung die tägliche Arbeit kaum beeinträchtigt. Ausnahmen

bilden dabei rechenintensive Anwendungen und hohe Datentransferraten, wie man sie zum Beispiel im Videoschnitt hat. Ist der Computer also schon ausgelastet und müssen sehr viele Daten bewegt werden, dann fallen die zusätzlichen Kosten für Ver- und Entschlüsselung deutlich ins Gewicht.

5.3 Zusammenfassender Vergleich

Zum Abschluss werden TrueCrypt als Vertreter für blockbasierte und eCryptFS für stackbasierte Verschlüsselung gegenübergestellt.[10][17]

	TrueCrypt	eCryptFS
Konzept und Implementierung	Einfach	Sehr komplex
Speicherung verschlüsselter Daten	Allokieren eines Blocks aus dem vorhandenen Dateisystem; Komplette Partition verschlüsselt	Auf das gemountete Dateisystem; Ggf. Unverschlüsselte Daten auf gleicher Partition
Daten	Absolut nicht reproduzierbar; Vor Mounten keine Sichtbarkeit	MetaDaten unverschlüsselt; Sichtbar im Dateisystem
Portabilität	Ganzer Container, Partitionen nicht portabel	Einzelne Dateien, Verzeichnisse → sehr gut
Swap - Space	Verschlüsselbar	Nicht verschlüsselbar
Betriebssysteme	Windows, MacOS, Linux	Linux
Mehrbenutzerbetrieb	Dateiberechtigung des OS	User Access Control

Tabelle 1: Vergleich der beiden Festplattenverschlüsselungsverfahren

In der Tabelle werden die Unterschiede zwischen den zwei Verschlüsselungssystemen deutlich gemacht. Während TrueCrypt aus einem vorhandenen Dateisystem Blöcke[12] herausnimmt und diese dann nur noch für verschlüsselte Daten zur Verfügung stehen, setzt eCryptFS auf das bereits gemountete Dateisystem auf und die Daten können darauf überall gespeichert werden, sie sind also unabhängig von einem bestimmten Block. Dabei kann es vorkommen, dass, im Gegensatz zur blockbasierten Verschlüsselung, sich verschlüsselte, deren Metadaten unverschlüsselt sind, und im Klartext vorliegende Daten auf der gleichen Partition befinden. Die codierten Daten liegen also sichtbar im Dateisystem. Mit TrueCrypt verschlüsselte Daten lassen sich wiederrum absolut nicht reproduzieren. Man erkennt nicht, dass es sich hierbei um einen verschlüsselten Block handelt, was zu

[12] Es handelt sich hier um Containerdateien bzw. ganze Partitionen, die TrueCrypt wie in Kapitel 3 beschrieben zur Verschlüsselung benötigt

zusätzlicher Sicherheit führt. Dies hat allerdings den Nachteil, dass man ganze Container zur Sicherung verschieben muss. Stattdessen braucht man bei eCryptFS nur die einzelnen Dateien bzw. Verzeichnisse bewegen, die gesichert werden sollen. Ein großer Nachteil von eCryptFS ist, dass der Auslagerungsspeicher nicht verschlüsselbar ist. TrueCrypt punktet hier mit der Fähigkeit komplette Partitionen verschlüsseln zu können. Außerdem ist es für nahezu alle Betriebssysteme vorhanden, eCryptFS hingegen ist ein reines Linux-Verschlüsselungssystem, welches sich dafür aber durch seinen speziellen Mehrbenutzerbetrieb auszeichnet.

Literatur

[1] http://www.heise.de/ct/artikel/Datentresor-289846.html

[2] TrueCrypt and Documentation: www.truecrypt.org

[3] eCryptFS: http://ecryptfs.sourceforge.net/

[4] Heise.de: Bootkit hebelt Festplattenverschlüsselung aus (http://www.heise.de/security/meldung/Bootkit-hebelt-Festplattenverschluesselung-aus-748859.html)

[5] http://www.sicherheitsblog.info/blog/index.php?/archives/222-2009-07-31.html

[6] Erez Zadok, Jason Nieh: FiST: A Language for Stackable File Systems

[7] Michael Austin Halcrow: eCryptfs: An Enterprise-class Cryptographic Filesystem for Linux

[8] Michael Austin Halcrow: eCryptfs v0.1 Design Document

[9] Zusammenfassung virtual file system mit weiterführenden Links: http://en.wikipedia.org/wiki/Virtual_file_system

[10] http://ecryptfs.sourceforge.net/ecryptfs-faq.htm

[11] Seminarvortrag der Universität Heidelberg von Nguyen und Müller: http://pvs.informatik.uni-heidelberg.de/Teaching/DASY-07/nguyen.pdf

[12] Max Lindner – Verschlüsselte Dateisysteme in Mehrbenutzer-Szenarien

[13] http://www.linuxjournal.com/article/9400?page=0,2

[14] Seminarvortrag der Universität Heidelberg von Gärtner und Seeliger: http://pvs.informatik.uni-heidelberg.de/Teaching/DASY-07/seeliger.pdf

[15] Chris J. Mitchell - Error Oracle Attacks on CBC Mode: Is There a Future for CBC Mode Encryption?

[16] http://www.linux-magazin.de/Heft-Abo/Ausgaben/2007/03/Geheime-Geschaefte/(offset)/2

[17] http://en.wikipedia.org/wiki/Comparison_of_disk_encryption_software