EINSTUFUNG AUFGEHOBEN

VS – NUR FÜR DEN DIENSTGEBRAUCH

[redacted]

[redacted]

Untersuchung TrueCrypt

Arbeitspaket 4

Software-Analyse

[redacted]

[redacted]

Historie

Version	Autor	Kommentar	Datum
0.1	[redacted]	Initiale Dokumentenversion erstellt	26.08.10
0.2		TOE Design, Beschreibung, Codeanalyse	03.09.10
0.3		Ergänzungen Architektur, Erweiterbarkeit	08.09.10
0.4		Use Case Realization	14.09.10
0.5		Use Cases I/O, SAD Struktur	16.09.10
0.6		SAD, Key Management, Korrekturen	22.09.10
0.7		Sprachversionen, Erweiterung Zufall, Änderung des PBA Hashes, Verbot von Dateivolumes, Use Cases Fortsetzung, SAD Caching, Öffnen, Einhängen, Passwort ändern, Header sichern/wiederherstellen, Sicheres Löschen, Functional Specification Bootloader und CoreService	01.10.10
0.8	[redacted]	Functional Specification FUSE und Kernel Driver, Bewertung und Empfehlung für Schwächen, Integrität, Security Token, Key Management, Full-Disc-Encryption Linux	08.10.10
0.9	[redacted]	FUSE Driver Race Condition, SAD Secure Startup, Linux FDE Anmerkungen, Bewertung AES-Implementierung	11.10.10
0.9.1	[redacted]	Erste Überarbeitung entsprechend Kommentaren	27.10.10
0.9.2	[redacted]	Zweite Überarbeitung: Alle noch ausstehenden Kommentare ergänzt. Kleinere sonstige Korrekturen.	03.11.10
0.9.3	[redacted]	Review von 0.9.2 und kleinere Korrekturen	03.11.10
1.0	[redacted]	Finale Version 1.0	15.11.10

Inhaltsverzeichnis

1 Architektur von TrueCrypt

1.1 TOE Design / High-Level Design

TrueCrypt in der hier vorliegenden und untersuchten Version 7 besteht aus mehreren Komponenten und Submodulen. Es werden Sourcecodepakete für Windows und Linux angeboten, die sich zwar in ihrem Inhalt unterscheiden, jedoch trotzdem Code vom jeweils anderen Betriebssystem enthalten. Eine Unterscheidung nach den herausgegebenen Paketen ist daher nicht sinnvoll, stattdessen bietet sich eine Unterscheidung nach der eigentlichen Verwendung an (ob gemeinsam oder nur in einer der beiden Umgebungen):

Gemeinsam genutzter Code:
• Common • Crypto • *Platform** • Release • Resources
Windows-Only:
• Boot/Windows • Driver • Mount • Format • Setup
Unix-generic:
• *Core** • Main • *Platform*/Unix • Resources/Icons • Volume • *Driver*/Fuse

Linux:
• *Core*/Linux
MacOSX:
• Build/Resources/MacOSX • *Core*/MacOSX
Solaris:
• *Core*/Solaris
FreeBSD:
• *Core*/FreeBSD

Damit sind für Windows/Linux die folgenden Verzeichnisse interessant:

Windows:	**Linux:**
• Boot/Windows • Common • Crypto • Driver • Format • Mount • Platform • Release • Resources • Setup	• Core • Core/Linux • Common • Crypto • Driver/Fuse • Main • Main/Unix • Platform • Platform/Unix • Release • Resources • Resources/Icons • Volume

Im Folgenden werden (aufgetrennt nach Linux/Windows) die *Source Lines of Code* (SLOC) je nach Ordner dargestellt. Dadurch lässt sich vor allem überblicken, in welchen Sprachen das Produkt hauptsächlich geschrieben ist.

1.1.1 Sourcecode-Analyse Linux

```
SLOC    Directory       SLOC-by-Language (Sorted)
23829   Common          ansic=19452,cpp=3056,xml=1321
15260   Main            cpp=15260
9485    Crypto          ansic=7195,asm=2290
4730    Volume          cpp=4730
4006    Core            cpp=4006
3761    Platform        cpp=3761
32      Build           xml=32
0       Resources       (none)

Totals grouped by language (dominant language first):
cpp:          30813 (50.43%)
ansic:        26647 (43.61%)
asm:           2290 (3.75%)
xml:           1353 (2.21%)
```

Deutlich sichtbar ist hier, dass der Sourcecode in allen Ordner bis auf *Common* und *Crypto* ausschließlich in C++ programmiert ist. Die *Crypto*-Implementierung ist hierbei tatsächlich die einzige gemeinsam genutzte Komponente. *Common* wird zwar während des Linuxbuilds auch benutzt, allerdings werden dort lediglich 4 Implementierungen (`Crc`, `Endian`, `GfMul` und `Pkcs5`) und einige Header benutzt. Berücksichtigt man diese Tatsache für das *Common* Verzeichnis, so stellt dieses Verzeichnis nur noch 1400 SLOC in C, sowie 613 SLOC in C++ bereit. Dadurch steigt in der Gesamtbetrachtung der Anteil von C++ am tatsächlich genutzten Code, während der C Anteil sinkt:

```
Totals grouped by language (dominant language first):
cpp:          28370 (69.86%) (30813 - 3056 + 613)
ansic:         8595 (21.17%) (26647 - 19452 + 1400)
asm:           2290 (5.64%)
xml:           1353 (3.33%)
```

Damit liegt der Anteil von C++ bei knapp **70%.**

1.1.2 Einstiegspunkt bei Linux

Unter Linux startet das Programm wie folgt:

```
/Unix/Main.cpp
```

In der `main()`-Routine wird zunächst das verwendete Betriebssystem und die Systemumgebung überprüft. Dann wird geprüft, ob das Programm im CoreService-Modus ausgeführt werden soll. In diesem Falle wird ausschließlich

- `CoreService::ProcessElevatedRequests()`

gestartet. Ist dies nicht der Fall, so wird im Normalmodus zuerst

- `CoreService()`
- `EncryptionThreadPool()`

gestartet. Im Anschluss wird überprüft, ob TrueCrypt als reine textbasierte Anwendung:

```
Application::Initialize (UserInterfaceType::Text)
```

laufen oder die GUI gestartet werden soll:

```
Application::Initialize (UserInterfaceType::Graphic)
```


1.1.3 Sourcecode-Analyse Windows

```
SLOC    Directory       SLOC-by-Language (Sorted)
23829   Common          ansic=19452,cpp=3056,xml=1321
9485    Crypto          ansic=7195,asm=2290
8537    Mount           ansic=7625,cpp=912
8325    Format          ansic=8143,cpp=182
6102    Driver          ansic=6102
3424    Boot            cpp=2752,ansic=507,asm=165
3091    Setup           ansic=3024,cpp=67
2796    Platform        cpp=2796
0       Release         (none)
0       Resources       (none)
0       top_dir         (none)

Totals grouped by language (dominant language first):
ansic:         52048 (79.35%)
cpp:            9765 (14.89%)
asm:            2455 (3.74%)
xml:            1321 (2.01%)
```

Im Gegensatz zu Linux besteht die Windowsimplementierung zu knapp **80%** aus C-Code und nur zu einem sehr geringen Teil aus C++.

1.1.4 Einstiegspunkt bei Windows

Unter Windows gibt es zwei Einstiegspunkte in unterschiedlichen Subsystemen, die durch zwei getrennte Binaries begründet sind.

Volume Creation Wizard

Die `WinMain()` Methode des *Volume Creation Wizard* im *Format*-Subsystem (`Format/Tcformat.c`) instantiiert zunächst ein *BootEncryption*-Objekt zur Verwaltung der Full-Disk-Encryption unter Windows, und führt dann die generische Initialisierungsfunktion `InitApp()` aus. Im Anschluss daran werden:

- `RandInit()`
- `DriverAttach()`
- `AutoTestAlgorithms()`

gestartet. Erst nach erfolgreichem Abschluss dieser Funktionen wird das eigentliche Hauptdialogfenster angezeigt.

Mount-Anwendung

Die `WinMain()` Methode der *Mount-Anwendung* im *Mount*-Subsystem (`Mount/Mount.c`) verhält sich ähnlich zum *Volume Creation Wizard*, führt aber die Funktionen `RandInit()` und `AutoTestAlgorithms()` nicht aus.

1.1.5 Subsysteme

Im Folgenden wird die Ordnerstruktur des Sourcecodes zur Unterteilung in einzelne Subsysteme genutzt. Sofern nicht anders gekennzeichnet, ist das Subsystem und seine Beschreibung für beide Betriebssysteme (Linux und Windows) zutreffend.

Übersicht

Eine Gesamtübersicht der Subsysteme wird in Bild 1 sowie Bild 2 gegeben.

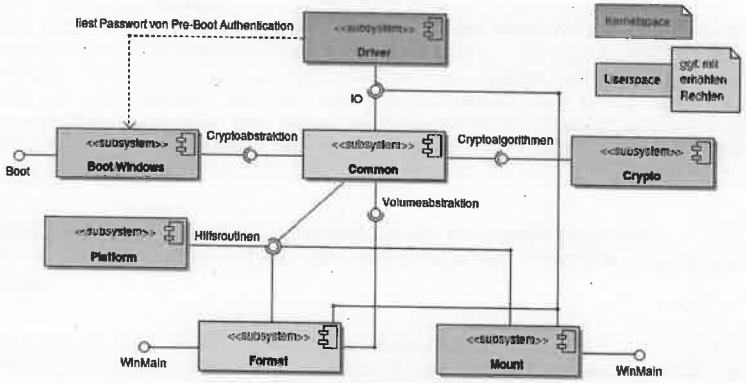

Bild 1: Übersicht der Subsystembeziehungen in Windows

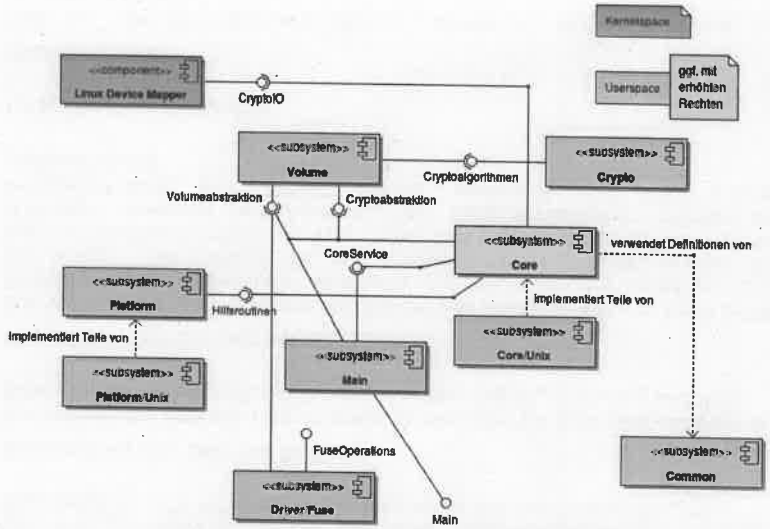

Bild 2: Übersicht der Subsystembeziehungen in Unix/Linux

Boot (nur Windows)

Das Boot-Subsystem enthält die nötige Implementierung für die *Pre-Boot Authentication*-Funktion; welche ausschließlich unter Windows zur Verfügung steht und das Booten des Betriebssystems von einer verschlüsselten Partition erlaubt. Hierzu wird ein eigener Bootloader verwendet, welcher die Auswahl des zu bootenden Systems anbietet und mittels Eingabe eines Passworts das Öffnen einer verschlüsselten Betriebssystempartition ermöglicht. Nach erfolgreichem Öffnen wird der eigentliche Betriebssystembootloader gestartet.

Common (hauptsächlich Windows)

Das Common-Subsystem enthält verschiedenste Grundfunktionen, die hauptsächlich unter Windows genutzt werden. So stellt dieses System die Crypto- und Volume-Abstraktionen, den Zufallszahlengenerator sowie viele Low-Level-Funktionen (z.B. Funktionen zum Formatieren von FAT/NTFS, Zugriff auf die Windows-Registry, Laden von Sprachdateien, mathematische Primitive, Prüfsummen uvm.) zur Verfügung. Dieses Subsystem scheint veraltet und wurde für unixartige Betriebssysteme in mehrere Subsysteme (Volume, Core, Main), aufgeteilt. Nur ein sehr geringer Teil dieses Systems (insgesamt nur 4 Komponenten) wird auch von anderen Betriebssystemen verwendet.

Core (nur Unix und Derivate)

Das *Core*-Subsystem enthält generische Implementierungen bestimmter Kernelemente von TrueCrypt, die parallel auch in den *Common-/Format-/Mount*-Subsystemen für Windows implementiert sind. Dazu zählen u.a. der `RandomNumberGenerator` und der `VolumeCreator`. Das Subsystem ist so aufgebaut, dass es betriebssystemspezifisch erweiterbar ist (siehe *Core/Unix*).

Core/Unix (nur Unix und Derivate)

Das *Core/Unix*-Subsystem erweitert *Core* um Unix-spezifische Implementierungen. In weiteren Unterordnern wird diese Implementierung für Unixderivate (Linux, MacOSX, FreeBSD, usw.) vervollständigt. So wird in *Core/Unix/Linux* beispielsweise die Interaktion mit dem Linux Device Mapper implementiert.

Crypto

Das *Crypto*-Subsystem enthält die Implementierung sämtlicher unterstützter kryptographischer Primitive, d.h. Verschlüsselungs- und Hashalgorithmen.

Driver (nur Windows)

Das *Driver*-Subsystem enthält Kernel-Mode-Treiber für Windows zum Betreiben verschlüsselter Laufwerke. Der Treiber implementiert unter anderem einen Filter für Laufwerke und Volumes, der Ver- und Entschlüsselung beim Zugriff transparent abwickelt. Zu diesem Zwecke kommuniziert der Treiber mit dem *Common*-Subsystem und nutzt dort den `EncryptionThreadPool`. Das notwendige Passwort erhält der Treiber entweder aus dem Userspace oder direkt vom *Boot*-Subsystem (bei *Pre-Boot Authentication*).

Driver/Fuse (nur Unix und Derivate)

Das Fuse-Subsystem enthält eine Treiberimplementierung für den Zugriff auf ein TrueCrypt-Volume mittels des FUSE-Frameworks (Filesystem in USErspace), welches unter mehreren Unixderivaten zur Verfügung steht. Dieses Framework kann genutzt werden, wenn die Nutzung des Linux Device Mapper nicht möglich ist (z.B. unter MacOSX mit MacFUSE).

Format (nur Windows)

Das *Format*-Subsystem stellt Routinen zum Formatieren (Erzeugen der TrueCrypt Struktur) von Partitionen unter Windows bereit, sowie die nötige GUI zum Bedienen dieser Routinen (*Volume Creation Wizard*). Teile dieser Komponente benötigen erhöhte Rechte und können über Windows UAC (User Account Control) entsprechend Administratorrechte anfordern (durch Bestätigung des Benutzers). Diese Komponente kann zusätzlich über ein COM Interface angesprochen werden, u.a. durch Teile des *Common*-Subsystems (z.B. `BootEncryption`). Die eigentlichen Routinen zum Formatieren des inneren Dateisystems liegen ebenfalls im *Common*-Subsystem und werden von *Format* dort aufgerufen.

Die COM-Technik, eine windowseigene Technik für Interprozesskommunikation, kommt hier zum Einsatz, damit das entsprechende Interface zur Verfügung gestellt werden kann, unabhängig davon, in welchem Prozess es gerade läuft. Dies ist insbesondere wichtig für Windows UAC, wenn das Interface mit erhöhten Rechten gestartet (separater Prozess), aber dennoch von der Implementierung mit Nutzerrechten bedient werden soll.

Main

Das *Main*-Subsystem stellt den Entrypoint unter unixartigen Systemen bereit und enthält sowohl das textuelle Interface, als auch die GUI (wxWidgets-basiert). Dieses Subsystem kommuniziert nach dem Start hauptsächlich mit dem *Core*-Subsystem, um die gewünschten Aktionen durchzuführen.

Mount (nur Windows)

Das *Mount*-Subsystem enthält die Implementierung für die Mount-Anwendung von TrueCrypt, welche für den Umgang mit vorhanden Volumes (z.B. Öffnen und Schließen) verantwortlich ist. Diese Komponente kann ebenfalls über das COM-Interface angesprochen werden (siehe *Format*-Subsystem).

Platform

Das *Platform*-Subsystem enthält generische Implementierungen oft benötigter Basisfunktionen (z.B. für Buffer, Stringmanipulation, File und Console I/O, etc.).

Platform/Unix (nur Unix und Derivate)

In *Platform/Unix* sind die *Platform*-Implementierungen für Unix (und Derivate) zu finden. Diese umfassen z.B. File/Filesystem, Pipes, Logging, Threading und Time.

Volume (nur Unix und Derivate)

Das *Volume*-Subsystem stellt eine Abstraktion eines TrueCrypt-Volumes sowie seiner Bestandteile dar und beinhaltet gleichzeitig eine Abstraktion und Implementierung der nötigen Betriebsmodi für die Kryptoalgorithmen (Hierfür werden die kryptographischen Primitive aus dem *Crypto*-Subsystem verwendet). Das Volume-Subsystem verwendet Definitionen aus dem *Common*-Subsystem (welches die Volumeabstraktion für Windows enthält), verwendet aber nicht dessen Implementierung.

1.1.6 Use Case Realization

Volume erzeugen

Über GUI oder Kommandozeile werden zunächst die nötigen Parameter der Operation festgelegt:

- Dateibasiertes oder diskbasiertes Volume

- Standard- oder Hidden-Volume (siehe ***Verstecktes Volume erzeugen***)
- Verschlüsselungs- und Hashalgorithmus
- Volumegröße bei dateibasiertem Volume
- Passwort oder zu verwendendes Keyfile
- Zu verwendendes Dateisystem

Unter Windows ist hier konkret die Anwendung `Tcformat` aus dem *Format*-Subsystem zuständig, unter unixartigen Systemen ist das *Main*-Subsystem zuständig. Sind alle notwendigen Daten vorhanden, kann das Volume angelegt und ggf. formatiert werden.

Unter Linux wird hierzu die Komponente `VolumeCreator` aus dem *Core*-Subsystem für die Erzeugung des Volumes angesteuert. Diese übernimmt auch die Abwicklung der notwendigen kryptografischen Schritte (Erzeugung des Master-Keys, Salting, Headerverschlüsselung, etc.). Ist eine Formatierung gewünscht, so wird diese entweder durch die Komponente `FatFormatter` (für FAT) ebenfalls aus dem *Core*-Subsystem oder durch Methode `CreateVolume` im `TextUserInterface` (für ext2 u.a.) im Anschluss durchgeführt.

Unter Windows übernimmt die `Format`-Komponente aus dem *Common*-Subsystem sowohl die Erzeugung als auch die Formatierung des Volumes. Die Erzeugung des Volume-Headers (inkl. der oben genannten kryptografischen Schritte) wird hierbei von der Volume-Komponente im gleichen Subsystem übernommen.

Verstecktes Volume erzeugen

Die notwendigen Schritte um ein verstecktes Volume zu erzeugen sind bis auf wenige Abweichungen identisch zum vorhergehenden Kapitel ***Volume erzeugen***:

- Der Datenteil des versteckten Volumes wird innerhalb eines normalen Volumes erzeugt und am Ende dieses Volumes auf einen kontinuierliche Bereich freien Speichers abgelegt.
- Der Header des versteckten Volumes wird direkt hinter den Header des normalen Volumes abgelegt. Dieser Platz ist in jedem Fall für den versteckten Header reserviert und, sofern kein verstecktes Volume vorhanden ist, mit Zufallsdaten gefüllt.

Beide Teile sind äußerlich gesehen nicht von Zufallsdaten unterscheidbar, lediglich die Headerposition des versteckten Volumes ist festgelegt.

Bestehende Partition verschlüsseln (nur Windows)

Unter Windows können auch Partitionen verschlüsselt werden, die bereits Daten enthalten. Hierzu werden von der `InPlace`-Komponente im *Format*-Subsystem folgende Schritte ausgeführt:

- Verkleinerung des Dateisystems mittels der zugehörigen Windowsfunktion: Hierbei wird das Dateisystem soweit verkleinert, dass für insgesamt 4 TrueCrypt-Header (2 am Anfang und 2 am Ende als Sicherung) Platz entsteht. Voraussetzung hierfür ist, dass genug freier Speicher zur Verfügung steht. Ist der Speicher so fragmentiert, dass nicht genügend freier Speicher am Ende der Partition zur Verfügung steht, so versucht TrueCrypt selbst, die Daten zu verschieben, bevor eine Verkleinerung vorgenommen wird.
- Erzeugen des Headers im Speicher (genau wie bei der Erzeugung eines normalen Volumes) und Schreiben des Backup-Headers. Der Original-Header wird erst nach Abschluss der gesamten Operation geschrieben.
- Umwandlung (Verschlüsselung) der Partition, beginnend am Ende. Hierzu wird in jedem Schritt ein Klartextblock gelesen, verschlüsselt und dann einen Block hinter die ursprüngliche Position geschrieben. Wie weit die Verschlüsselung bereits durchgeführt wurde, wird im Header nach jeder Blockoperation festgehalten. Bei einem Ausfall des Systems während der Operation gehen so keinerlei Daten verloren und der Prozess kann auch nach Unterbrechung jederzeit mit den Informationen im Backup-Header wieder aufgenommen werden.
- Schreiben des Headers am Anfang der Partition.

Systempartition oder -festplatte verschlüsseln (FDE/PBA) (nur Windows)

Das Einrichten einer verschlüsselten Systempartition oder -festplatte wird über die Anwendung `Tcformat` aus dem *Format*-Subsystem gesteuert. Die tatsächliche Implementierung der einzelnen benötigten Schritte während des Setups implementiert die Klasse `BootEncryption` im *Common*-Subsystem.

Zunächst ist zu bemerken, dass im Gegensatz zur Verschlüsselung normaler Partitionen keine Verkleinerung der Systempartition erfolgt, wenn diese verschlüsselt werden soll. Der Grund hierfür ist, dass der TrueCrypt Header nicht in die Systempartition geschrieben werden muss.

Die einzelnen Schritte, die von `BootEncryption` ausgeführt werden, sind:

- Sicherung des Systembootloaders
- Erzeugung des Volume-Headers im Speicher, ähnlich wie bei der Erzeugung normaler Volumes. Abweichend ist lediglich eine **verringerte Rundenzahl** für die PBKDF2 Key Derivation (hier werden nur 1000 statt den üblichen 2000 Zyklen verwendet), vermutlich weil in der Pre-Boot Authentication die normale Zyklenzahl eine zu hohe Verzögerung für den Benutzer bringen würde.
- Installation des TrueCrypt-Bootloaders (Sektoren 0 bis 62)
- Installation des Volume-Headers im letzten Sektor des TrueCrypt-Bootloaders
- Registrierung des TrueCrypt-Gerätetreibers als Boot-Treiber

Danach wird zunächst das System neu gestartet um zu testen, ob der Bootloader korrekt arbeitet. Nach dem Systemstart wird sofort wieder TrueCrypt gestartet,

welches dann nach Bestätigung ein Signal an den TrueCrypt-Gerätetreiber zum Starten der Verschlüsselung gibt. Daraufhin beginnt der TrueCrypt-Gerätetreiber mit der Verschlüsselung der Systempartition bzw. -festplatte. Der eigentliche Vorgang der Umschlüsselung verläuft genau wie im vorherigen Punkt für normale Partitionen, d.h. die Verschlüsselung wird vom Ende zum Anfang durchgeführt und der Fortschritt wird im Header gespeichert.

Volume öffnen

Generell wird zum Öffnen eines Volumes zunächst aus dem Passwort (und evtl. einem bzw. mehreren Keyfiles) des Benutzers sowie dem *Salt* (Plaintext am Anfang des Headers) das Secret (entsprechend *PKCS #5 PBKDF2*) abgeleitet. Mit diesem Secret wird der erste Header des Volumes entschlüsselt. Ob diese Operation erfolgreich war, wird mittels zwei *CRC32* Prüfsummen im entschlüsselten Header überprüft. Da weder der zu verwendende Hash für *PBKDF2*, noch der kryptografische Algorithmus und sein Betriebsmodus bekannt sind, testet TrueCrypt alle möglichen Kombinationen dieser Parameter durch.

Konnte der Header erfolgreich geöffnet werden und wurde die *Hidden Volume Protection* als Option für das Öffnen aktiviert, so wird mit dem angegebenen *Hidden Volume Password* der gesamte Vorgang (Schlüsselableitung, Entschlüsselung) für den zweiten Header wiederholt. War dieser Vorgang ebenfalls erfolgreich, so wird mittels der Informationen im *Hidden Volume Header* der Bereich des versteckten Volumes von TrueCrypt als schreibgeschützt betrachtet (was unter Windows durch den Gerätetreiber und unter Linux durch den *FUSE Treiber* umgesetzt wird).

Dann wird das geöffnete Volume zur Verfügung gestellt (siehe Volume einhängen).

Konnte im ersten Schritt der Header nicht erfolgreich entschlüsselt werden, so wird die gesamte Operation für den zweiten Header wiederholt. Sind die Schritte für den zweiten Header erfolgreich, so wird das versteckte Volume geöffnet und zur Verfügung gestellt. Sind die Schritte ebenfalls nicht erfolgreich, so erhält der Benutzer eine Fehlermeldung.

Die komplette Abfolge dieses Vorgangs ist auch in Bild 3, der Unterprozess für die Entschlüsselung des Headers in Bild 4 beschrieben.

VS – NUR FÜR DEN DIENSTGEBRAUCH

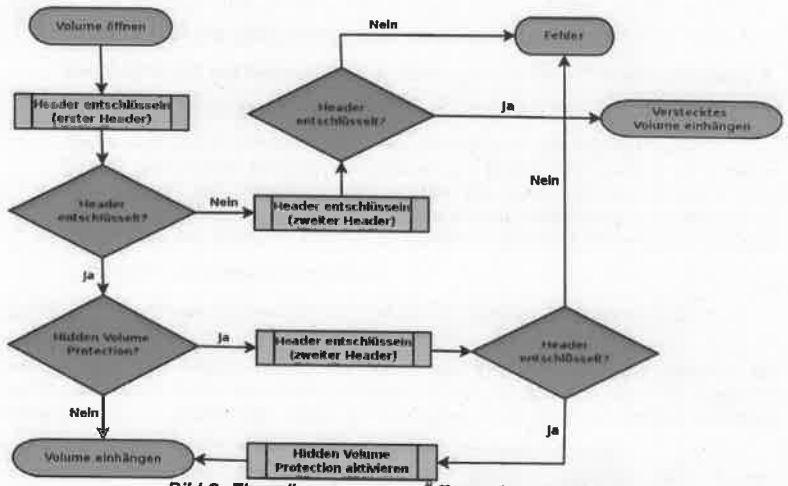

Bild 3: Flussdiagramm zum Öffnen eines Volumes

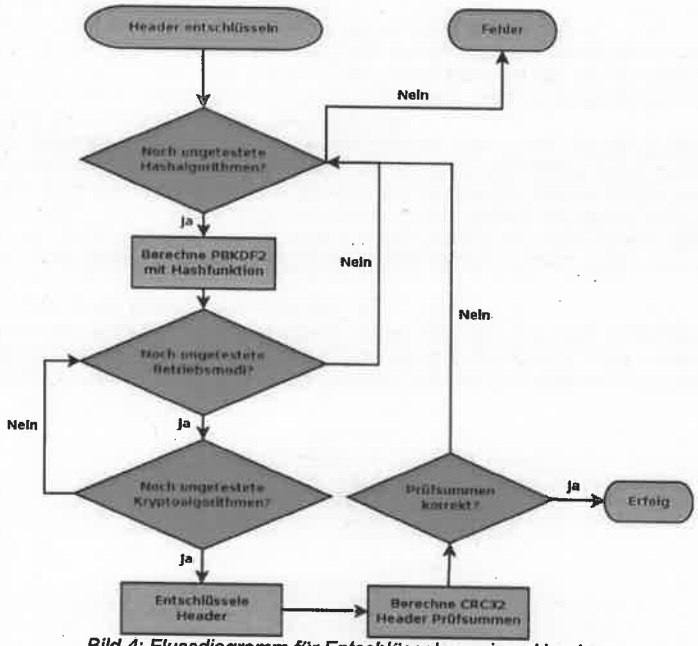

Bild 4: Flussdiagramm für Entschlüsselung eines Headers

Volume einhängen

Wurde ein Volume erfolgreich geöffnet, so muss es danach noch betriebssystemabhängig eingehängt werden, um dem Nutzer zur Verfügung zu stehen.

Unter Windows wird das Einhängen direkt durch den Gerätetreiber durchgeführt, sodass das Volume dann als eigenes Laufwerk zur Verfügung steht.

Unter Linux hängt der Vorgang davon ab, ob der *Linux Device Mapper* genutzt werden soll (wenn kryptografische Algorithmen des Kernels genutzt werden sollen), oder die *FUSE*-Implementierung von TrueCrypt.

Wird der Linux Device Mapper verwendet, so wird zunächst ein *Loop Device* aufgesetzt, falls das Volume dateibasiert ist (für Partitionen/Geräte entfällt dieser Schritt). Dann wird der Device Mapper für das Volume konfiguriert (erhält u.a. das *Master Secret* aus dem entschlüsselten Header sowie Informationen über den verwendeten Algorithmus/Betriebsmodus) und das entstehende Mapper Device eingehängt.

Wird die FUSE-Implementierung genutzt, so wird vor dem eigentlichen Öffnen des Volumes zunächst ein Verzeichnis im temporären Verzeichnis des Systems angelegt, dorthin das TrueCrypt-FUSE-Dateisystem gemountet und mit den nötigen Parametern zum Öffnen versorgt. War das Öffnen erfolgreich, so wird für eine spezielle Datei in diesem Dateisystem ein Loop Device aufgesetzt, dass dann eingehängt wird.

Anmerkung: Wurde im Schritt Volume öffnen ein Volume mit Hidden Volume Protection angefordert, so wird **immer** die FUSE-Implementierung zum Einhängen benutzt, weil nur mit dieser Implementierung der Schutz des Hidden Volumes vor Überschreiben sichergestellt werden kann.

Volumezugriff Lesen/Schreiben

Der Datenfluss beim Lese- und Schreibzugriff hängt davon ab, welches Betriebssystem mit welchen Einstellungen zum Einsatz kommt (siehe auch ***Volume öffnen***).

Unter **Windows** läuft die gesamte Abwicklung über den TrueCrypt-Gerätetreiber. Hierfür registriert der Treiber sogenannte *Filter* für den Zugriff auf Volumes und Devices. Das Filtersystem ist ein natives Interface vom Windows Device Driver Kit um I/O von Devices oder Volumes durch eine zusätzliche Implementierung zu filtern. TrueCrypt nutzt dieses Interface, um die nötigen I/O-Operationen durchzuführen, sich Zustände der einzelnen Geräte zu merken und die Devices/Volumes betreffende Kerneloperationen zu implementieren.

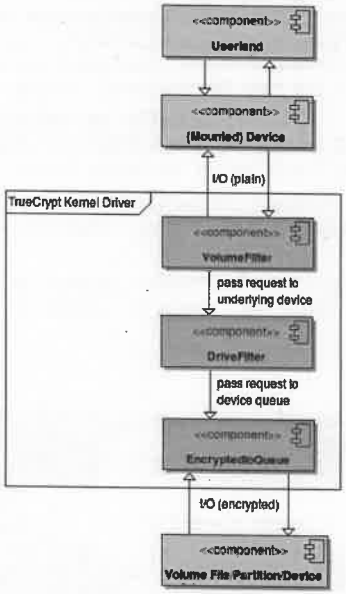

Bild 5: I/O Schichten für TrueCrypt unter Windows

Um die Ver- bzw. Entschlüsselung abzuwickeln, fügt der *DriveFilter* den I/O Request in die *EncryptedIoQueue* ein, die dem Gerät/Volume zugeordnet ist. Dort werden die eigentlichen kryptografischen Aktionen sowie die I/O auf das verschlüsselte Device/File durchgeführt (siehe auch Bild 5).

Unter **Linux** ist der Betriebsmodus mit dem *Linux Device Mapper* (Bild 6 links) von dem mit *FUSE* (Bild 6 rechts) zu unterscheiden. Kommt der Linux Device Mapper zum Einsatz, so läuft die I/O direkt über diesen Device Mapper im Kernel. Dort findet die Ver- bzw. Entschlüsselung statt. Handelt es sich um eine verschlüsselte Partition oder Disk, so findet dann die verschlüsselte I/O direkt mit diesem Gerät statt. Bei einem dateibasierten Volume ist ein Umweg über ein *Linux Loop Device* erforderlich, welches die Datei als Device zur Verfügung stellt.
Kommt statt dem Device Mapper jedoch *FUSE* zum Einsatz, so findet die I/O immer auf einem *Loop Device* statt. Dieses Loop Device leitet die I/O auf eine spezielle Datei „volume" weiter, die sich im zuvor eingehängten FUSE Dateisystem für dieses Volume befindet. Die TrueCrypt-FUSE-Implementierung kann Lese- und Schreibzugriffe auf diese Datei abfangen und so intern die Ver- und Entschlüsselung, sowie die I/O auf das verschlüsselte Volume durchführen.

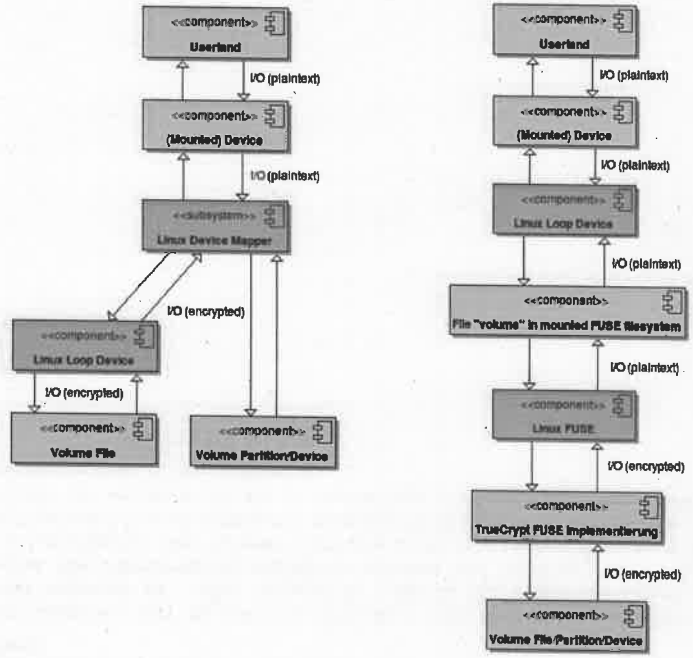

Bild 6: I/O Schichten unter Linux mit Device Mapper (links) und FUSE (rechts)

Passwort ändern

Das Ändern des Passworts entspricht bei TrueCrypt einer Umverschlüsselung des entsprechenden Headers. Das bisherige Passwort ist also zwingend erforderlich, da sonst die entschlüsselte Version des Headers für eine Neuverschlüsselung nicht vorliegt. Folgende Schritte werden für die Änderung des Passworts ausgeführt:

- Eingabe des alten und neuen Passworts
- Auswahl der Hashfunktion für PBKDF2
- Erzeugung eines neuen Salts
- Neuverschlüsselung des Headers

Unter Windows liegt die Implementierung hierfür im *Common*-Subsystem in der `Password`-Komponente (Funktion `ChangePwd`).

Unter Linux liegt die Kernimplementierung hier in der Klasse `CoreBase` im *Core*-Subsystem (Funktion `ChangePassword`) unter Verwendung der `ReEncryptHeader`-Funktion in der `Volume`-Klasse.

Header sichern/wiederherstellen

Die Funktionalität zum Sichern und Wiederherstellen von Volume-Headern ist auf Windows und Linux getrennt implementiert, beide Implementierungen sind jedoch semantisch ähnlich und führen folgende Aktionen zum Sichern durch:

- Entschlüsseln des bisherigen Headers
- Neuverschlüsselung des bisherigen Headers mit neuem Salt
- Falls ein Hidden Volume vorhanden ist, werden die beiden Aktionen für den Hidden Volume Header wiederholt, ansonsten wird dieser Platz im Backup mit Zufallsdaten initialisiert (der Nutzer muss angeben, ob ein Hidden Volume vorhanden ist).
- Die neu verschlüsselten Header werden in einer externen Datei gespeichert.

Zum Wiederherstellen kann der User zunächst wählen, ob aus dem internen Backup (am Ende des Volumes), oder einer externen Backupdatei wiederherstellen möchte. Der Header wird, wie beim Sichern auch, zunächst geöffnet und dann mit neuem Salt verschlüsselt geschrieben.

Unter Windows befindet sich die Implementierung hierfür im *Mount*-Subsystem in den Funktionen `Backup/RestoreVolumeHeader()`.

Unter Linux liegt die Implementierung stattdessen im *Main*-Subsystem in der Klasse `TextUserInterface` (Funktionen `Backup/RestoreVolumeHeaders()`).

Sicheres Löschen/Überschreiben

TrueCrypt implementiert in der Wipe-Komponente des Common-Subsystems mehrere Algorithmen zum sicheren Löschen von Daten (Übersicht der Algorithmen in 2.12). Diese werden jedoch ausschließlich verwendet, um bei der Verschlüsselung bestehender Daten die Klartextdaten zu löschen, bevor sie mit der verschlüsselten Variante überschrieben werden.

Siehe Sicheres Löschen von Volumes für eine Erweiterung von TrueCrypt zum sicheren Löschen von Containern/Headern.

1.1.7 Designbewertung und Auffälligkeiten

Auffällig am Design ist, dass es praktisch aus zwei Designs besteht. Vermutlich wurde das erste Design (wenig objektorientiert) nur für Windows geschrieben. Es besteht hauptsächlich aus C-Code und wird bis zum jetzigen Zeitpunkt weiterhin für Windows verwendet. Das zweite Design wurde offenbar nachträglich hinzugefügt,

nachdem TrueCrypt auf andere Betriebssysteme, wie z.B. Unix und Derivate, portiert werden sollte. Dieses Design ist deutlich stärker objektorientiert und darauf ausgerichtet, unterschiedliche Plattformen einfach zu unterstützen. Es ist nahezu vollständig in C++ geschrieben.

Es ist zu vermuten, dass der jetzige Windowscode zu einem späteren Zeitpunkt ebenfalls in das neue Design integriert werden soll.

1.1.8 Sicherheitsrelevante Designschwächen

Least Privilege Prinzip

TrueCrypt bietet sowohl unter Windows als auch unter Unix die Möglichkeit, sich für bestimmte Aktionen erhöhte Rechte zu beschaffen. Unter Windows wird hierfür UAC (User Account Control) verwendet, unter unixartigen Systemen führt TrueCrypt mittels „sudo" eine zweite Instanz von sich selbst aus, welche bedingt durch spezielle Parameter nur einen „CoreService" betreibt, der dann bestimmte Aktionen mit Rootrechten ausführt. Hierfür muss zumindest das Ausführen von TrueCrypt mit dem entsprechenden Parameter mit „sudo" für den User erlaubt sein.

Problematisch ist dies, wenn TrueCrypt in einer Umgebung eingesetzt wird, in der mehrere User das Produkt (auf dem gleichen Rechner) verwenden sollen. In einer solchen Situation darf kein User vollen Rootzugriff erhalten. Es liegt aber nahe, den Usern per „sudo" generell das Ausführen von TrueCrypt zu erlauben. In diesem Falle prüft TrueCrypt **nicht**, ob erhöhte Privilegien nicht benötigt werden und startet ggf. auch Browser und andere Anwendungen als root (siehe auch *POS02-C*, Cert Secure Coding Standard).

Ein Konfigurationsfehler kann so sehr einfach zu vollem Rootzugriff durch TrueCrypt führen. Die Sicherheit des CoreService-Interfaces muss gesondert geprüft werden (siehe 1.2.5).

Bewertung: In einer Mehrbenutzerumgebung besteht das Risiko einer Falschkonfiguration durch administratives Personal (**moderat**). In einer Umgebung, in der der Nutzer sowieso vollen Rootzugriff erhalten soll, ist das Problem nicht relevant.

CoreService Designfehler

Wie im letzten Abschnitt bereits beschrieben, ist der sogenannte *CoreService* der Teil von TrueCrypt, der letztendlich unter Rootrechten läuft, falls dies erforderlich ist. Dazu muss dieser Teil (also mindestens der Befehl `truecrypt --core-service`) mittels `sudo` startbar sein. Wie auch schon zuvor beschrieben, ist es hier wichtig, dass jeder einzelne Nutzer keine vollen Rootrechte hat, sofern mehrere Nutzer das Produkt am gleichen Rechner nutzen sollen.

Das Design des CoreService erlaubt es jedoch jedem Benutzer, der den *CoreService* mit Rootrechten mittels `sudo` ausführen darf, auch volle Rootrechte zu erlangen: Über die *SetFileOwnerRequest*-Anforderung kann TrueCrypt den CoreService dazu veranlassen, den Besitzer einer bestimmten Datei zu ändern. Dies wird

normalerweise dazu verwendet, um Mountpoints nach dem Einhängen dem Benutzer zugänglich zu machen. Jedoch kann der Benutzer auch selbst beliebige Anforderungen an den CoreService stellen und somit u.a. von jeder beliebigen Datei den Besitzer beliebig ändern.

Bewertung: In einer Mehrbenutzerumgebung kann jeder Benutzer vollen Rootzugriff erhalten (**kritisch**). In einer Umgebung, in der der Nutzer sowieso vollen Rootzugriff erhalten soll, ist das Problem nicht relevant.

1.1.9 Sicherheitsrelevante Codeschwächen

Veraltete/Unsichere Funktionen

In *MSC34-C* des Cert Secure Coding Standard werden eine Reihe von C-Funktionen aufgelistet, die als veraltet oder unsicher gelten und daher nicht mehr verwendet werden sollten. Hierzu zählen u.a. auch bekannte Funktionen wie strcpy, strcat und ihre Widecharacter-Äquivalente. Ein Beispiel, wo eine solche Funktion sogar mit einer Variable verwendet wird, die aus einer anderen Funktion übergeben wird, findet sich z.B. im *Mount*-Subsystem:

```
Mount/Mount.c Line 3363 ff.

static BOOL Mount (HWND hwndDlg, int nDosDriveNo, char
*szFileName) {
...
strcpy (PasswordDlgVolume, szFileName);
```

Weiterhin hat TrueCrypt sogar eigene unsichere Funktionen im gleichen Stil, hierzu zählen z.B. z.B. Upper/LowerCaseCopy:

```
Common/Dlgcode.c Line 322 ff.

void UpperCaseCopy (char *lpszDest, const char *lpszSource)
{
        int i = strlen (lpszSource);
        lpszDest[i] = 0;
        while (--i >= 0)
        {
                lpszDest[i] = (char) toupper (lpszSource[i]);
        }
}
```


Diese Funktionen haben keinerlei Informationen über die Größe des Zielpuffers, kopieren aber trotzdem alle Daten des Quellpuffers (siehe *ARR33-C* des Cert Secure Coding Standard).

Die Verwendung solcher Funktionen sollte **generell** vermieden werden. Bekannte Funktionen können über statische Analyse (z.B. flawfinder / rats) gefunden werden. Eigene Implementierungen solcher Funktionen sollten neu geschrieben werden.

Unsichere Umgebungsvariablen / execvp() ohne Pfadangaben

Der Cert Secure Coding Standard gibt vor, dass Umgebungsvariablen grundsätzlich als ungeprüft angesehen werden müssen, und daher vor allem beim Starten von externen Programmen das Environment geprüft werden muss (siehe *ENV03-C*). TrueCrypt fügt hierfür bestimmte Pfade in die PATH-Variable ein, um zu erzwingen, dass Programmaufrufe aus diesem Pfad gewählt werden:

```
Main/Unix/Main.cpp Line 30 ff.

// Make sure all required commands can be executed via
default search path
string sysPathStr = "/usr/sbin:/sbin:/usr/bin:/bin";

char *sysPath = getenv ("PATH");
if (sysPath)
{
        sysPathStr += ":";
        sysPathStr += sysPath;
}
setenv ("PATH", sysPathStr.c_str(), 1);
```

Problematisch ist hier, dass die eigene Pfadvariable weiterhin im PATH enthalten ist. Zwar wird der PATH durch `sudo` in seiner Defaulteinstellung zurückgesetzt (`env_reset`), dennoch sollte die Sicherheit des Codes nicht auf der Konfiguration von sudo beruhen. Alle externen Programme, die von TrueCrypt gestartet werden, werden **ohne Pfadangaben** gestartet. Liegt das Programm also im Suchpfad aus irgendeinem Grund nicht vor, so kann möglicherweise Programmcode des Angreifers ausgeführt werden.

Modprobe Umgebungsvariable

Direkt zusammenhängend zum vorhergehenden Problem steht ein Problem mit `modprobe` und Umgebungsvariablen. Unter Linux liest `modprobe` standardmäßig seine Argumente auch aus der Umgebungsvariable *MODPROBE_OPTIONS*. Da

man über diese Argumente auch den Ort angeben kann, aus dem die jeweiligen Kernelmodule geladen werden sollen, kann man so TrueCrypt dazu veranlassen, eigene Kernelmodule zu laden (TrueCrypt lädt standardmäßig unter Linux die Kernelmodule für den Device Mapper).

In einer Mehrbenutzerumgebung, in der einzelne User keine vollen Rootrechte haben sollen, kann dies eine **kritische** Sicherheitslücke darstellen, sofern `sudo` nicht so konfiguriert ist, dass alle Umgebungsvariablen zurückgesetzt werden (`env_reset`).

Um das Problem zu lösen, muss entweder die Environment-Variable von TrueCrypt selbst gefiltert werden (zu empfehlen), oder ein statischer Kernel verwendet werden, der von Haus aus keinerlei Module erlaubt.

Off-by-one Overflow

Die im *Platform*-Subsystem enthaltene Implementierung von `Process::Execute` enthält einen Off-by-one-Overflow:

```
Platform/Unix/Process.cpp Line 25 ff.
string Process::Execute (const string &processName, const list
<string> &arguments, int timeOut, ProcessExecFunctor *execFunctor,
const Buffer *inputData) {


char *args[32];
if (array_capacity (args) <= arguments.size())
        throw ParameterTooLarge (SRC_POS);


[...]


int argIndex = 0;
if (!execFunctor)
    args[argIndex++] = const_cast <char*> (processName.c_str());


foreach (const string &arg, arguments) {
        args[argIndex++] = const_cast <char*> (arg.c_str());
}
args[argIndex] = nullptr;
```

Im letzten Schritt wird `args[argIndex]` mit NULL überschrieben, wobei `argIndex` hier maximal den Wert 32 haben kann (was einem off-by-one Fehler entspricht). Da `args` als erste Variable auf dem Funktionsstack liegt, wird hier der Stack Frame Pointer überschrieben, was zu einem Crash führen kann. Kann die Adresse NULL

alloziert werden, ist diese Schwachstelle auch exploitbar.

1.1.10 Bewertung

Auffällig ist, dass die meisten Designschwächen im Linuxcode (d.h. im neuen Design) zu finden sind. Dies ist hauptsächlich dadurch erklärbar, dass zum einen der Code sehr viel neuer und jünger ist, zum anderen aber auch die Entwickler offenbar mehr Erfahrung mit der Windowsplattform haben.

Bei einer oberflächlichen Analyse des Windows Gerätetreibers, insbesondere der extern zugänglichen I/O Befehle, konnten bislang keine nennenswerten Designschwächen entdeckt werden.

Die Existenz von Codeschwächen im Windowscode hingegen ist hauptsächlich dadurch zu erklären, dass als Sprache hier C verwendet wird und der Code gewachsen ist.

1.1.11 Empfehlung

Sowohl Design- als auch Codeschwächen müssen genauer analysiert werden. Ein vollständiger Reviewprozess für alle betroffenen Codestellen, ggf. unterstützt durch statische Analyse muss durchgeführt werden.

Ideal wäre eine Behebung aller Design- und Codeschwächen durch den Einsatz von sicheren Funktionen und der gezielten Abgabe von Privilegien. Eine vollständige Umsetzung des Cert Secure Coding Standards ist anzuraten. Hier sind für die Beseitigung von Codeschwächen (z.B. Verwendung veralteter Funktionen) ca. **8 Personentage** pro Betriebssystem zu veranschlagen. Eine genaue Abschätzung der durchzuführenden Arbeiten ist jedoch ohne ein vollständiges Codereview nicht abschließend durchführbar.

Die Designschwachstelle im CoreService (nur Linux) erfordert vermutlich eine komplette Designüberarbeitung, sofern das Produkt in einer Mehrbenutzerumgebung unter Linux ohne administrativen Zugriff jedes Benutzers eingesetzt werden soll. Hier sind mindestens **14 Personentage** für eine Ausbesserung zu veranschlagen.

1.2 Functional Specification

Im folgenden werden die einzelnen extern sichtbaren Schnittstellen, ihre Parameter, Sicherheitsleistungen und Fehlermeldungen dargestellt.

1.2.1 Entrypoints (Windows)

Wie zuvor beschrieben, besitzt TrueCrypt unter Windows zwei Entrypoints, zum einen die *Mount*-Anwendung, zum anderen den *Formatter*.

1.2.2 Entrypoints (Unix)

Unter unixbasierten Systemen wird die Anwendung ausschließlich über die zuvor beschriebene *Main*-Methode gestartet. Bei Angabe des Kommandozeilenparameters `--core-service` wird der Betrieb als *Core Service* aufgenommen, siehe dazu 1.2.5

. In Abwesenheit dieses Parameters wird, sofern möglich und nicht explizit abgeschaltet (mit `--text`) ein grafisches Benutzerinterface gestartet.

1.2.3 Bootloader (Windows)

Der Bootloader wird für die Pre-Boot Authentication unter Windows verwendet, und bietet die Möglichkeit, sich zu authentifizieren und so ein verschlüsseltes System zu booten, aber auch unverschlüsselte Systeme zu booten. Die möglichen Interaktionen mit dem Bootloader sowie die daraus resultierenden Aktionen sind auch in Bild 7 beschrieben.

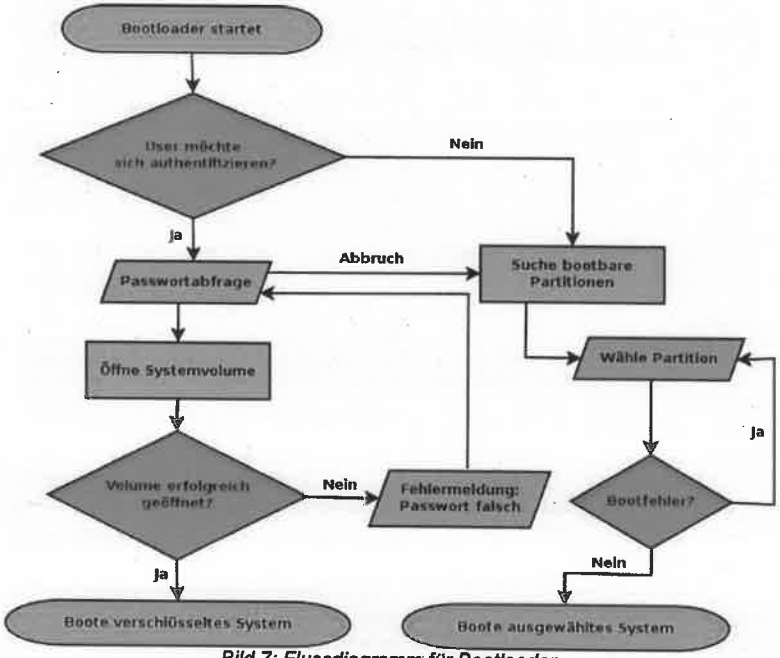

Bild 7: Flussdiagramm für Bootloader

1.2.4 Kernel-Mode-Treiber (Windows)

Der Kernel-Mode-Treiber unter Windows registriert beim Laden mehrere Funktionen, die durch den Kernel angesprochen werden:

- `DriverAddDevice`

- `TCDispatchQueueIRP` (Handler für DeviceIOControl())
- `TCUnloadDriver` (aufgerufen, wenn Treiber entladen wird)

Die Funktion `TCDispatchQueueIRP` ist hier besonders interessant, da sie alle User-Mode-Anfragen behandelt. Sie unterscheidet verschiedene Kategorien von IOControl-Befehlen, die durch separate Funktionen behandelt werden:

- `ProcessMainDeviceControlIrp()`, wenn ein Aufruf durch `IRP_MJ_DEVICE_CONTROL()` für den Gerätetreiber selbst ist.
- `DriveFilterDispatchIrp()` und
- `VolumeFilterDispatchIrp()`, wenn es für ein Filter Device ist. Die Implementierung ist dann in *Driver/DriveFilter.c* bzw. *Driver/VolumeFilter.c*

Nachfolgende Tabellen enthalten alle Befehle, die der Nutzer durch DeviceIOControl() stellen kann.

Funktionsname	Beschreibung
`IRP_MJ_SHUTDOWN`	Veranlasst den Treiber, den Betrieb einzustellen.
`IRP_MJ_DEVICE_CONTROL`	Gibt einen Device Control Befehl (IOCTL) an das jeweilige Device weiter, siehe unten.
`IRP_MJ_READ / IRP_MJ_WRITE`	Fügt eine Lese- bzw. Schreibanfrage in die EncryptedIOQueue des jeweiligen Devices ein.
`IRP_MJ_FLUSH_BUFFERS`	Veranlasst den Treiber, seine Puffer zu leeren.
`IRP_MJ_PNP`	Wird verwendet, um bestimmte Plug-and-Play Befehle an die Devices weiterzuleiten. (*Weitere Codes diese Kategorie betreffend werden nicht aufgeführt*)

Control Codes aus TCDispatchQueueIRP, DriveFilterDispatchIRP und VolumeFilterDispatchIRP

Funktionsname	Beschreibung
`IOCTL_DISK_IS_WRITABLE`	*Nur bei laufendem versteckten System*: Bei der ersten Abfrage dieses Befehls (in der Regel das Systemdevice) wird dieses gespeichert, für alle anderen wird hier zurückgegeben, dass das angefragte Device schreibgeschützt ist.
`TC_IOCTL_DISK_IS_WRITABLE`	*Nur bei laufendem versteckten System*: Prüft, ob ein angefordertes Gerät schreibbar ist.

Zusätzliche Control Codes aus VolumeFilterDispatchIRP

Funktionsname	Beschreibung
IRP_MJ_SHUTDOWN	Veranlasst den Treiber, den Betrieb einzustellen.
IRP_MJ_DEVICE_CONTROL	Gibt einen Device Control Befehl (IOCTL) an das jeweilige Device weiter, siehe unten.
IRP_MJ_READ / IRP_MJ_WRITE	Fügt eine Lese- bzw. Schreibanfrage in die EncryptedIOQueue des jeweiligen Devices ein.
IRP_MJ_FLUSH_BUFFERS	Veranlasst den Treiber, seine Puffer zu leeren
IRP_MJ_PNP	Wird verwendet, um bestimmte Plug-and-Play Befehle an die Devices weiterzuleiten.

Control Codes aus TCDispatchQueueIRP /

Funktionsname	Beschreibung
TC_IOCTL_MOUNT_VOLUME	Öffnet das angeforderte Volume und mountet es.
TC_IOCTL_DISMOUNT_VOLUME	Unmountet das angeforderte Volume und schließt es.
TC_IOCTL_GET_DRIVER_VERSION	Gibt die Gerätetreiberversion zurück.
TC_IOCTL_LEGACY_GET_DRIVER_VERSION	Gibt die Gerätetreiberversion zurück (veraltet).
TC_IOCTL_GET_BOOT_LOADER_VERSION	Gibt die Bootloaderversion zurück.
TC_IOCTL_DISMOUNT_ALL_VOLUMES	Unmountet alle Volumes, auf die der Nutzer Zugriff hat und schließt sie.
TC_IOCTL_GET_MOUNTED_VOLUMES	Gibt eine Liste aller offenen Volumes zurück, auf die der Nutzer Zugriff hat.
TC_IOCTL_GET_VOLUME_PROPERTIES	Gibt Informationen über ein Volume zurück, auf dass der Nutzer Zugriff hat.
TC_IOCTL_GET_DEVICE_REFCOUNT	Gibt die Anzahl der Referenzen auf das angeforderte Device zurück.
TC_IOCTL_WAS_REFERENCED_DEVICE_DELETED	Liefert zurück, ob ein Gerät entfernt wurde, auf das noch Referenzen existierten.

Funktionsname	Beschreibung
TC_IOCTL_IS_ANY_VOLUME_MOUNTED	Liefert zurück, ob irgendein Volume auf dem System geöffnet ist.
TC_IOCTL_GET_PASSWORD_CACHE_STATUS	Liefert zurück, ob der Passwortcache leer ist oder nicht.
TC_IOCTL_WIPE_PASSWORD_CACHE	Löscht den Passwortcache.
TC_IOCTL_OPEN_TEST	Kann verwendet werden um festzustellen ob ein unverschlüsseltes Dateisystem oder der TrueCrypt Bootloader auf dem Gerät vorhanden ist.
TC_IOCTL_GET_DRIVE_PARTITION_INFO	Liefert Informationen über die Partitionstabelle des Geräts zurück.
TC_IOCTL_GET_DRIVE_GEOMETRY	Liefert Informationen über die Disk Geometry zurück.
TC_IOCTL_PROBE_REAL_DRIVE_SIZE	Gibt die Tatsächliche Größe eines Geräts zurück, auf das der Nutzer Zugriff hat.
TC_IOCTL_GET_RESOLVED_SYMLINK	Liefert das Ziel der angegebenen Symbolischen Verknüpfung auf.
TC_IOCTL_GET_BOOT_ENCRYPTION_STATUS	Liefert Informationen über die Boot Encryption (FDE) und ihren Status zurück.
TC_IOCTL_BOOT_ENCRYPTION_SETUP	Startet das Aufsetzen der Boot Encryption, sofern der Nutzer Zugang zum betreffenden Gerät hat.
TC_IOCTL_ABORT_BOOT_ENCRYPTION_SETUP	Bricht einen laufenden Vorgang des Aufsetzens der Boot Encryption ab, sofern der Nutzer Zugang zum betreffenden Gerät hat.
TC_IOCTL_GET_BOOT_ENCRYPTION_SETUP_RESULT	Liefert zurück, ob das Aufsetzen der Boot Encryption erfolgreich war.
TC_IOCTL_GET_BOOT_DRIVE_VOLUME_PROPERTIES	Gibt Informationen über das Boot Volume zurück.
TC_IOCTL_REOPEN_BOOT_VOLUME_HEADER	Öffnet das Boot Volume erneut, sofern der Nutzer Zugriff hat (nach Passwortänderung der Boot Encryption).
TC_IOCTL_GET_BOOT_ENCRYPTION_ALGORITHM_NAME	Liefert den Verschlüsselungsalgorithmus des Boot Volumes zurück.
TC_IOCTL_GET_PORTABLE_MODE_STATUS	Liefert den „PortableMode" Status zurück (Treiber wird entladen, wenn nicht mehr gebraucht).
TC_IOCTL_SET_PORTABLE_MODE_STATUS	Setzt den „PortableMode" Status für den Treiber, sofern der Nutzer Zugriff auf Geräte hat.

Funktionsname	Beschreibung
TC_IOCTL_IS_HIDDEN_SYSTEM_RUNNING	Liefert zurück, ob das Betriebssystem aus einem Hidden Volume heraus läuft („Hidden System").
TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG	Liefert für ein Gerät die TrueCrypt Bootloader Version und seine Konfiguration zurück.
TC_IOCTL_DISK_IS_WRITABLE	Gibt zurück, ob das Gerät schreibbar ist.
TC_IOCTL_START_DECOY_SYSTEM_WIPE	Startet den Löschvorgang für ein unverschlüsseltes Betriebssystem (nachdem ein verstecktes Betriebssystem installiert wurde).
TC_IOCTL_ABORT_DECOY_SYSTEM_WIPE	Bricht den Löschvorgang für ein unverschlüsseltes Betriebssystem ab.
TC_IOCTL_GET_DECOY_SYSTEM_WIPE_STATUS	Gibt den Status des Löschvorgangs für ein unverschlüsseltes Betriebssystem zurück.
TC_IOCTL_GET_DECOY_SYSTEM_WIPE_RESULT	Liefert zurück, ob der Löschvorgangs für ein unverschlüsseltes Betriebssystem erfolgreich war.
TC_IOCTL_WRITE_BOOT_DRIVE_SECTOR	Schreibt einen neuen Bootsektor auf das Gerät, sofern der Nutzer Zugriff auf Geräte hat.
TC_IOCTL_GET_WARNING_FLAGS	Liefert bestimmte Warnungen zurück, die der Treiber gespeichert hat.
TC_IOCTL_SET_SYSTEM_FAVORITE_VOLUME_DIRTY	Markiert ein Volume als „Dirty" (nicht richtig geschlossen/unmounted), sofern der Nutzer Zugriff auf Geräte hat.
TC_IOCTL_REREAD_DRIVER_CONFIG	Veranlasst den Treiber, seine Einstellungen aus der Registry neu einzulesen.
TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG	Liefert einen Dump der Konfiguration des Systemdevice zurück (*bei Pre-Boot Authentication*)
TC_IOCTL_LEGACY_GET_MOUNTED_VOLUMES	Veralteter Befehl, wird nicht mehr bedient.

Control Codes aus ProcessMainDeviceControlIrp

Für alle Befehle gibt es verschiedene Arten der Zugriffskontrolle:

1. Explizite Zugriffskontrolle durch den Treiber
2. Implizite Zugriffskontrolle durch das Betriebssystem
3. Keine Zugriffskontrolle

Art der Zugriffskontrolle	Funktionen
Explizite Zugriffskontrolle durch den Treiber (1).	TC_IOCTL_SET_PORTABLE_MODE_STATUS , TC_IOCTL_PROBE_REAL_DRIVE_SIZE , TC_IOCTL_SET_SYSTEM_FAVORITE_VOLUME_DIRTY , TC_IOCTL_GET_MOUNTED_VOLUMES , TC_IOCTL_GET_VOLUME_PROPERTIES , TC_IOCTL_DISMOUNT_VOLUME , TC_IOCTL_DISMOUNT_ALL_VOLUMES, TC_IOCTL_BOOT_ENCRYPTION_SETUP, TC_IOCTL_ABORT_BOOT_ENCRYPTION_SETUP, TC_IOCTL_START_DECOY_SYSTEM_WIPE , TC_IOCTL_ABORT_DECOY_SYSTEM_WIPE, TC_IOCTL_WRITE_BOOT_DRIVE_SECTOR, TC_IOCTL_REOPEN_BOOT_VOLUME_HEADER
Implizite Zugriffskontrolle durch den Treiber (2).	TC_IOCTL_GET_DRIVE_PARTITION_INFO TC_IOCTL_GET_DRIVE_GEOMETRY TC_IOCTL_MOUNT_VOLUME IRP_MJ_* *auf ein TrueCrypt Device*
Keine Zugriffskontrolle	TC_IOCTL_GET_DRIVER_VERSION , TC_IOCTL_LEGACY_GET_DRIVER_VERSION , TC_IOCTL_GET_DEVICE_REFCOUNT , TC_IOCTL_WAS_REFERENCED_DEVICE_DELETED , TC_IOCTL_IS_ANY_VOLUME_MOUNTED , TC_IOCTL_OPEN_TEST , TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG , TC_IOCTL_WIPE_PASSWORD_CACHE , TC_IOCTL_GET_PASSWORD_CACHE_STATUS , TC_IOCTL_GET_PORTABLE_MODE_STATUS , TC_IOCTL_GET_WARNING_FLAGS , TC_IOCTL_REREAD_DRIVER_CONFIG , TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG , TC_IOCTL_GET_RESOLVED_SYMLINK , TC_IOCTL_GET_BOOT_ENCRYPTION_STATUS , TC_IOCTL_GET_BOOT_ENCRYPTION_SETUP_RESULT ,

Art der Zugriffskontrolle	Funktionen
	TC_IOCTL_GET_BOOT_DRIVE_VOLUME_PROPERTIES , TC_IOCTL_GET_BOOT_LOADER_VERSION , TC_IOCTL_GET_BOOT_ENCRYPTION_ALGORITHM_NAME , TC_IOCTL_IS_HIDDEN_SYSTEM_RUNNING , TC_IOCTL_GET_DECOY_SYSTEM_WIPE_RESULT , TC_IOCTL_GET_DECOY_SYSTEM_WIPE_STATUS

Bei den Befehlen in Kategorie (1) wird explizit geprüft, ob der User Zugang zu einem Device/Volume hat. Ist dies nicht der Fall, schlägt der Befehl entweder komplett fehl, oder es werden nur Teilinformationen angezeigt bzw. Teilaktionen ausgeführt (z.B. nur eigene Devices dismountet obwohl ein `DISMOUNT_ALL` angefordert wurde).

Bei den Befehlen in Kategorie (2) hingegen werden entweder Requests an das darunterliegende Treibersystem weitergeleitet, so dass dieses System dann für Zugriffskontrolle zuständig ist, oder es werden Dateifunktionen mit Zugangsprüfung durch das Betriebssystem benutzt.

1.2.5 CoreService Interface (Unix)

Das CoreService-Interface wird von der gleichnamigen Klasse im Core/Unix-Subsystem implementiert und liest Input von stdin, nachdem es mit erhöhten Rechten gestartet wurde. Dabei werden folgende Funktionen unterstützt:

Funktionsname	Beschreibung
`ExitRequest`	Stoppt den CoreService
`CheckFileSystemRequest`	Führt den Befehl `fsck` für das angegebene Dateisystem aus
`DismountFileSystemRequest`	Ruft den Befehl `umount` für den angegebenen Mountpoint auf
`DismountVolumeRequest`	Unmountet und schließt das angegebene Volume
`GetDeviceSectorSizeRequest`	Liefert die Sektorgröße des angegebenen Devices zurück
`GetDeviceSizeRequest`	Liefert die Gesamtgröße des angegebenen Devices zurück
`GetHostDevicesRequest`	Liefert alle verfügbaren Partitionen/Disks zurück
`MountVolumeRequest`	Öffnet das angegebene Volume und hängt es ein
`SetFileOwnerRequest`	Ändert den Besitzer der angegebenen Datei auf den angegebenen Nutzer bzw. die angegebene UID

Die genannten Requests werden über eine eigene Serialisiererimplementierung an den CoreService übergeben. Es finden keine sicherheitsrelevanten Prüfungen statt, der CoreService begreift die Eingaben als vertrauenswürdig, obwohl auch der User direkt den CoreService aufrufen kann. Insbesondere finden keine Rechteprüfungen (z.B. beim Öffnen eines Volumes) statt. Es ist also möglich, auch ein nur von „root" lesbares Volume als User mit TrueCrypt zu öffnen.

1.2.6 Userspace Treiber (Unix mit FUSE)

Wird unter Linux die Verwendung der kryptografischen Algorithmen aus dem Kernel abgeschaltet, so wird stattdessen ein FUSE-Dateisystem verwendet, das TrueCrypt bereitstellt. Das Dateisystem selbst enthält im geöffneten Zustand nur zwei Dateien, zum einen die Datei, die das geöffnete Volume repräsentiert, zum anderen eine spezielle Steuerdatei, die Informationen über das Volume empfängt und auch bereitstellt. Daher sind diese Steuerdatei und das Dateisystem selbst als getrennte Schnittstellen zu betrachten.

Dateisystemfunktionen

Funktionsname	Beschreibung	Parameter	Fehler
`fuse_service_access`	Benutzt durch *access()* Systemcall bei Zugriffsprüfung	Pfad, Modus	1
`fuse_service_getattr`	Liefert Datei- oder Ordnerattribute zurück.	Pfad, Statstruktur für Rückgabe	1,4 nur wenn Pfad nicht „/"
`fuse_service_opendir`	Öffnet ein Verzeichnis	Pfad zu Verzeichnis	1,3
`fuse_service_open`	Öffnet eine Datei	Pfad zu Datei	1,4
`fuse_service_readdir`	Liefert Verzeichnisinhalte zurück	Pfad zu Verzeichnis	1,3
`fuse_service_read`	Liest aus einer geöffneten Datei	Pfad, Größe und Offset der zu lesenden Daten, Puffer für Rückgabe	1,4
`fuse_service_write`	Schreibt in eine geöffnete Datei	Pfad, Datenpuffer, Größe der Daten, Schreiboffset	1,2,4

Die in der obigen Tabelle verwendeten Fehlernummern werden in der folgenden Tabelle beschrieben. Beschrieben sind ausschließlich explizite Prüfungen und Fehler, keine sonstigen (nicht speziell auf Sicherheit) bezogene Exceptions oder Codes.

Fehlernummer (und Code)	Beschreibung
1 (EACCES)	Nutzer ist weder *root*, noch der Nutzer, der das Dateisystem gestartet hat.
2 (EACCES)	Informationen über das geöffnete Volume wurden bereits empfangen
3 (ENOENT)	Angeforderter Pfad ist nicht "/"
4 (ENOENT)	Angeforderte Datei ist weder Volumen- noch Steuerdatei

Race Condition für Fehlercode 2

Wenn ein Device mittels FUSE gemountet werden soll, so wird zunächst das FUSE-Dateisystem auf einem temporären Verzeichnis eingehängt. Dann werden von TrueCrypt bestimmte Informationen über das Volume, nämlich der Name des virtuellen Devices und der Name des Loopdevices übertragen. Zur Übertragung werden diese in die Steuerdatei geschrieben. Das FUSE-Dateisystem nimmt diese Informationen genau einmal an und gibt bei folgenden Schreibversuchen auf die Steuerdatei wie beschrieben Fehlercode 2 zurück. Der Nutzer hat jedoch theoretisch ein kurzes Zeitfenster, in dem er selbst diese Informationen setzen kann. Dies ist insofern problematisch, als das TrueCrypt später diese Informationen für gewisse Aktionen weiterverwendet, z.B.:

- Beim Schließen des Volumes um das Loopdevice zu entfernen
- Bei der Dateisystemprüfung

Beide Aktionen werden mit Systemrechten ausgeführt, die letzte der beiden Aktionen ist geeignet, dem Nutzer **Systemrechte** zu verschaffen.

1.2.7 Security Token

TrueCrypt selbst enthält kein extern zugängliches Interface für Security Token. Der Zugriff auf solche Token erfolgt über eine externe PKCS-#11-Library, die TrueCrypt zuvor bekannt gemacht werden muss. Unter Windows kann bei der Konfiguration eine passende PKCS-#11-Library auch automatisch gesucht werden.

1.3 Security Architecture Description

1.3.1 Encryption Layout/Design

Ein TrueCrypt Volume besteht semantisch aus zwei Teilen, dem *TrueCrypt Header* und einem Bereich für die Nutzdaten (siehe Bild 8). Der *TrueCrypt Header* wird durch das Nutzerpasswort (unter Anwendung von PKCS #5 PBKDF2 und einem unverschlüsselten Salt) verschlüsselt und ist somit nur dem Benutzer mit dem korrekten Passwort zugänglich. Die Nutzdaten sind wiederum mit einem Schlüssel verschlüsselt, der im Header abgelegt ist, somit ist der Zugang zu den Nutzdaten nur

über den Header möglich (*„Non-bypassability"*).

Anmerkung: Entgegen der Darstellung in Bild 8 enthält der Datenträger eine Sicherheitskopie des verschlüsselten Headers, die zur Datenrettung dient, falls der Originalheader überschrieben wurde (z.B. durch eine versehentliche Schnellformatierung). Da diese Kopie im Klartext jedoch identisch zum Originalheader ist, ist sie hier sicherheitstechnisch nicht von Belang. Weiterhin existiert zwischen dem Header und den Nutzdaten noch Platz für einen weiteren Header, der allerdings nur für versteckte Volumes genutzt wird.

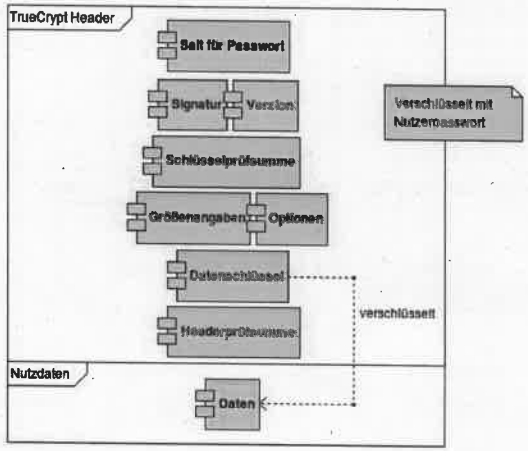

Bild 8: TrueCrypt Volume schematisch dargestellt

1.3.2 Sicherer Zufall

Linux

Das Design für unixartige Systeme enthält im *Core*-Subsystem die Klasse `RandomNumberGenerator`, um sicheren Zufall zur Verfügung zu stellen. Dabei handelt es sich um einen, auf Hashfunktion basierten Zufallszahlengenerator, der seine Entropie sowohl aus dem System als auch aus anderen Quellen beziehen kann. Zufallsdaten können durch zwei verschiedene Funktionen angefordert werden:

- `GetData()`: Hierbei liest der Zufallszahlengenerator zunächst Zufallszahlen vom Kernel aus */dev/random*, danach werden nochmals Daten aus */dev/urandom* bezogen.
- `GetDataFast()`: Hierbei liest der Zufallszahlengenerator beide Male aus */dev/urandom*.

Zusätzlich zur den Entropiequellen */dev/random* und */dev/urandom* wird bei bestimmten Vorgängen (z.B. Schlüsselgenerierung) der Benutzer miteinbezogen. Hierzu werden Interaktionen mit Eingabegeräten (Maus, Tastatur) dem Pool des Zufallszahlengenerators hinzugefügt.

Die Methode `getData()` zur Nutzung von */dev/random* wird verwendet bei den folgenden Aktionen verwendet:

- Initialisierung des RNG
- Erzeugung von Masterschlüssel für kryptographischen Algorithmus
- Erzeugung von Schlüsseldateien
- Erzeugung von Salts (außer im Wipe Modus wenn Header immer wieder überschrieben werden, hier nur im letzten Durchlauf).

Die Methode `getDataFast()` wird lediglich für die folgenden Aktionen verwendet:

- Erzeugung von Salts im Wipe Modus (außer letzte Iteration)
- Erzeugung von zufälliger Volume ID für FAT Dateisystem

Zusätzliche Entropie (`UserEnrichRandomPool`) wird bei den folgenden Vorgängen verwendet:

- Erzeugen von Volumes
- Ändern des Passworts
- Sichern/Wiederherstellen von Headern

Wenn diese Aktionen über die GUI erfolgen, so wird die zusätzliche Entropie aus den Eingabegeräten (Tastatur und Maus) bezogen. Läuft TrueCrypt auf der Kommandozeile, so kann der Nutzer Zufallsdaten selbst eingeben. Im nicht-interaktiven Modus kann auch eine zusätzliche Datei angegeben werden, aus der Entropie bezogen werden soll.

Schwachstelle in der Anbindung von /dev/random unter Linux

Wie bereits beschrieben bezieht die Linuximplementierung des RNG auch Entropie aus /dev/random. Problematisch ist hier, dass die Funktion read() benutzt wird und nicht geprüft wird, wie viel die Funktion überhaupt gelesen hat. Weiterhin liest diese Funktion implementierungsabhängig u.U. eine geringere Anzahl als die Maximalanzahl an Bytes die angegeben wird. Es ist so möglich, dass

- weniger Entropie gelesen wird, als der Puffer groß ist (unter Linux generell 128 Byte), auch wenn genügend Entropie zur Verfügung steht.
- weniger als 128 Byte Entropie gelesen werden, sofern nicht genügend Entropie vorhanden ist.

- **keine** Entropie gelesen wird, wenn keine Entropie vorhanden ist.

Keiner der genannten Fehler wird geprüft oder abgefangen, das Programm arbeitet also ggf. auch ganz ohne Entropie (bis auf Benutzereingaben und */dev/urandom*) weiter.

Windows

Unter Windows befindet sich die Implementierung des Zufallszahlengenerators in *Common/Random.(c|h)*. Ähnlich zur Linuximplementierung basiert der Generator auf einer Hashfunktion und kann wie unter Linux auch die Interaktion des Nutzers mit Eingabegeräten nutzen.

Um Zufallsdaten vom System zu bekommen, setzt die Implementierung jedoch eine vielseitigere Implementierung ein. In der *SlowPoll* Routine werden Zufallsdaten gewonnen aus:

- der Windows Crypto API (`CryptGenRandom()`) sofern vorhanden
- I/O der Festplatten und der Netzwerkkarten

In der *FastPoll* Routine werden Zufallsdaten gewonnen aus:

- Verschiedenen Window Handles
- Eigenschaften der Zwischenablage
- Prozesseigenschaften (Threads, Prozess ID, Heap Handle), Laufzeiten)
- Millisekunden seit Systemstart

Beide Funktionen werden in `RandgetData()` eingesetzt, wobei die *SlowPoll* Routine nur bei der ersten Anforderung von Daten benutzt wird und ab diesem Zeitpunkt nur, wenn der aufrufende Code dies explizit angibt. Dies ist der Fall bei folgenden Operationen:

- Initialisierung des RNG
- Erzeugung von Masterschlüssel für kryptographischen Algorithmus
- Erzeugung von Schlüsseldateien
- Erzeugung von Salts (außer im Wipe-Modus, wenn Header immer wieder überschrieben werden, hier nur im letzten Durchlauf)
- Erzeugung eines Schlüssels für sicheres Löschen (Wipe) bei Einrichtung von verstecktem Betriebssystem
- Teilweise Erzeugung eines zufälligen temporären Schlüssels für die Randomisierung des Puffers bei Sicherung des Volume-Headers

Die Implementierung nutzt hingegen ausschließlich die FastPoll() Methode für:

- Erzeugung zufälliger Schlüssel zum Wipen von Headern
- Erzeugung von FAT Volume ID

- Erzeugung von Zufallsdaten zum Überschreiben von freiem Speicher auf FAT/NTFS Dateisystemen
- Überschreiben der Konfiguration für verstecktes Betriebssystem
- Teilweise Erzeugung von zufälligem temporären Schlüssel für Randomisierung von Puffer bei Sicherung von VolumeHeader

Zusätzlich wird, wie unter Linux auch, für bestimmte Operationen die Interaktion des Benutzers (Maus/Tastatur) in den Prozess miteinbezogen. Außer den bereits bei Linux genannten Operationen wird dies unter Windows auch beim Einrichten eines versteckten Betriebssystems benutzt (zum sicheren Löschen).

1.3.3 I/O Separation

Linux

Wird unter Linux der *Device Mapper* verwendet, so findet das I/O-Handling und die damit verbundenen kryptografischen Operationen im Kernel pro Device statt. Für die Trennung ist der Device Mapper selbst zuständig.

Wird stattdessen *FUSE* verwendet, so laufen alle I/O Requests über das *FUSE*-Interface im Kernel und werden an getrennte TrueCrypt-CoreService-Prozesse im Userspace weitergeleitet, welche dann die kryptografischen Operationen durchführen.

Windows

Unter Windows findet das I/O-Handling im TrueCrypt-Gerätetreiber statt. Hierzu hat jedes Treiberobjekt, das mit einem Volume assoziiert ist, eine eigene *EncryptedIoQueue*, in der I/O-Requests eingereiht und dann von Threads aus einem Pool abgearbeitet werden. Das nötige Schlüsselmaterial liegt pro Queue separat vor.

1.3.4 Cache Design

Sowohl unter Windows, als auch unter Linux bietet TrueCrypt die Möglichkeit, Passwörter zu cachen, damit sie nicht erneut eingegeben werden müssen, wenn z.B. mehrere Volumes nacheinander mit dem gleichen Passwort gemountet werden sollen. Diese Funktionalität muss jedoch beim Öffnen eines Volumes explizit aktiviert werden.

Linux

Unter Linux findet das Caching im laufenden Programm statt. Sobald TrueCrypt geschlossen wird, wird der Passwortcache durch Überschreiben im Speicher gelöscht. Die Änderung wird u.U. nicht sofort auf den RAM übertragen (z.B. durch L2-Cache), vonseiten des Programms ist dies aber nicht weiter beeinflussbar.

Windows

Unter Windows findet das Caching **nicht** in der Programminstanz selbst statt, sondern im Gerätetreiber. Es gibt nur **einen** Cachepool in diesem Treiber, was zur Folge hat, dass sich auf einem System mehrere Benutzer den gleichen Cache teilen, eine Trennung erfolgt hier nicht. Öffnet Benutzer A auf dem Windows-System ein Volume und aktiviert den Passwortcache, so kann Benutzer B unter seinem Account dieses Volume (oder jedes andere mit gleichem Passwort) öffnen, ohne das Passwort einzugeben.

Je nach Einstellung wird der Passwortcache durch Überschreiben im Speicher gelöscht, wenn der Benutzer das Programm schließt, das Volume wieder schließt oder das Volume automatisch geschlossen wird (z.B. bei Systemsperre/Bildschirmschoner).

Zusätzlich kann der Windowstreiber das Passwort der Pre-Boot-Authentication ebenfalls dem normalen Passwortcache hinzufügen und dann nutzen, um normale Volumes zu öffnen, sofern die Option hierfür aktiviert wurde.

1.3.5 Privilege Separation

Sowohl unter Windows als auch unter Linux benötigt nur ein Teil der Implementierung administrativen Zugang (unter Windows auch nur für bestimmte Operationen). Es ist daher notwendig festzustellen, welche Teile des Programms wann unter erhöhten Rechten laufen müssen.

Insbesondere im Hinblick auf Mehrbenutzersysteme (d.h. wenn mehrere Nutzer an einem System die Software einsetzen sollen, nicht notwendigerweise mit dem gleichen Volume) ist dann auch die Sicherheit der Komponenten mit erhöhten Rechten gegenüber dem Nutzer als Angreifer relevant (*Privilege Escalation*).

Linux

Unter Linux wird grundsätzlich für alle Operationen ein Prozess mit Rootrechten benötigt. Dabei handelt es sich um den sogenannten *CoreService* (siehe auch 1.2.5) welcher von TrueCrypt selbst mittels `sudo` gestartet wird. Danach kommuniziert TrueCrypt mit diesem neuen Prozess über `stdin` mittels einer selbst implementierten Befehlsserialisierung. Bild 9 zeigt eine Übersicht über die beteiligten Komponenten.

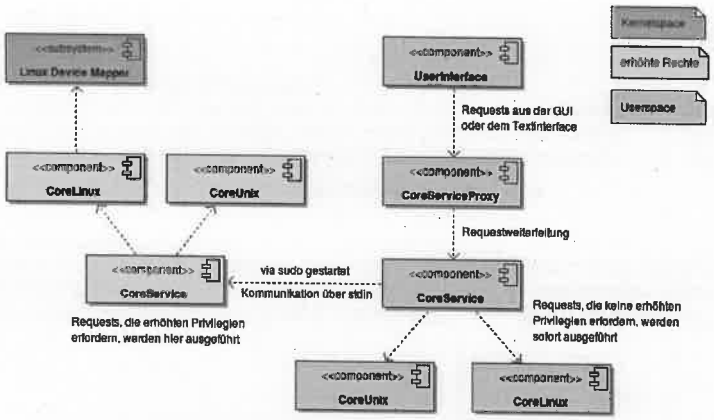

Bild 9: Übersicht der Privilege Separation in Unix/Linux

Wie die Analyse in 1.1.8 bereits zeigte, ist dieser Dienst jedoch nicht sicher gegenüber einem Privilege-Escalation-Angriff durch den Nutzer.

Windows

Das Verhalten unter Windows ist im Gegensatz zu Linux ungleich komplizierter. Grundsätzlich läuft der installierte Gerätetreiber von TrueCrypt immer mit erhöhten Rechten (da er im Kernelspace läuft), kann aber vom Nutzer auch ohne administrative Rechte angesprochen werden. Die Schnittstelle hierfür wird in 1.2.4 erläutert.

Zusätzlich können jedoch auch Teile oder das gesamte Programm mit administrativen Rechten gestartet werden. Unter Windows XP und älteren Windowsversionen gibt es hierfür noch keinen windowseigenen Mechanismus, daher muss das gesamte Programm immer mit administrativen Rechten gestartet werden, sofern die gewünschte Operation dies benötigt. Folgende Operationen benötigen erhöhte Rechte:

- Volume mit NTFS-Dateisystem formatieren (`UacFormatNtfs`)
- Analyse von Dateisystem für Hidden Volumes (`UacAnalyzeHiddenVolumeHost`)
- Falls erforderlich zur Ansteuerung des Gerätetreibers (`CallDriver`)
- Falls erforderlich zum Zugriff auf Devices oder Files (`ReadWriteFile`)
- Registrieren des Services für Systemfavoriten (`RegisterSystemFavoritesService`)
- Kopieren/Löschen von Dateien betreffend die Systemfavoriten

(`Copy/DeleteFileAdmin`)

- Installieren/Registrieren der Filtertreiber (`RegisterFilterDriver`)
- Starteinstellungen für Treiberdienst festlegen (`SetDriverServiceStartType`)
- Prüfen ob Auslagerungsdatei aktiv ist (`IsPagingFileActive`)
- Speichern von Einstellungen in Registry unter HKEY_LOCAL_MACHINE (`WriteLocalMachineRegistryDwordValue`)
- Volumeheader sichern/wiederherstellen falls kein dateibasierter Container (`UacBackupVolumeHeader/UacRestoreVolumeHeader`)
- Kerneldump erzeugen (`UacAnalyzeKernelMinidum`)
- Passwort ändern (`UacChangePwd`) falls erforderlich für Zugriff auf Container oder bei Systempartition

Unter Windows Vista und höher kommt hierfür UAC (User Account Control) zum Einsatz, wodurch nur ein Teil von TrueCrypt mit erhöhten Rechten gestartet wird, wenn es erforderlich ist. Grundsätzlich ist zu beachten, dass es unter Windows nicht vorgesehen ist, dass ein Nutzer mittels UAC nur eine bestimmte Komponente starten kann, d.h. ein Nutzer, der UAC mit TrueCrypt nutzen kann, hat schon volle administrative Rechte. Es ist dennoch sinnvoll, zu analysieren, welche Implementierungsteile letztendlich unter erweiterten Rechten ausgeführt werden können, z.B. um festzustellen, inwiefern Malware mit Nutzerrechten diese Teile angreifen könnte.

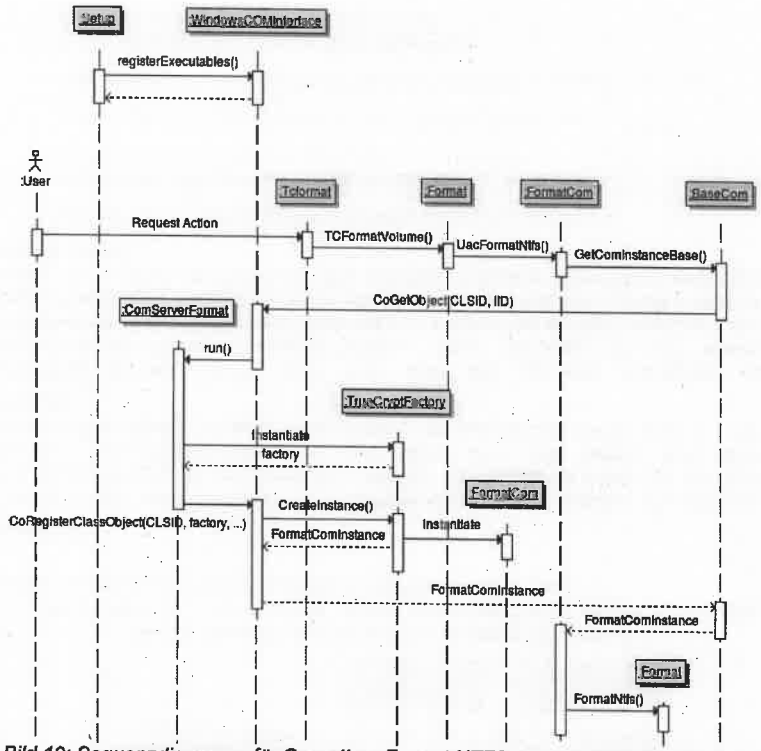

Bild 10: Sequenzdiagramm für Operation „Format NTFS Volume" mit Windows UAC

Das Sequenzdiagramm in Bild 10 beschreibt beispielhaft den vollständigen Ablauf der Operation „Format NTFS Volume" als Teil der Erzeugung eines neuen Volumes mit NTFS Dateisystem.

Analyse der Komponenten bezüglich Angriffen durch Malware

Beim Registrieren des Services für Systemfavoriten wird die ausführbare Datei von TrueCrypt mit Systemrechten kopiert und als Dienst installiert. Hier entsteht u.U. eine Race Condition, wenn Malware die entsprechende Datei umbenennen/schreiben und so durch eine eigene Datei ersetzen kann:

```
char appPath[TC_MAX_PATH];
throw_sys_if (!GetModuleFileName (NULL, appPath,
```



```
    sizeof (appPath)));
throw_sys_if (!CopyFile (appPath, servicePath.c_str(),
    FALSE));
```

Beim Installieren der Pre-Boot-Authentifizierung muss der TrueCrypt Treiber speziell installiert werden. Die hierfür verwendete Routine `RegisterFilter` in der `BootEncryption` Klasse schreibt eine Setupdatei (`.inf`) in das temporäre Verzeichnis des Nutzers, schließt diese Datei und installiert sie anschließend mit administrativen Rechten. Hier entsteht eine Racecondition, in der der Nutzer bzw. Malware unter Nutzerrechten diese Datei verändern könnte, bevor sie durch TrueCrypt installiert wird.

Beide Operationen werden nur während der Installation einer Full-Disk Encryption ausgeführt. Es muss also sichergestellt werden, dass das System bei diesem Prozess frei von schädlicher Software ist.

1.3.6 Secure Startup

TrueCrypt enthält keinerlei Vorkehrung zur sicheren Prüfung seiner Komponenten beim Start des Systems. Ein Angriff gegen die Pre-Boot-Authentication existiert bereits seit geraumer Zeit (siehe auch „Stoned" Bootkit vorgestellt in "Stoned bootkit White Paper" (PDF). *Black Hat Technical Security Conference USA 2009*.)

1.3.7 Auslagerung im Betrieb und bei Ruhezustand

Windows

Bei *aktiver FDE* prüft TrueCrypt, dass die Auslagerungsdatei auch auf dem verschlüsselten Device liegt. Eine Auslagerung erfolgt ausschließlich auf das verschlüsselte Device selbst.

Ist *keine FDE* aktiv, so scheint TrueCrypt in seiner Standardeinstellung nicht zu verhindern, dass durch das Wechseln in den Ruhezustand sensible Daten auf die unverschlüsselte Festplatte geschrieben werden. Es besteht jedoch die Möglichkeit, vor der Aktivierung des Ruhezustands alle offenen Volumes automatisch zu schließen. Genauere Analysen und Angriffe hierzu werden in AP3 und AP7 betrachtet.

Im laufenden Betrieb verwendet TrueCrypt für Daten in *EncryptedIoQueue* die Funktion `MmGetSystemAddressForMdlSafe` um nicht-swapbaren Speicher zu allozieren und dort Klartext zu speichern. Außerhalb des Kerneltreibers wird die Funktion `VirtualLock` verwendet, um Variablen mit sensiblem Inhalt vor der Auslagerung zu schützen.

Linux

Unter Linux sind **keinerlei Mechanismen erkennbar**, die sensible Daten vor einer

Auslagerung im Betrieb oder im Ruhezustand schützen würden. Hierfür nötig wäre die Verwendung von Funktionen wie `mmap` bzw. `mlock`, diese sind jedoch nicht vorhanden. Angriffe hierzu werden in AP3 und AP7 betrachtet.

2 Modularisierbarkeit / Erweiterbarkeit von TrueCrypt

2.1 Kryptoalgorithmen

2.1.1 Beschreibung

TrueCrypt implementiert unter allen Betriebssystemen folgende Kryptoalgorithmen:

- AES-256
- Serpent-256
- Twofish-256
- Blowfish-448 (als veraltet/legacy markiert)
- TripleDES (als veraltet/legacy markiert)
- CAST-128 (als veraltet/legacy markiert)

Zusätzlich wird für AES die Verwendung von Hardwarebeschleunigung über die CPU (wie in Intel i5/i7 Prozessoren neu integriert) angeboten. Algorithmen, die als veraltet markiert sind, werden zwar von der Implementierung unterstützt, jedoch nicht bei der Erzeugung neuer Volumes angeboten.

Unter Linux können statt der eingebauten Kryptoimplementierungen auch die des Kernels verwendet werden (was u.a. einen weiteren Geschwindigkeitsvorteil bringen kann, wenn der Kernel Zugang zu Beschleunigungshardware hat). Es können jedoch ausschließlich Kryptoalgorithmen verwendet werden, die TrueCrypt zusätzlich auch selbst implementiert.

2.1.2 Ort notwendiger Änderungen

Unabhängig vom Betriebssystem müssen zum Hinzufügen weiterer Kryptoalgorithmen diese zunächst im *Crypto*-Subsystem implementiert werden.

Linux

Unter Linux muss dann im *Volume*-Subsystem folgende Änderungen vorgenommen werden:

- Die Cipherabstraktion in der `Cipher`-Klasse (unter Verwendung der neuen Implementierung im *Crypto*-Subsystem) muss implementiert werden.
- Der Kryptoalgorithmus muss in der Klasse `EncryptionAlgorithm` verankert werden. Hier kann auch festgelegt werden, in welchen Betriebsmodi er verwendbar ist, oder ob man ihn als Teil einer *Kaskade* nutzen möchte.
- In der `VolumeLayout`-Klasse muss der zuvor hinzugefügte `EncryptionAlgorithm` noch verfügbar gemacht werden.

Schließlich sollte man in Main/Forms/ die Forms anpassen, die spezifische

Informationen über den Kryptoalgorithmus zeigen sollen (z.B. `EncryptionOptionsWizardPage`, wo eine Kurzhilfe/Erklärung zum Algorithmus gezeigt wird).

Windows

Unter Windows sind hauptsächlich Änderungen im *Common*-Subsystem zu machen, wo die *Crypto*-Abstraktion für Windows zu finden ist. In `Crypto.(c|h)` müssen dort entsprechende Änderungen vorgenommen werden. Wenn der Kryptoalgorithmus auch für Pre-Boot Authentication verwendet werden soll, so muss dort auch die Klasse `BootEncryption` angepasst werden.

Wie unter Linux auch sollte der Code für bestimmte Dialogfenster angepasst werden. Diese finden sich u.a. in `Common/DlgCode.c` und `Format/Tcformat.c`.

2.1.3 Bewertung

Linux

Das Design für unixbasierte Systeme erlaubt eine sehr flexible Einbindung von neuen Kryptoalgorithmen. Es gilt zu beachten, dass eine Implementierung im Kernel nicht ausreichend ist, der entsprechende Algorithmus muss immer auch nativ im *Crypto*-Subsystem implementiert werden.

Für die Änderungen sind (ohne die tatsächliche Implementierung des Kryptoalgorithmus selbst) **2 Personentage** zu veranschlagen.

Windows

Unter Windows sind im Vergleich zu Linux etwas mehr Änderungen erforderlich (was vor allem der fehlenden Objektorientiertheit und den zusätzlichen Features wie Pre-Boot-Authentication geschuldet ist).

Für die Änderungen sind (ohne die tatsächliche Implementierung des Kryptoalgorithmus selbst) **3 Personentage** zu veranschlagen.

2.1.4 Implementierungsaufwand für einen AES-ähnlichen Krypto-Algorithmus

Um den Implementierungsaufwand für einen AES-ähnlichen, symmetrischen Krypto-Algorithmus abzuschätzen, müssen zunächst einige Vorbedingungen getroffen werden.

Angenommen sei:

- Sämtliche mathematischen Grundlagen sind dem Entwickler vertraut, z.B. der Umgang mit Galois Fields (sofern nötig).
- Der Algorithmus ist vollständig spezifiziert.
- Es wird nur eine fest spezifizierte Schlüssellänge betrachtet.
- Es liegen Test-Vektoren für alle notwendigen Runden vor, incl. der Input- / Output-Parameter von ggf. vorhandenen Teilblöcken. Beim AES wären dies z.B. Ergebnisse nach
  - der Schlüssel-Expansion
  - der Vorrunde
  - jeder Verschlüsselungsrunde
  - der Schlussrunde.
- Idealerweise liegen auch die Input- / Output-Parameter der einzelnen Funktionsaufrufe von Unterfunktionen pro Runde (z.B. Substitution, ShiftRow, MixColumn) vor.
- Der Algorithmus wird zunächst in der Programmiersprache C implementiert.
- Es wird der Electronic Code Book (ECB) Betriebsmodus gewählt, d.h. jeder Block wird für sich ver- und entschlüsselt ohne weitere Abhängigkeiten zu anderen Blöcken.
- Es werden keine Optimierungen hinsichtlich Ausführungszeit oder Codegröße vorgenommen.

Unter den genannten Voraussetzungen lassen sich folgende Entwicklungszeiten abschätzen:

- Lesen und Verstehen der Spezifikation: **1 Tag**
- Implementierung und Tests der mathematischen Operationen, z.B.GF: **2 Tage**
- AES-Ablauf und Testprogramm zur Steuerung des AES-Ablauf (ohne die eigentliche Implementierung der Funktionen): **1 Tag**
- Implementierung und Tests der Schlüssel-Verwaltung und -Expansion: **1 Tag**
- Implementierung der AES-Funktionen: **2 Tage**
- Generierung / Ableitung / Implementierung von S-Boxen: **1 Tag**
- Test der Implementierung, Unit-Tests, Dokumentation: **1 Tag**
- Entschlüsselung, d.h. Umdrehen der Ablaufreihenfolge sowie Inversenberechnung: **1 Tag**

Damit ergibt sich eine grob geschätzte Gesamtentwicklungszeit von ca. **10 Personentagen**.

Je nach Erfahrungen im Bereich der Implementierung, der Mathematik und der Kryptographie kann es deutlich kürzer oder länger sein.

Darüber hinaus lassen sich Optimierungen mit folgenden zusätzlichen Aufwänden abschätzen:

- Anderer Betriebsmodus, z.B. CBC oder XTS: + **1 Tag**
- Unterstützung für weitere Schlüssellängen (erfordert meist Änderungen am Rundenverhalten sowie am Schlüsselspeicher): + **1 Tag**
- Optimierung hinsichtlich des Ressourcen-Verbrauchs (z.B. bei Einsatz in Bootloader mit begrenztem Platz): **+ 10 Tage**

  Hier ist sicherlich sowohl der Einsatz von Assembler nötig als auch eine dynamische Generierung Werten anstelle von platzintensiven Lookup-Tables.
- Optimierung hinsichtlich der Laufzeit: **+ 10 Tage**

  Auch hier wird sicherlich zur Optimierung Assembler für bestimmte Operationen eingesetzt, um die CPU-Architektur bestmöglich zu nutzen. Ebenfalls können effiziente Implementierungstechniken wie „BitSlicing" implementiert werden, welche mehrere Blöcke zeitgleich verschlüsselt (in Abhängigkeit vom gewählten Betriebsmodus).
- Optimierung hinsichtlich der Sicherheit: **+ n Tage**

  Um eine sichere Implementierung zum Schutz vor Seitenkanalangriffen zu erzielen, ist es notwendig, die Informationen, welche über Seitenkanäle abgegriffen werden können, zu minimieren. In der Regel stehen an einer PC-Plattform meist nur Timing- und Cache-Angriffe zur Verfügung, wohingegen auf Smartcards z.B. auch der Stromverbrauch gemessen werden kann. Ziel einer sicheren Implementierung kann daher sein, die Ausführungsdauer sowie den Stromverbrauch pro Funktionsaufruf konstant, d.h. unabhängig vom verwendeten (Teil-)Schlüssel, zu machen.Je nach Zielplattform sind dazu zunächst Testframeworks, Testhardware etc. notwendig. Ebenfalls muss ein enormer Testaufwand betrieben werden, ggf. gepaart mit zusätzlichem Rechen- und Implementierungsaufwand zur Analyse der Seitenkanal-informationen. Daher kann hier keine Abschätzung hinsichtlich einer sicheren Implementierung getroffen werden.

2.2 *Hashalgorithmen*

2.2.1 Beschreibung

TrueCrypt implementiert unter allen Betriebssystemen folgende Hashalgorithmen:

- RIPEMD-160
- SHA-512
- Whirlpool
- SHA-1 (als veraltet/legacy markiert)

Algorithmen, die als veraltet markiert sind, werden zwar von der Implementierung unterstützt, jedoch nicht bei der Erzeugung neuer Volumes angeboten.

2.2.2 Ort notwendiger Änderungen

Genau wie bei den Kryptoalgorithmen zuvor, müssen zusätzliche Hashalgorithmen betriebssystemunabhängig zunächst im *Crypto*-Subsystem implementiert werden.

Linux

Unter Linux müssen zusätzlich folgende Änderungen im *Volume*-Subsystem vorgenommen werden:

- In der `Hash`-Klasse muss die entsprechende Hashabstraktion implementiert werden (unter Verwendung der Hashimplementierung aus dem *Crypto*-Subsystem).
- In der `Pkcs5Kdf`-Klasse muss mittels der Hashabstraktion die Schlüsselableitung implementiert werden.
- Die soeben implementierte Schlüsselableitung muss noch in der Klasse `VolumeLayout` verfügbar gemacht werden.

Windows

Unter Windows müssen mehrere Änderungen im Common-Subsystem durchgeführt werden:

- In *Common/Crypto.(c|h)* muss die Hashabstraktion implementiert werden (mittels der Implementierung im *Crypto*-Subsystem)
- In *Common/Pkcs5.(c|h)* muss die entsprechende Schlüsselableitung implementiert werden.
- In *Common/Volumes.(c|h)* und in *Common/EncryptionThreadPool.(c|h)* muss die eben implementierte Schlüsselableitung verankert werden.

- Im Zufallszahlengenerator (*Common/Random.(c|h)*) muss die entsprechende Hashabstraktion ebenfalls verankert werden.

2.2.3 Bewertung

Linux

Die erforderlichen Änderungen sind ähnlich zu denen für Kryptoalgorithmen. Durch das flexible Design sind wenig Änderungen erforderlich.

Für die Änderungen sind (ohne die tatsächliche Implementierung des Hashalgorithmus selbst) **2 Personentage** zu veranschlagen.

Windows

Wie bei den Kryptoalgorithmen auch, sind in der Windowsimplementierung mehr Änderungen erforderlich, wiederum bedingt durch fehlende Objektorientierung bzw. die Verwendung von C als Sprache.

Für die Änderungen sind (ohne die tatsächliche Implementierung des Hashalgorithmus selbst) **3 Personentage** zu veranschlagen.

2.2.4 Hashalgorithmus der Full-Disk-Encryption austauschen (Windows)

Für die Full-Disk Encryption mit Pre-Boot Authentication ist keinerlei Auswahl des zu verwendenden Hashalgorithmus möglich, es wird immer RIPEMD-160 verwendet. Beschrieben werden hier die notwendigen Schritte, um diesen Algorithmus durch einen anderen zu ersetzen.

Ort notwendiger Änderungen

In *Common/Volumes.c* existiert eine eigene Version der Funktion `ReadVolumeHeader` für die Verwendung mit Pre-Boot-Authentication. Hierbei ist in Zeile 600 die Funktion `derive_key_ripemd160` fest einprogrammiert und muss lediglich ausgetauscht werden.

Um auch bei der Erzeugung eines Volumes mit Pre-Boot-Authentication den richtigen Hashalgorithmus zu verwenden, muss weiterhin in *Common/Crypto.h* die Definition von `DEFAULT_HASH_ALGORITHM_BOOT` entsprechend geändert werden.

Bewertung

Die notwendigen Änderungen erscheinen nur geringfügig und sollten innerhalb von **2 Personentagen** zu erledigen sein.

2.3 Betriebsarten für die Kryptoalgorithmen

2.3.1 Beschreibung

TrueCrypt implementiert unter allen Betriebssystemen folgende Betriebsarten für Kryptoalgorithmen:

- XTS
- LRW (nur für Legacysupport)
- CBC (nur für Legacysupport)

Grundsätzlich verwendet TrueCrypt den XTS-Modus und greift auf einen der restlichen Modi nur zu, wenn ein altes Volume mit diesem Modus erstellt wurde. Betrachtet werden soll die Implementierung eines zusätzlichen Betriebsmodus mit optionalem Zähler für die Anzahl der bereits geschriebenen Daten seit dem letzten Schlüsseltausch. Es wird hierbei davon ausgegangen, dass das gesamte Volume mit einem Schlüssel verschlüsselt werden kann (bei XTS als Betriebsmodus bis zu 1 TB nach NIST) und der Schlüssel geändert werden muss, sobald die vorgeschriebene Grenze an geschriebenen Daten erreicht wurde (Dann müssen alle Daten neuverschlüsselt werden, bevor das Volume wieder beschrieben werden kann). Die Neuverschlüsselung (d.h. Austausch des Schlüsselmaterials) wird hier explizit nicht betrachtet, da dies unter *Austausch des Schlüsselmaterials* bereits betrachtet wird.

2.3.2 Ort notwendiger Änderungen

Linux

Unter Linux müssen zusätzlich folgende Änderungen im *Volume*-Subsystem vorgenommen werden:

- Der Betriebsmodus muss als Ableitung der `EncryptionMode`-Klasse implementiert werden und dann in `EncryptionMode` selbst verankert werden.
- Jeder Kryptoalgorithmus bzw. jede Kaskade, der/die den Betriebsmodus nutzen soll, muss diesen als unterstützten Betriebsmodus in `EncryptionAlgorithm` angeben.
- In der Klasse `VolumeLayout` muss der Betriebsmodus für die gewünschten Layouts verfügbar gemacht werden.
- Die Klasse `VolumeHeader` muss so angepasst werden, dass statt XTS der neue Betriebsmodus genutzt wird. Je nach Art des Betriebsmodus (Schlüsselgröße) muss dies im Header berücksichtigt werden.
- In der Klasse `Volume` muss die Methode `WriteSectors` angepasst werden, um einen Datenzähler im Header zu verwenden und ggf. weitere Schreibzugriffe zu verhindern. Hier kann ein Teil des reservierten Speichers im

Header für den Zähler verwendet werden.

Um den Betriebsmodus als Standard beim Erstellen eines Volumes zu verwenden, muss zusätzlich im *Core*-Subsystem der `VolumeCreator` so angepasst werden, dass er statt XTS den neuen Modus verwendet.

Soll der Linux Device Mapper verwendet werden, müssen zusätzliche Änderungen im *Core*-Subsystem vorgenommen werden:

- In der Klasse `CoreLinux` muss die Funktion *MountVolumeNative* entsprechend angepasst werden, so dass sie die nötigen Daten und Parameter an den Device Mapper übergibt.

Schließlich müssen im *Main*-Subsystem einige Forms und Anzeigen angepasst werden.

Windows

Unter Windows müssen mehrere Änderungen im *Common*-Subsystem durchgeführt werden:

- Der Betriebsmodus muss analog zu *Common/xts.(c|h)* implementiert werden.
- In *Common/Crypto.(c|h)* muss der Betriebsmodus umfangreich verankert werden. Soll der Betriebsmodus als Standardmodus verwendet werden und/oder für Systemverschlüsselung, so sind zusätzliche Änderungen in dieser Komponente nötig.
- In *Common/Volumes.(c|h)* muss der Betriebsmodus bei der Volumeabstraktion berücksichtigt werden (Header).
- In *Common/Fat.(c|h)* muss eine Änderung durchgeführt werden, falls der implementierte Betriebsmodus eine andere Schlüsselanzahl nutzt, als die bisherigen.
- In *Driver/EncryptedIoQueue.c* muss die Methode `MainThreadProc` angepasst werden, um einen Datenzähler im Header zu verwenden und ggf. weitere Schreibzugriffe zu verhindern. Hier kann wie unter Linux ein Teil des reservierten Speichers im Header für den Zähler verwendet werden.

Zusätzlich müssen in den *Mount-* und *Format*-Subsystemen geringfügige Anpassungen durchgeführt werden, um den neuen Betriebsmodus verfügbar zu machen.

2.3.3 Bewertung

XTS unterscheidet sich durch seine zwei Schlüssel von den übrigen Betriebsmodi (LRW, CBC) die nur einen Schlüssel einsetzen. Dieser Umstand erfordert an einigen Stellen eine Sonderbehandlung für XTS, was unter beiden Betriebssystemen einen Mehraufwand bedeutet.

Linux

Die erforderlichen Änderungen unter Linux sind zunächst ähnlich zu den Änderungen, welche für Krypto- und Hashalgorithmen erforderlich sind. Dennoch sind zusätzliche Änderungen erforderlich, da das Design nicht konsequent umgesetzt ist und XTS als Modus an einigen Stellen direkt verankert ist.

Für die Änderungen sind **8 Personentage** zu veranschlagen.

Wird ein zusätzlicher Datenzähler gewünscht, so erhöht sich der Arbeitsaufwand um weitere **4 Personentage**.

Windows

Wie bei den Krypto- und Hashalgorithmen sind in der Windowsimplementierung mehr Änderungen erforderlich, wiederum aus den bereits zuvor genannten Gründen.

Für die Änderungen sind **12 Personentage** zu veranschlagen.

Wird ein zusätzlicher Datenzähler gewünscht, so erhöht sich der Arbeitsaufwand um weitere **8 Personentage**.

2.4 Zusätzliche Quellen für Zufallszahlengenerator

2.4.1 Beschreibung

Die hier beschriebenen Änderungen zeigen, wie sich unter den verschiedenen Betriebssystemen die Zufallsgeneratoren so verändern lassen, dass sie weitere externe Zufallsquellen zur Generierung hinzuziehen.

2.4.2 Ort notwendiger Änderungen

Linux

Hier muss im *Core*-Subsystem die `RandomNumberGenerator`-Klasse angepasst werden. Es empfiehlt sich, die Funktion `AddSystemDataToPool` zu erweitern. Diese Funktion ist normalerweise dafür verantwortlich, Zufall aus */dev/random* und */dev/urandom* zu gewinnen.

Windows

Hier müssen im *Common*-Subsystem die Dateien *Random.(c|h)* erweitert werden. Ähnlich zum RNG in Linux hat dieser Zufallszahlengenerator Zugriff auf schnelle und langsame Zufallsquellen, diese sind aber in verschiedenen Funktionen untergebracht. Zu betrachten/erweitern sind hier die Funktionen `SlowPoll` und `FastPoll`.

2.4.3 Bewertung

Die Zufallszahlengeneratoren sind auf den beiden betrachteten Betriebssystemen ähnlich aufgebaut und in beiden Fällen einfach erweiterbar. Nimmt man die eigentliche Implementierung der zusätzlichen Quellen aus, so sollte unter beiden Systemen eine Einbindung höchstens **1 Personentag** in Anspruch nehmen.

Zu beachten ist die Tatsache, dass beide Zufallszahlengeneratoren zwischen schnellen und langsamen Quellen unterscheiden, und daher unterschiedlich Gebrauch von diesen machen. Um Performanceprobleme zu vermeiden sollte man sich vergewissern, die Quelle an der richtigen Stelle einzubauen.

2.5 Verbot dateibasierter Volumes

2.5.1 Beschreibung

Beschrieben werden sollen Änderungen, die die Nutzung von dateibasierten Volumes mit TrueCrypt gänzlich verhindern. Voraussetzung hierfür ist, dass der Nutzer keinerlei administrative Rechte hat.

2.5.2 Ort notwendiger Änderungen

Beschrieben werden hier nur die Änderungen, die erforderlich sind, um die Nutzung solcher Volumes unmöglich zu machen. Weitere kosmetische Änderungen (z.B. GUI) sind hier nicht beschrieben.

Linux

Hier müssen im *Core*-Subsystem in den Klassen `CoreUnix` und `CoreLinux` die Methoden `MountVolume` und `MountVolumeNative` so angepasst werden, dass sie keine Dateien als Volumes akzeptieren. Für Dateien gibt es in diesen beiden Methoden bereits eine gesonderte Behandlung, da hier im Falle einer Datei ein *Linux Loop Device* aufgesetzt werden muss.

Windows

Unter Windows muss die entsprechende Änderung im Kerneltreiber erfolgen. Hierfür kann in *Driver/Ntdriver.c* in der Methode `ProcessMainDeviceControlIrp` die Behandlung des IoControl-Befehls `TC_IOCTL_MOUNT_VOLUME` verändert werden, so dass für Dateien der Befehl fehlschlägt.

2.5.3 Bewertung

Unter beiden Betriebssystemen sind unter den genannten Voraussetzungen nur geringfügige Änderungen erforderlich, um die genannte Funktionalität zu entfernen. Für die Änderungen sind pro Betriebssystem mit **2 Personentagen** zu rechnen.

2.6 Integritätsprüfung

In diesem Kapitel sollen Möglichkeiten aufgezeigt werden, wie die TrueCrypt-Implementierung Veränderungen, die am geschlossenen Volume durchgeführt wurden, erkennen kann. Hierzu gibt es zwei verschiedene Ansatzmöglichkeiten:

2.6.1 Verwendung eines Dateisystems mit Integritätsprüfung

Innerhalb eines Volumes kann zum Integritätsschutz ein Dateisystem- oder ein dateibasierter Mechanismus eingesetzt werden, der die Integrität sicherstellt. Zumindest unter Linux gibt es hierfür bereits einige Ansätze, z.B.

- ZFS – Ein Dateisystem von Sun, das auch Integritätsprüfung mittels Prüfsummen unterstützt.
- i3FS – Ein stapelbares Dateisystem für Linux, das zu einem bisherigen Dateisystem (z.B. ext3) Integritätsprüfung hinzufügt
- EncFS Ein FUSE-basiertes verschlüsseltes Dateisystem, das MAC für Integritätsprüfung benutzt
- TripWire – Ein dateibasiertes Werkzeug um Änderungen an bestimmten Dateien oder Dateigruppen zu entdecken (für Windows und Linux)

Vorteil: Keine funktionale Änderung in TrueCrypt selbst erforderlich.

Nachteil: Aufwand der Implementierung eines passenden Mechanismus insbesondere unter Windows schwierig abschätzbar.

2.6.2 Kryptografische Hashes auf Blockebene

Will man den Integritätsschutz direkt in TrueCrypt implementieren, so wäre ein möglicher Ansatz das Bilden von kryptografischen Hashes über Blöcke oder Blockgruppen. Hierfür würde das eigentliche Dateisystem kleiner angelegt werden, als das Volume, so dass der restliche Platz ausreichend ist, um die Integritätsdaten zu speichern. Beim erstmaligen Lesen eines Blocks muss dann die Prüfsumme berechnet und verglichen, bei jedem Schreibzugriff neu geschrieben werden. Dies bedeutet vor allem in Anwendungsbereichen mit vielen Schreibzugriffen eine drastische Verringerung der Performance. Zusätzlich muss auf die Konsistenz nach einem Ausfall geachtet werden.

Falls kein kryptografischer Hash gewünscht ist, kann stattdessen auch eine andere Form der Prüfsumme, z.B. CRC eingesetzt werden, um die Performance zu erhöhen.

Vorteil: Direkte Implementierung in TrueCrypt, betriebssystemunabhängig.

Nachteil: Starker Performanceverlust, falls kryptografische Hashes eingesetzt werden sollen.

Ort notwendiger Änderungen (Windows)

Unter Windows würde man diese Änderung im Gerätetreiber im *Driver*-Subsystem implementieren, da hier auch sämtliche I/O Requests bearbeitet werden.

Ort notwendiger Änderungen (Linux)

Die nötigen Änderungen sind abhängig von der Verwendungsweise (FUSE oder Kernel Device Mapper). Für die weitere Betrachtung wird angenommen, dass lediglich das FUSE-Dateisystem eingesetzt werden soll, da für die Anpassung des Device-Mappers Änderungen außerhalb von TrueCrypt (nämlich im Kernel selbst) erforderlich wären.

Mit FUSE würden die notwendigen Änderungen in *Driver/Fuse* gemacht werden, wo alle Lese- und Schreibzugriffe zusammenlaufen.

Bewertung

Sowohl unter Linux als auch unter Windows erscheint die Änderung machbar, jedoch ist der Performanceverlust stark abhängig vom Verwendungsszenario.

Für die Implementierung ist pro Betriebssystem mit **mindestens 14 Personentagen** zu rechnen.

2.6.3 RAID-artige Redundanz innerhalb eines Volumes

Eine weitere Möglichkeit, Integritätsschutz zu erreichen ist die Verwendung von bestimmten Schemata wie sie für RAID Systeme bekannt sind, beispielsweise RAID 1 oder RAID 5. Im einfachsten Szenario (RAID 1) würde ein Volume alle Daten doppelt vorhalten (mit unterschiedlichen Schlüsseln), und bei jedem Lesevorgang die Daten aus beiden Quellen lesen und vergleichen. Beim Schreiben würden immer beide Datensätze aktualisiert werden. Der Platzverbrauch kann weiter reduziert werden, indem wie bei RAID 5 ein XOR Schema eingesetzt wird, so dass 66% des gesamten Platzes als Nutzdaten zur Verfügung stehen, und 33% für Integritätsdaten genutzt werden. Voraussetzung für die Sicherheit eines solchen Ansatzes ist ein robuster Betriebsmodus für die verwendeten kryptografischen Algorithmen. Da hier keine Hashes zum Einsatz kommen, sind Angriffe eher denkbar als im vorhergehenden Kapitel, wenn der Betriebsmodus (semi-)sinnvolle Änderungen am Plaintext durch den Ciphertext ermöglicht. Generell sollte der Betriebsmodus jedoch sowieso so gewählt werden, dass dies nicht möglich ist (z.B. XTS).

Vorteil: Direkte Implementierung in TrueCrypt, betriebssystemunabhängig.
Keine kostspielige Hash-/Prüfsummenberechnung (Performance)

Nachteil: Größere Einbußen bei der Nutzkapazität des Datenträgers.

Ort notwendiger Änderungen

Die notwendigen Änderungen sind ähnlich zu den Änderungen im Kapitel Kryptografische Hashes auf Blockebene und würden auch in den gleichen Subsystemen implementiert werden.

Bewertung

Da die gewünschte Funktionalität mit zwei Volumes implementiert werden kann, ist weniger Aufwand zu erwarten, als bei der vorher genannten Methode. Dennoch sind pro Betriebssystem wohl **mindestens 12 Personentage** zu veranschlagen.

Alternativ könnte man auch versuchen, die gewünschte RAID Funktionalität gar nicht in TrueCrypt selbst zu implementieren, sondern in durch einen Betriebssystemmechanismus zu erreichen. Beispielsweise kann man unter Linux durch Loop Devices das geöffnete Volume wiederum in 3 Devices unterteilen und den Linux Kernel Device Mapper dazu nutzen, RAID 5 auf diesen Devices zu betreiben. Hierdurch sind nur unwesentliche Änderungen in TrueCrypt selbst notwendig (ca. **3-4 Personentage**). Unter Windows wäre hier dennoch vermutlich ein zusätzlicher Gerätetreiber erforderlich.

2.7 Key-Management

Die in diesem Kapitel beschriebenen Remoteverfahren beschreiben jeweils nur die Möglichkeiten, diese auf der Clientseite zu integrieren. Mögliche vollständige Infrastrukturen (Server, sichere Übertragung, etc.) werden in AP 10 betrachtet.

2.7.1 Verfahren zum entfernten Aufsetzen von FDE oder Container mit Escrow

Beschreibung

Es soll ermöglicht werden, eine Full-Disk Encryption auf einem bestehenden System zu aktivieren. Dabei soll jedoch der verwendete Schlüssel auch in einem zentralen System registriert werden um z.B. bei Verlust des Schlüssels auf das System zugreifen zu können. Die gleiche Funktionalität soll auch für das Erstellen einzelner Container betrachtet werden.

Voraussetzungen

Voraussetzung ist, dass das System frei von schädlicher Software ist. Weiterhin muss der Prozess von einem Benutzer ausgeführt werden, der administrative Rechte erlangen kann.

Für die Übertragung muss eine sichere Verbindung zum zentralen System vorausgesetzt werden, die eine beidseitige Authentifizierung durchführt sowie den Kanal verschlüsselt.

Ort notwendiger Änderungen (FDE)

Die Komponente, die die FDE startet, muss entsprechend angepasst werden, sodass die externe Verbindung aufgebaut wird und das Schlüsselmaterial übermittelt werden kann, bevor der Prozess gestartet wird.

Unter *Windows* wäre diese Änderung in der Format-Anwendung durchzuführen. Unter *Linux* ist dies davon abhängig, wie die FDE implementiert wird.

Ort notwendiger Änderungen (Container)

In beiden System bietet es sich auf Grund des Designs an, das Volume zuerst zu erzeugen und dann nachträglich eine Verbindung zum zentralen System aufzubauen und das Volume dort „anzumelden". Nach Erzeugung eines Volumes würde die GUI dann entsprechend einen weiteren Dialog anzeigen, der die Anmeldung ermöglicht.

Unter Windows sind diese Änderungen wie auch für die FDE in der Format-Anwendung durchzuführen.

Unter Linux müsste man hier die GUI im Main-Subsystem entsprechend anpassen.

Bewertung (FDE)

Unter Windows beschränken sich die Änderungen auf eine Komponente,der Aufwand ist jedoch auch abhängig davon, wie die externe Anbindung zustande

kommt. Für die Integration kann grob mit einem Aufwand von ca. **10 Personentagen** gerechnet werden (sofern die Voraussetzungen erfüllt sind und eine sichere Verbindung bereits existiert).

Für Linux gilt das gleiche wie für Windows, mit dem Zusatz, dass hier noch keine FDE Implementierung existiert, was eine sinnvolle Abschätzung unmöglich macht.

Bewertung (Container)

Die zusätzlichen GUI Änderungen sind minimal, ansonsten erfolgt die Anbindung auf die gleiche Weise wie bei FDE. Kann auf die FDE Implementierung zurückgegriffen werden, so ist mit einem Aufwand von **4 Personentagen** pro Betriebssystem zu rechnen.

2.7.2 Verfahren bei Verlust des Schlüssels für einen Container

Beschreibung

Hier soll beschrieben werden, wie z.B. bei verlorenem Schlüssel ein neuer Schlüssel sicher bereitgestellt werden kann, falls es sich um einen Container handelt.

Voraussetzungen

Es gelten die gleichen Voraussetzungen wie in 2.7.1.

Ort notwendiger Änderungen

Die entsprechende Funktion kann sowohl von einer externen Anwendung, als auch durch die TrueCrypt Funktion „Backup wiederherstellen" ermöglicht werden. Im letzteren Fall würde man die entsprechende Funktion RestoreVolumeHeader so anpassen, dass sie das Backup direkt durch die externe sichere Verbindung beziehen kann.

Bewertung

Die Änderungen sind ähnlich zu denen in 2.7.1. Der Aufwand der Integration kann mit ca. **8 Personentagen** pro Betriebssystem veranschlagt werden (sofern die Voraussetzungen erfüllt sind und eine sichere Verbindung bereits existiert).

2.7.3 Verfahren bei Verlust des Schlüssels für FDE

Beschreibung

Wie in 2.7.2 geht es hier um den Verlust des Schlüssels, bzw. die Bereitstellung eines neuen Schlüssels, allerdings unter der Voraussetzung, dass eine Full-Disk Encryption aktiv ist. Die zusätzliche Schwierigkeit ergibt sich hier nun daraus, dass keinerlei Zugriff mehr auf das Betriebssystem möglich ist.

Voraussetzungen

Da ein Onlineverfahren hier nicht möglich ist, ist anderem Wege sicherzustellen, dass die Wiederherstellungsinformationen sicher übertragen werden (z.B. per Kurier).

Ort notwendiger Änderungen

Änderungen an der Software selbst sind nicht erforderlich. TrueCrypt bietet die Möglichkeit, von einem Rettungsmedium (z.B. CD) zu booten und den Header der Full-Disk Encryption auszutauschen. Ein neuer Header muss mittels des zentralen Systems erstellt und auf ein Rettungsmedium übertragen werden. Dieses Medium sowie das entsprechende Passwort müssen sicher zugestellt werden.

Bewertung

An TrueCrypt selbst sind keinerlei Änderungen erforderlich.

2.7.4 Austausch des Schlüsselmaterials

Mittels der Funktion „Passwort ändern" werden lediglich die Header eines Volumes neuverschlüsselt. Beschrieben werden sollen hier Änderungen, die nötig sind, um das gesamte Volume neuzuverschlüsseln.

Ort notwendiger Änderungen

Unter Windows entspricht die gewünschte Funktionalität in etwa der Komponente `Inplace` im *Format*-Subsystem. Diese wird verwendet, um ein bestehendes Dateisystem zu verschlüsseln. Zu implementieren wäre eine ähnliche Komponente, die nicht nur ver- sondern umschlüsselt. Eine Dateisystemverkleinerung wie bei `Inplace` ist hier nicht erforderlich, da das Volume bereits verschlüsselt ist. Die Funktionalität muss dann in *Format/Tcformat.c* hinzugefügt werden.

Unter Linux existiert eine ähnliche Funktionalität noch nicht, kann aber an das Vorgehen unter Windows angelehnt werden. Hierzu bietet es sich an, die Klasse `Volume` im *Volume*-Subsystem um Funktionen für die Umschlüsselung zu erweitern (hier liegt auch die Funktion zum Umschlüsseln des Headers). Für die Umschlüsselung verwendet man den *Backupheader*, bis die Operation vollständig abgeschlossen ist. Dadurch ist sichergestellt, dass man die Operation mit der notwendigen Implementierung auch wieder aufnehmen kann (wie unter Windows). Die Integration erfolgt dann im *Core/Unix*- und im *Main*-Subsystem.

Bewertung

Unter Windows ist zwar eine ähnliche Implementierung bereits vorhanden, dennoch wird eine Erweiterung durch die fehlende Objektorientierung eingeschränkt, so dass **12 Personentage** zu veranschlagen sind.

Unter Linux liegt keine ähnliche Implementierung vor, kann aber in das bestehende Design leicht ergänzt werden. Auch hier wären **12 Personentage** zu veranschlagen.

2.8 Security Token

2.8.1 SmartCard beim Bootvorgang

TrueCrypt unterstützt die Verwendung von SmartCards erst, nachdem das Betriebssystem geladen wurde (also nicht mit Pre-Boot-Authentication), da auf eine externe PKCS-#11-Library zugegriffen werden muss. Beschrieben werden soll die Erweiterung der Anbindung von Security Tokens auf die Pre-Boot-Authentication.

Ort notwendiger Änderungen

Da die entsprechenden Änderungen den Bootloader betreffen, muss die Implementierung im *Boot/Windows*-Subsystem erfolgen. Zu beachten ist, dass es eine Größenbeschränkung für die resultierende Implementierung gibt, da der gesamte Bootloader nicht mehr als 32 kB groß sein darf (sofern nicht noch an anderer Stelle auf der Platte Code im Klartext abgelegt werden soll).

Der bisherige Bootloader enthält:

- 512 Byte MBR
- 2 kB Decompressor
- ca. 11 kB komprimierter Code
- 3 Byte Prüfsummen
- Backup von Decompressor und komprimiertem Code

Entfernt man das Backup, so erhält man insgesamt ca. 18 kB freien Speicher für die nötige Implementierung. Da die Implementierung dann Teil des komprimierten Codes ist, steht sogar noch etwas mehr Speicher hierfür zur Verfügung.

Bewertung

Die Implementierung um mit dem jeweiligen Token zu kommunizieren muss für diesen Zweck explizit programmiert werden. Der Aufwand und ob eine Implementierung in der nötigen Größe überhaupt möglich ist, ist hier stark abhängig von der Komplexität des Geräts.

Sofern eine entsprechende Implementierung vorhanden ist, kann sie ohne größeren Aufwand integriert werden. Treten keinerlei unerwartete Probleme bei der Integration auf, so dürfte diese mit **6 Personentagen** zu bewältigen sein.

2.8.2 Anbindung nicht-PKCS#11 kompatibler Geräte

Beschrieben werden sollen notwendige Änderungen an TrueCrypt zum Anbinden von Security Token, die beispielsweise vom bereits unterstützten PKCS #11 Standard abweichen.

Ort notwendiger Änderungen

Die `SecurityToken` Komponente von TrueCrypt im *Common* Interface ist eine der wenigen C++ Komponenten die sowohl unter Linux als auch unter Windows gemeinsam verwendet werden. Diese Komponente abstrahiert den Zugriff auf die externe PKCS #11 Library und kann ggf. abgeleitet oder angepasst werden, um Zugriff auf andere Interfaces/Libraries zu ermöglichen. Voraussetzung hierfür ist, dass das Token einen Datenblob zur Verfügung stellen kann, der über die Keyfile-Funktionalität als Schlüsselmaterial nutzbar ist.

Bewertung

Unter der Voraussetzung, dass bereits eine Implementierung zur Kommunikation mit dem zu verwendenden Token existiert, ist eine Integration in TrueCrypt vermutlich unproblematisch. Da eine Abstraktionsebene für alle Betriebssysteme existiert, muss die Implementierung lediglich an diese Abstraktion angepasst werden. Hierfür sind ca. **6 Personentage** zu veranschlagen.

2.9 Authentisierung

Zusätzliche Authentisierungsmethoden sollten wenn möglich extern implementiert werden und die bereits vorhandenen Mechanismen (z.B. Keyfile) verwenden. Die notwendigen Änderungen zur Anbindung zusätzlicher Hardware wurde bereits im Kapitel Security Token beschrieben.

2.10 Multirollen/ -user Betrieb

2.10.1 Beschreibung

Wie in 1.3.1 beschrieben, enthält jedes TrueCrypt Volume einen Header, der mit einem Schlüssel verschlüsselt wird, welcher vom Passwort des Nutzers abgeleitet wird. Die Verwendung mehrerer Passwörter ist nativ nicht vorgesehen.

Um dieses Problem zu beheben und so mehreren Nutzern Zugang zu einem Volume zu geben, existieren mehrere Lösungsansätze, von denen zwei hier ausgeführt werden.

Verwendung verschiedener Headerkopien

TrueCrypt bietet zu Backup- und Verwaltungszwecken eine Funktion zum Anfertigen von Kopien des verschlüsselten Headers. Diese Funktion kann kombiniert mit der Funktion „Passwort ändern" dazu genutzt werden, verschiedene Headerkopien anzulegen, die mit verschiedenen Schlüsseln geöffnet werden können, aber dennoch alle in der Lage sind, das TrueCrypt-Volume zu entschlüsseln. Mittels eines externen Verwaltungsprogramms könnte dieser Prozess auch halbautomatisch gestaltet und an andere Zugangskontrollen geknüpft werden. Ein Benutzer, der auf sein TrueCrypt-Volume zugreifen möchte, müsste vorher lediglich seine eigene Headerkopie laden und dann das Volume ganz normal öffnen.

Vorteil: Die TrueCrypt Implementierung muss nicht zwingend geändert werden.

Nachteil: Erfordert externes Management der verschiedenen Header

Verwendung eines anderen Headerformats

Bei diesem Ansatz würde der bisherige TrueCrypt Header erweitert werden, so dass er zudem eine fixe Anzahl von sogenannten Key Slots enthält. Jeder dieser Key Slots würde, sofern er in Benutzung ist, den Schlüssel enthalten, der für den restlichen Header gebraucht wird. Verschlüsselt wären diese Key Slots mit den jeweils persönlichen Passwörtern der Benutzer. Dieses Design ist angelehnt an *Linux Unified Key Setup (LUKS)* und wird in Bild 11 veranschaulicht:

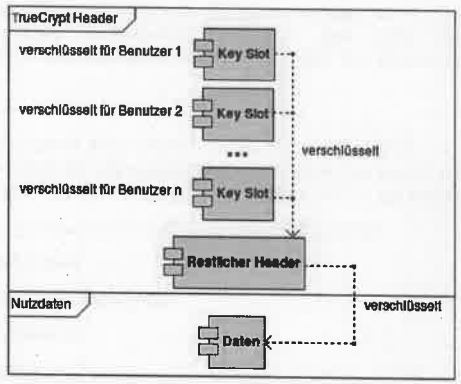

Bild 11: Key Management mittels Key Slots im Header

Will man mit diesem Headerformat Escrow erreichen, so kann ein Key Slot für diesen Zweck reserviert und mit einem zentral hinterlegten Passwort verschlüsselt werden. Das Einrichten kann dann ähnlich zu den in Kapitel 2.7 aufgezeigten Verfahren erfolgen.

Vorteil: Zugriffsmanagement im TrueCrypt-Volume selbst, keine externe Verwaltung nötig.

Nachteil: Erfordert komplexe Änderungen in TrueCrypt, die Inkompatibilität zu normalen TrueCrypt-Volumes erzeugt.

2.10.2 Ort notwendiger Änderungen

Für den ersten Vorschlag sind keine zwingenden Änderungen in TrueCrypt selbst erforderlich. Der Arbeitsaufwand ist abhängig davon, welche Form des externen

Managements gewünscht wird. Im Folgenden werden die nötigen Änderungen für eine Headeranpassung beschrieben.

Linux

Im *Volume*-Subsystem müssen folgende Änderungen vorgenommen werden:

- Der neue Header kann als Ableitung der Klasse `VolumeHeader` implementiert werden. Bestimmte Funktionen müssen überladen werden, um die neue Funktionalität zu unterstützen.
- Mittels der Klasse `VolumeLayout` muss eine Klasse für das neue Header Layout geschrieben werden (definiert Größen, Offsets und unterstützte Kryptoparameter usw.). Weiterhin muss dieses Layout die eben implementierte Ableitung von `VolumeHeader` zurückliefern.

Weiterhin sind Änderungen im *Core*-Subsystem notwendig:

- Es müssen in `CoreBase` Funktionen implementiert werden, die das ändern einzelner Keyslots unterstützen.
- Die Klasse `VolumeCreator` muss angepasst werden, um zusätzlich die Keyslots anzulegen.

Windows

Es sind massive Änderungen im Common-Subsystem, sowie in Mount und Format erforderlich.

Zusätzliche Änderungen für Full Disk Encryption

Da die FDE im Bootloader eine eigene Implementierung aus dem Windows Code zum Öffnen des Volumes benutzt, muss dieser so angepasst werden, dass der neue Headeraufbau berücksichtigt wird. Hierzu sind folgende Änderungen notwendig:

- Die BootMain-Komponente im Boot-Subsystem muss u.U. einen größeren verschlüsselten Header liefern und an die Volumes Komponente weitergeben
- Die ReadVolumeHeader Methode der Volumes Komponente im Common-Subsystem muss so umgeschrieben werden, dass sie das neue Headerformat kennt und jeden einzelnen Keyslot testet.

2.10.3 Bewertung

Linux

Die Änderungen sind sehr viel komplexer, als die bisher beschriebenen Erweiterungen, aber dank des objektorientierten Designs noch als überschaubar anzusehen.

Für die Änderungen sind **16 Personentage** zu veranschlagen.

Windows

Durch die fehlende Objektorientierung ist der Aufwand für die vorgeschlagene Änderung schwer bis überhaupt nicht abschätzbar.

Zusätzliche Änderungen für Full-Disk-Encryption

Durch die Anpassung des Bootloaders ist mit einem zusätzlichen Arbeitsaufwand von ca. **12 Personentagen** zu rechnen. Eine genaue Abschätzung ist jedoch nur schwer möglich, da hier hauptsächlich Windowskomponenten involviert sind, für die aus den eben genannten Gründen der Aufwand und mögliche Komplikationen schwer abschätzbar sind.

2.11 Trennen von Header und Volume

2.11.1 Beschreibung

Beschrieben werden sollen hier die notwendigen Änderungen um Volume und Header voneinander zu trennen. Dies muss sowohl während der Erzeugung als auch beim Öffnen des Volumes betrachtet werden. Full-Disk Encryption wird hier nicht betrachtet, da es zum Bootzeitpunkt nicht möglich ist, externe Headerfiles zu spezifizieren (eine solche Anforderung wäre ggf. mit Security Token o.ä. zu verknüpfen und getrennt zu betrachten).

2.11.2 Ort notwendiger Änderungen

Linux

Unter Linux sind Änderungen in den folgenden Bereichen erforderlich:

- In der Klasse `Volume` müssen die Methoden `Open` und `ReEncryptVolumeHeader` angepasst werden, so dass ein externes Headerfile übergeben und verwendet werden kann.
- In der Klasse `CoreBase` muss die Methode `OpenVolume` angepasst werden um ein externes Headerfile zu akzeptieren.
- In der Klasse `CoreUnix` muss die Methode `MountVolume` angepasst werden um ein externes Headerfile zu verwenden.
- Im *Core*-Subsystem müssen die Definitionen von `MountOptions` sowie `CreateVolumeOptions` um ein externes Headerfile erweitert werden.
- Im *Main*-Subsystem müssen die entsprechenden Änderungen in der Bedienoberfläche durchgeführt werden.

Windows

Unter Windows sind Änderungen in den folgenden Bereichen erforderlich:

- Im Gerätetreiber muss in *Ntdriver.c* muss der `TC_IOCTL_MOUNT_VOLUME` Request bzw. die darin verwendete Struktur für die Mountabfrage um ein Headerfile erweitert werden. In *Ntvol.c* muss in der Methode `TCOpenVolume` das Headerfile dann gelesen und verwendet werden.
- Im *Common*-Subsystem in *Dlgcode.c* muss die Methode `MountVolume` erweitert werden, so dass ein Headerfile verwendet werden kann.
- Die Anwendung *Mount* muss angepasst werden um Headerfiles beim Öffnen übergeben zu können.
- Die Anwendung *Format* muss angepasst werden um das Headerfile beim Erzeugen separat zu speichern.

2.11.3 Bewertung

Linux

Die Änderungen sind dank des objektorientierten Designs überschaubar.

Für die Änderungen sind ca. **8 Personentage** zu veranschlagen.

Windows

Sowohl durch die fehlende Objektorientierung als auch durch den Gerätetreiber sind die Änderungen unter Windows deutlich umfangreicher.

Hier sind für die Änderungen ca. **16 Personentage** zu veranschlagen.

2.12 *Sicheres Löschen von Volumes*

2.12.1 Beschreibung

Wie bereits in Sicheres Löschen/Überschreiben beschrieben, implementiert TrueCrypt zumindest unter Windows bereits Algorithmen, die für sicheres Löschen geeignet sind. Eingesetzt werden diese bisher nur bei der Verschlüsselung bereits bestehender Datenträger. Bei den eingesetzten Algorithmen handelt es sich um:

- DoD 5220.22-M (E)
- DoD 5220.22-M (ECE)
- Gutmann-Methode
- Einfaches Überschreiben mit Zufallszahlen

Es gibt zwei Ansätze, das Programm so zu erweitern, dass ein Volume sicher gelöscht werden kann. Zum einen kann dem Programm eine Funktionalität hinzugefügt werden, die einfach die **gesamten Daten** überschreibt. Der Nachteil dieser Methode ist der große zeitliche Aufwand bei der Vernichtung größerer Datenmengen. Alternativ kann auch **nur der Header** (und seine Sicherheitskopie auf der Platte) überschrieben werden. Der Nachteil dieser Methode ist, dass es noch

externe Kopien des Headers geben könnte. Mittels einer solchen Kopie könnten die Daten dann wiederhergestellt werden.

2.12.2 Ort notwendiger Änderungen

Linux

Unter Linux würde man im Main-Subsystem die GUI bzw. das textuelle Interface um die neue Funktionalität erweitern und diese entweder auch in TextUserInterface implementieren oder im Volume-Subsystem ansiedeln. Die eigentlichen Algorithmen aus *Common/Wipe.(c|h)* können hierfür verwendet werden.

Windows

Unter Windows kann man die entsprechende Funktionalität direkt im Mount-Subsystem implementieren, unter Verwendung vom *Common/Wipe.(c|h)*.

2.12.3 Bewertung

Unter beiden Betriebssystemen stellt die Erweiterung keinen großen Eingriff dar und sollte unproblematisch zu implementieren sein. Für die Änderungen sind je Betriebssystem **2 Personentage** zu veranschlagen.

2.13 *Full-Disc-Encryption unter Linux*

2.13.1 Beschreibung

Die Full-Disc-Encryption unter Windows verwendet einen speziellen Bootloader von TrueCrypt, der lediglich einen *Interrupt Filter* installiert und dann den Bootsektor der zu bootenden Partition ausführt. Da dieser Vorgang nicht abhängig vom verwendeten Bootloader zu sein scheint und auch *GRUB* in einen Partitionsbootsektor installiert werden kann, kann diese Methode genauso für Linux funktionieren. Experimentell wurde bereits bestätigt, dass *GRUB* erfolgreich einen Linuxkernel von einer vollverschlüsselten Platte laden und ausführen kann.

Sobald dann der Kernel gebootet ist und das *initramfs* ausführt, müsste dann innerhalb dieses Initramfs das Device mit TrueCrypt und dem *Linux Device Mapper* neu geöffnet werden, bevor der Bootvorgang fortgesetzt wird. Dies erfordert jedoch lediglich eine Anpassung im *initramfs* des jeweiligen Linuxsystems.

Die Installation des Linuxbetriebssystems innerhalb eines verschlüsselten Volumes kann aus einem bereits laufenden Betriebssystem erfolgen. Es ist theoretisch auch möglich, ein bestehendes System zu verschlüsseln. Dies erfordert jedoch aufwändige Zusatzimplementierung, da hierzu das Originaldateisystem verkleinert werden muss (abhängig vom jeweiligen Dateisystem) und die reine Windowsimplementierung dieser Funktionalität portiert werden müsste.

Wird die Verwendung vollverschlüsselter Linuxsysteme angestrebt, so ist die Installation innerhalb eines verschlüsselten Volumes nicht nur einfacher sondern

auch sicherer und sollte daher angestrebt werden.

2.13.2 Ort notwendiger Änderungen

Sofern der TrueCrypt-Bootloader mit GRUB den Linuxkernel starten kann, sind lediglich Änderungen im Initramfs des Linuxsystems erforderlich. Der Anpassungsprozess ist distributionsbezogen automatisierbar. Für die Komplexität ist hierbei entscheidend, ob die Distribution eine solche Anpassung des initramfs schon vorsieht (z.B. Ubuntu) und wie komplex das Festplattensetup des jeweiligen Einsatzszenarios ist.

2.13.3 Bewertung

Die Änderungen im Initramfs sind minimal und sollten binnen **1-2 Personentagen** zu bewältigen sein.

Kompatibilität zwischen Distributionen

Es ist davon auszugehen, dass die eigentliche Implementierung (Skript für Initramfs) nur minimale Inkompatibilitäten zwischen den Distributionen aufweisen wird. Wie bereits vorher beschrieben, muss jedoch der Integrationsprozess für jede Distribution separat implementiert werden. Insbesondere muss darauf geachtet werden, dass der betriebssystemeigene Updateprozess diese Änderungen berücksichtigt.

Test- und Integrationsaufwand für verschiedene Distributionen

Zusätzlich zur eigentlichen Implementierung sind für Integration und Stabilitätstests pro Distribution zusätzlich **4 Personentage** zu veranschlagen.

2.14 Sprachversionen unter Linux

2.14.1 Beschreibung

Die Linuxversion von TrueCrypt bietet von Haus aus oberflächlich keine Möglichkeit, die Sprachversion bei der Installation oder im Betrieb zu ändern.

Ein näherer Blick auf die unter Linux verwendete GUI bzw. das verwendete Konsolenprogramm zeigt jedoch, dass Installationen in verschiedenen Sprachen vorgesehen sind. Hierzu wird, falls die GUI unter Windows gebaut wird (was derzeit nie der Fall ist), die Windowsfunktionen `FindResource`/`LoadResource` verwendet, um ein XML File mit Übersetzungen zu laden. Unter Linux wird eine statische Version einer solchen Resource als Header eingebunden.

2.14.2 Ort notwendiger Änderungen

Um eine andere Sprache installieren zu können, muss in der `Resources`-Klasse im *Main*-Subsystem die Funktion `GetLanguageXml()` für Linux angepasst werden. Diese muss das entsprechende XML File laden und in das gleiche Format bringen, das Windows zurückliefern würde.

Sollen mehrere Sprachen parallel installierbar und im Programm umschaltbar sein, so muss hier ebenfalls die Funktion `GetLanguageXml()` angepasst werden, sodass die gewünschte Sprache übergeben werden kann. Weiterhin muss in der `Resources`-Klasse eine Abfrage der verfügbaren Sprachen ermöglicht werden. Schließlich muss die Klasse `LanguageStrings` erweitert werden, um im Betrieb das Sprachmapping zu ändern, ggf. müssen daraufhin alle aktiven Dialoge neu initialisiert werden.

2.14.3 Bewertung

Die Änderung zur Installation einer anderen Sprache (ausgenommen die eigentliche Übersetzung) ist minimal und innerhalb von **1 Personentag** zu bewältigen.

Sollen mehrere Sprachen parallel installierbar und wählbar sein, so ist zusätzlich ein Arbeitsaufwand von **4 Personentagen** zu veranschlagen.