HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Laura Pfeiffer

Forensische Analyse des Apple File System (APFS)

2017

BACHELOR THESIS

Mrs.
Laura Pfeiffer

Forensic Analysis of the Apple File System (APFS)

2017

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Forensische Analyse des Apple File System (APFS)

Autorin:
Laura Pfeiffer

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO14W2-B

Erstprüfer:
Prof. Dr. Christian Hummert

Zweitprüfer:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Mittweida, Oktober 2017

Faculty of **Applied Computer and Life Sciences**

BACHELOR THESIS

Forensic Analysis of the Apple File System (APFS)

Author:
Laura Pfeiffer

Study Programme:
General and Digital Forensics

Seminar Group:
FO14W2-B

First Referee:
Prof. Dr. Christian Hummert

Second Referee:
Prof. Dr. rer. pol. Dirk Pawlaszczyk

Mittweida, October 2017

Bibliografische Angaben

Pfeiffer, Laura: Forensische Analyse des Apple File System (APFS), 61 Seiten, 12 Abbildungen, 9 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften

Bachelorarbeit, 2017

Referat

Diese Arbeit beinhaltet eine erste Strukturanalyse des neuen Apple File System. Es wurden die Datenträgerstrukturen eines APFS-Containers anhand mehrerer unter macOS „Sierra“erstellten Images analysiert. Dabei wurde festgestellt, dass Grundzüge des Vorgänger-Dateisystems HFS+ auch bei APFS Anwendung finden, das neue System jedoch um einige Funktionalitäten verändert respektive erweitert wurde. Dazu zählen im Wesentlichen Snapshots, Copy-on-Write-Verfahren, Verschlüsselung und eine flexiblere Art der Partitionierung. Bezogen auf computerforensische Untersuchungen ist die Verschlüsselung der Nutzerdaten auf Apple-Computern eher als hinderlich zu betrachten. Die anderen Features werden dabei nach aktuellem Kenntnisstand keine oder leicht positive Auswirkungen haben.

Abstract

This Bachelor Thesis contains a first analysis of the structure of Apple File System (APFS). The structures of an APFS-container were analyzed on some images (created on macOS Sierra). Some basics of the previous file system HFS+ can be found in APFS, but there are also new features like snapshots, copy-on-write, encryption and flexible volumes. The impact of the file system basics an new features on forensic casework were discussed and evaluated as positive, neutral or negative at the end of this work.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

ca.	circa
ggf.	gegebenenfalls
u. a.	unter anderem
usw.	und so weiter
z. B.	zum Beispiel

V. Vorwort

Wissenschaftliche Publikationen zum Thema oder fundierte Erkenntnisse über die von Apple veröffentlichen Angaben hinaus existierten zur Zeit der Bearbeitung nicht. Viele Anhaltspunkte hat die Arbeit des Forensikers Jonas Plum alias Cugu geliefert, welche er in seinem Blog und einer Kaitai Struct Datei ([Cugu Kaitai]) veröffentlicht hat. Diese Datei wurde kurz vor Beendigung dieser Bachelorarbeit grundlegend erweitert und verändert, weswegen einige Inhalte hier keine Beachtung mehr finden. Sie wurde jedoch ergänzend als Text im Anhang an diese Arbeit angefügt.

1 Einleitung

1.1 Motivation

Zur forensischen Untersuchung von IT-Systemen und elektronischen Datenträgern kann es entweder nach technischem oder menschlichem Versagen sowie bei der Aufklärung delinquenten Verhaltens kommen. Dabei treten von Fall zu Fall verschiedene Fragestellungen zu Benutzern, Ort und Zeit der Geschehnisse, verwendeten Programmen oder bestimmten Daten auf. Es kann dabei z. B. darum gehen, wer ein Gerät wann verwendet hat und was für Dateien dabei angelegt, verändert oder gelöscht worden sind. Des Weiteren kann es notwendig sein, Metadaten zu Dateien, frühere Dateiversionen oder gelöschte Inhalte zu finden. Viele der benötigten Informationen sind nicht flüchtig, d. h. sie liegen an mindestens einer Stelle im Speicher permanent vor. Wo und wie solche Dateien abgelegt sind und auch wieder aufgefunden werden können, legt in der Regel das Dateisystem fest. Sowohl vom Benutzer angelegte Dateien, als auch Anwendungsdaten greifen darüber auf den Speicher zu. Dazu verwaltet das Dateisystem die vom Betriebssystem bereit gestellten Speicherbereiche möglichst effizient und sicher. Für die meisten Dateisysteme ist mehr oder weniger genau bekannt, wie sie funktionieren und z. B. relevante Daten gefunden werden können. Viele bekannte und weit verbreitete Dateisysteme wurden auch für Forensik Suites implementiert, sodass diese verschiedene Datenträgerimages automatisch interpretieren können.
Seit Juni 2016 führt das Unternehmen Apple Inc. schrittweise sein neues Dateisystem Apple File System (APFS) ein, welches nach und nach den Vorgänger HFS+ ablösen und somit zukünftig der Standard auf allen Apple-Geräten sein soll - von der Smartwatch bis zum Desktop PC. Da vor allem Apples Mobilgeräte weit verbreitet sind, werden schon in näherer Zukunft die ersten Geräte mit APFS in den Fokus von computerforensischen Untersuchungen rücken.

1.2 Zielstellung

Beim Apple File System wird es einige Neuerungen und Änderungen gegenüber HFS+ geben, die vor allem Sicherheit aber auch effiziente Speichernutzung und hohe Arbeisgeschwindigkeiten bieten sollen. Zu den geplanten Eigenschaften zählen unter anderem die Optimierung der Speichervorgänge für Flashspeicher, Datensicherungen über Snapshots, bessere Nutzung physikalischen Speichers durch Space Sharing sowie Klonen und Copy-on-Write Verfahren für schnelles und sicheres Kopieren und Ändern von Dateien [APFS Guide 1]. Neben der grundsätzlichen Funktionsweise des Dateisystems müssen die genannten sowie weitere neue Features unter folgenden Fragestellungen betrachtet werden: Wo sind die für computerforensische Untersuchungen relevanten Daten zu finden? Wie können ggf. gelöschte Dateien rekonstruiert werden? Können

Veränderungen an Dateien zeitlich nachvollzogen werden? Was für Metadaten sind wo abgelegt? Außerdem ist zu klären, welche Auswirkungen die Einführung des neuen Dateisystems auf die Forensik haben wird. Dabei soll diskutiert werden, welche der zentralen Features die forensischen Untersuchungen begünstigen oder erschweren.

2 Grundlagen

Dieses Kapitel beinhaltet allgemeine Grundlagen zur Speicherung und Verwaltung von Dateien, Verschlüssleung von Daten und zu Apples Software im Allgemeinen. Außerdem wird das Konzept und die theoretische Umsetzung des Apple File System behandelt.

2.1 Allgemeine Grundlagen

In diesem Abschnitt wird die Speicherverwaltung durch Betriebs- und Dateisysteme behandelt. Es geht außerdem um Begriffe wie Partitionierung und verschiedene permanente Speichermedien.

2.1.1 Betriebs- und Dateisysteme

„Ein Dateisystem ist eine Organisationsform von Daten, die in Form von Dateien vorliegen und logisch zu Verzeichnissen zusammengefasst werden.“ [Moritz/Steffens/Steffens, 2010, S. 187] Es realisiert nach den Vorgaben und Voraussetzungen des Betriebssystems die physikalische Ablage der Daten und stellt sie dem Benutzer in einer abstrakten Darstellung bereit. Dabei dient das Dateisystem als Verbindungsstück zwischen beiden Komponenten. Es regelt den Zugriff auf die Dateien und Verzeichnisse. Das passiert über die Zuordnung der von einer Datei oder einem Verzeichnis allokierten Blöcke eines Speichermediums zu einem für den Benutzer lesbaren Datei- oder Verzeichnisnamen. Das kann über verschiedenste Strukturen erfolgen, z. B. eine verkettete Liste wie bei FAT, eine Datenbankstruktur wie bei NTFS oder eine Indexstruktur wie bei HFS+.

2.1.2 Permanente Massenspeicher

Die wichtigsten Funktionen eines Dateisystems sind die effiziente permanente Speicherung und das schnelle Lesen von Dateien auf einem Speichermedium. Physikalisch permanente Massenpeicher sind z. B. magnetische Festplatten (HDDs) oder Flashspeicher (SSDs). Diese unterscheiden sich grundlegend in Zugriffszeiten und Anzahl möglicher Schreibvorgänge. Flashspeicher sind zwar wesentlich schneller, robuster und bieten eine gleichmäßige Leseperformance über alle Bereiche der Platte, „altern“ jedoch im Gegensatz zu herkömmlichen Festplatten. HDDs bestehen aus mehreren magnetischen, rotierenden Platten und Schreib- bzw. Leseköpfen. Die Platten teilen sich horizontal in verschiedene Sektoren und vertikal in verschiedene Zylinder. Jeder physische Block wird logisch über eine Kombination aus Zylinder, Schreib-/Lesekopf und Sektor adressiert. SDDs bestehen aus vielen Floating Gate Transistoren, die entweder in Reihe oder

parallel geschaltet kombiniert werden und jeweils zwei Zustände annehmen können. Logisch adressiert werden die Pages und Blocks mithilfe des Flash Controllers, der die logischen Speicheradressen physischen Bereichen des Flashspeichers über eine Lookup Table zuordnet. Beide Arten von Speichermedien sind jedoch für die Speicherung von permanenten Daten zuständig und können mehrere logische Laufwerke bzw. Partitionen enthalten, die jeweils durch ein Dateisystem verwaltet werden.

2.1.3 Partitionierung

Als Partition bezeichnet man einen zusammenhängenden Teil eines Speichermediums [TechTerms]. Verwaltet werden die Partitionen eines Speichermediums über eine Partitionstabelle. In dieser Tabelle sind Angaben zur Organisation des Datenträgers abgelegt, z. B. Anzahl, Größe, Position und Typ der einzelnen Partitionen. Auf aktuellen Rechnern ist, aufgrund der verwendeten 64-Bit-Architektur von Intel, GPT das am häufigsten vorkommenden Partitionierungsschema. Da auch Apple diese Architektur seit 2006 auf seinen Computern verwendet [Macprime1 2009], findet man dort meistens ebenfalls GPT-partitionierte Speicher. Abbildung 2.1 zeigt den Grundaufbau einer GPT-partitionierten Platte.

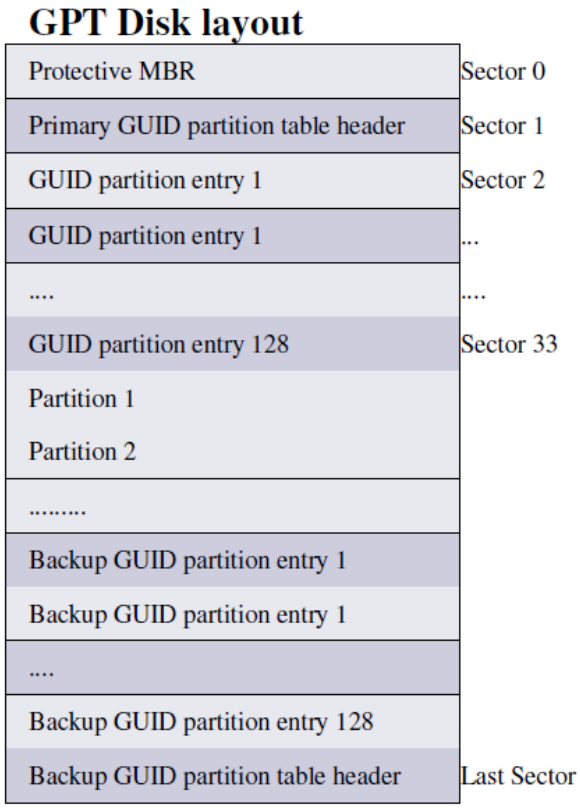


Abbildung 2.1: GPT Platten Layout - Darstellung von [Wilkinson 2012]

Der Protective MBR soll vor Zugriffen durch Systeme schützen, die nur das ältere Par-

titionierungsschema MBR interpretieren können und markiert dafür die gesamte Platte als belegt. Im nächsten Sektor legt der GPT Header, welcher mit der Signatur EFI PART beginnt, die Position des Backup Headers, des ersten und letzten durch Partitionen nutzbaren Sektors sowie Anzahl und Größe der Partitionseinträge fest. Diese folgen dann normalerweise in den Sektoren 2-33. Pro Sektor können Informationen zu vier Partitionen abgelegt sein. Diese beinhalten eine Partitionstyp-ID, Start- und Endsektor sowie den Namen der jeweiligen Partition.

2.2 Verschlüsselung

APFS implementiert verschiedene Möglichkeiten der nativen Verschlüsselung, also einer einfachen Verschlüsselung, die ohne zusätzliche Programme umzusetzen ist. Ursprünglich diente die Verschlüsselung zum Schutz geheimer Nachrichten vor dem Mitlesen durch Dritte. Heute werden verschiedene elektronische Daten verschlüsselt, um sie ebenfalls vor dem Zugriff durch Unbefugte zu schützen. Davon können sowohl einzelne Dateien oder Partitionen als auch ganze Festplatten betroffen sein. Um dies zu realisieren, existieren verschiedene Verfahren und Verschlüssselungsmodi.
Im Wesentlichen wird immer ein Klartext bzw. Dechiffrat durch einen oder mehrere Schlüssel zu einem Geheimtext bzw. Chiffrat verschlüsselt. Letzterer sollte nur dann interpretiert werden können, wenn der verwendete Schlüssel bekannt ist. Dabei unterscheidet man grundsätzlich symmetrische und asymmetrische Verfahren, wie in Abbildung 2.2 dargestellt. Bei symmetrischen Verfahren benutzen beide Kommunikationspartner ein und denselben Schlüssel zur Verschlüsselung und Entschlüsselung, bei asymmetrischen wird auf der einen Seite ein öffentlicher Schlüssel zum Verschlüsseln und ein privater auf der anderen Seite zum Entschlüsseln genutzt.
An dieser Stelle wird nur auf die symmetrischen Verfahren eingegangen, da lediglich solche für diese Arbeit von Interesse sind. Man unterscheidet bei der symmetrischen Verschlüsselung Stromchiffren und Blockchiffren. Diese Unterscheidung trennt Verfahren, bei denen Buchstaben bzw. Einzelelemente direkt hintereinander verschlüsselt werden und solche, bei denen der gesamte Text bzw. zu verschlüsselnde Bereich zuvor in Blöcke geteilt wird. Blockchiffrierung kann dabei in verschiedenen Betriebsmodi ablaufen.

2.2.1 Betriebsmodi der symmetrischen Verschlüsselung

Der einfachste und dabei recht unsichere Modus ist der Electronic Code Book Mode (ECB). Dieser Modus verschlüsselt die einzelnen Klartextblöcke jeweils mit dem gleichen Schlüssel und unabhängig von den anderen Klartextblöcken. Das führt dazu, dass gleiche Blöcke immer gleich verschlüsselt werden. Daher können im Chiffrat Muster erkannt und der Schlüssel so geknackt werden.
Dieses Problem löst unter anderem der Cipher Text Chaining Mode (CBC). Dabei wer-

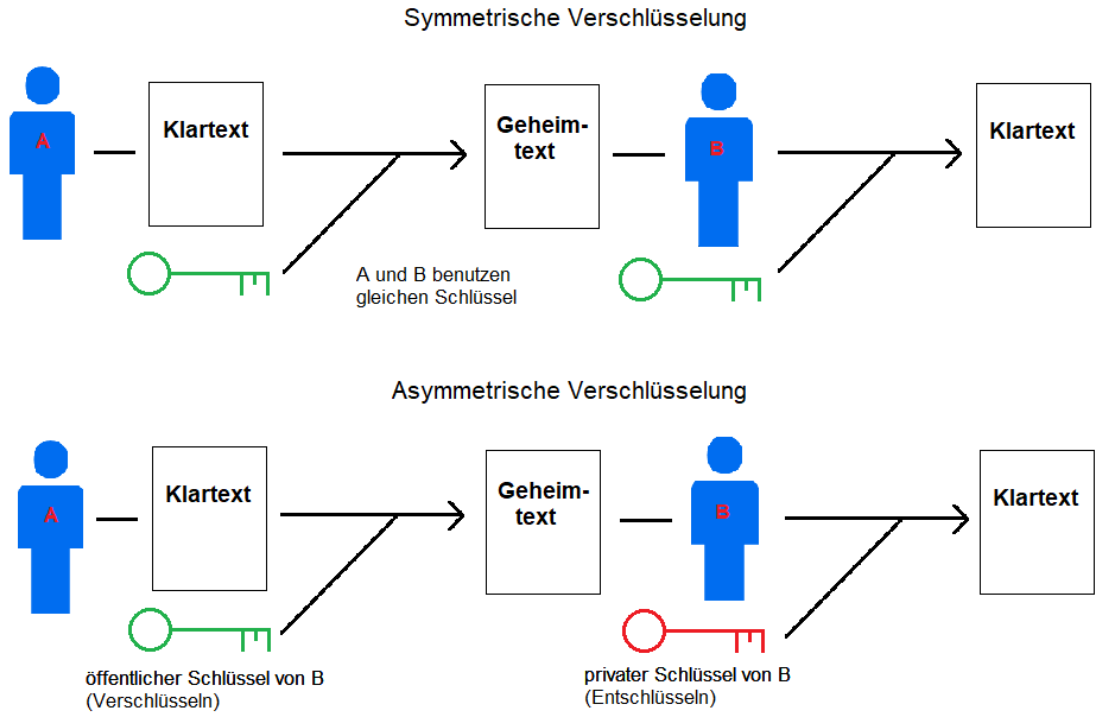


Abbildung 2.2: Symmetrische und Asymmetrische Verschlüsselung - eigene Darstellung

den die verschiedenen Klartextblöcke jeweils in Abhängigkeit vom vorhergehenden chiffrierten Block verschlüsselt. Der erste Block wird dabei mit einem Initialisierungsvektor XOR-verknüpft und dann mit dem Schlüssel chiffriert. Die weiteren Klartextblöcke werden jeweils mit dem vorhergehenden Geheimtextblock XOR-verknüpft und anschließend mi dem Schlüssel chiffriert. Gleiche Klartextblöcke führen bei diesem Modus zu unterschiedlichen Geheimtextblöcken.
Beim Cipher Feedback Modus (CFB) ist ein Geheimtextblock ebenfalls vom vorhergehenden abhängig, jedoch wird hier erst der Initialisierungsvektor bzw. der vorhergehende Cipherblock mit dem Schlüssel verschlüsselt und anschließend mit dem jeweiligen Klartextblock XOR-verknüpft.
Der Output Feedback Modus (OFB) funktioniert ähnlich wie CFB, jedoch wird hier ab dem zweiten Block nicht der vorhergehende Cipherblock sondern der verschlüsselte Initialisierungsvektor als neuer Initialisierungsvektor zugeführt und dieser nach erneuter Verschlüsselung mit dem Klartextblock XOR-verknüpft.
Der xor-encrpyt-xor-based tweaked-codebook mode with ciphertext stealing [XTS] - kurz XTS - gilt als einer der sichersten Modi. Er verwendet zwei verschiedene Schlüssel sowie eine Diffusionsfunktion [Kingston]. Ein sogenannter Tweak-Wert wird mit dem ersten Schlüssel verschlüsselt und anschließend mit der Diffusionsfunktion XOR-verknüpft; dieser Teil wird wiederum mit dem Klartext XOR-verknüpft; dieser dann mit dem zweiten Schlüssel verschlüsselt und nochmals mit dem Ergebnis aus Tweak-Wert, erstem Schlüssel und Diffusionsfunktion XOR-verschlüsselt [Kingston].

2.2.2 Der Advanced Encryption Standard

Ein aktuell häufig verwendeter Verschlüsselungsalgorithmus ist der Advanced Encryption Standard (AES) als Nachfolger des Data Encryption Standard (DES). DES gilt als nicht sicher, da der Schlüssel mit seiner Länge von effektiv 56 Bit mit verhältnismäßig geringem Aufwand herausgefunden werden kann [WhatIs: DES]. Auf AES sind bisher jedoch keine erfolgreichen Angriffe bekannt, es existieren jedoch einige theoretische Ansätze, die ständig erweitert und verbessert werden. Bei AES handelt sich um ein Blockchiffrierverfahren, bei welchem die Blocklänge 128 Bit beträgt und die Schlüssellänge 128, 192 oder 256 Bit betragen kann [WhatIs: AES]. Der hinter AES stehende Rijndael-Algorithmus kennt theoretisch jedoch sowohl Block- als auch Schlüssellängen von 128, 160, 192, 224 oder 256 Bit.
Die einzelnen Blöcke werden als Array bzw. Tabelle mit vier Zeilen und mehreren Spalten dargestellt, wobei jede Zelle für ein Byte steht. Die Anzahl der Spalten ist abhängig von der Blocklänge und somit bei AES 4 ($\frac{128 Bit Blocklänge}{8 Bit pro Zelle * 4 Zeilen}$) [WhatIs: Rijndael]. Jeder Block durchläuft beim Rijndael-AES je nach Schlüssellänge 10, 12 oder 14 Runden Verschlüsselung jeweils inklusive einer Schlussrunde. Es wird dabei nicht in jeder Runde mit dem gleichen Schlüssel gearbeitet, sondern der jeweilige Benutzerschlüssel je nach Anzahl der benötigten Runden erweitert und in verschiedene Rundenschlüssel aufgeteilt [WhatIs: AES]. Mit Ausnahme der Schlussrunde laufen alle Runden jeweils nach dem gleichen Muster ab:

1. Vor den eigentlichen Verschlüsselungsrunden wird zuerst der erste Rundenschlüssel auf jeden Klartextblock addiert. Im ersten Schritt jeder Runde wird dann jedes Byte eines Blocks mithilfe einer Substitutionsbox monoalphabetisch verschlüsselt.
2. Anschließend werden die einzelnen Zeilen der Tabelle nach einem bestimmten Muster nach links verschoben. Für die von AES verwendete Blocklänge von 128 Bit ist dieses Muster in Abbildung 2.3 skizziert.
3. Im dritten Schritt werden die Inhalte der Spalten vermischt. Die so entstandenen Blöcke werden anschließend mit dem jeweiligen Rundenschlüssel XOR-verknüpft.

In der Schlussrunde entfällt Schritt 3.

2.3 Überblick zu Apples Software

Das 1976 von Steve Jobs, Stephen Gary Wozniak und Ronald Gerald Wayne gegründete Unternehmen Apple Inc. war in seinen ersten Jahren von Chaos und rasantem Wechsel zwischen Erfolgen und Fehlschlägen geprägt. Erst um die Jahrtausendwende herum begann die Etablierung eines gut nachvollziehbaren und unternehmenstechnisch relativ stabilen Gesamtkonzepts [Macprime2 2009]. Für Apples Geräte iMac, iPod, iPad, iPhone, Apple Watch und Apple TV gibt es im Wesentlichen die Betriebssyste-

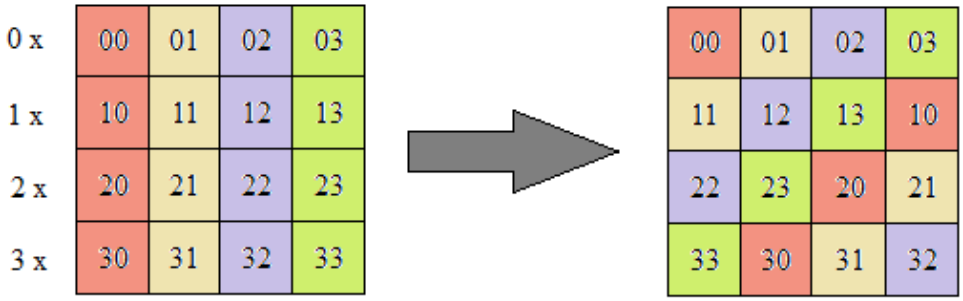


Abbildung 2.3: Schritt 2 bei AES: ShiftRows - eigene Darstellung

me macOS, iOS, watchOS und tvOS. Grundlage für alle bildet dort OSX bzw. macOS, das von Desktopcomputern und Notebooks verwendete Betriebssystem. Die Grundzüge dessen finden sich auch bei iOS, dem Betriebssystem für die Mobilgeräte iPhone, iPad und iPod. Wiederum darauf aufbauend sind tvOS für die Apple TVs und watchOS für die Apple Watches. Das Dateisystem als der Teil des Betriebsystems, der die Zuordnung von physikalischen Speicheradressen zu logischen Datei- und Verzeichnisnamen regelt, basiert bisher auf dem hierarchischen Dateisystem HFS. Die Desktopcomputer und Laptops verwenden im Allgemeinen HFS+, die Mobilgeräte HFSX. Jedoch ist die Umstellung bei allen Betriebssystemen zu APFS als Dateisystem im Gange. Als Entwicklervorschau wurde APFS bereits mit macOS 10.12 „Sierra“ eingeführt, offizieller Start war am 13.06.2016 mit macOS 10.12.6. Bei macOS 10.13 „High Sierra“ ist APFS standardmäßig implementiert. Bei iOS wurde neun Monate später ab Version 10.3 auf APFS umgestellt. In Tabelle 2.1 sind die Betriebssystemversionen aufgelistet, mit denen APFS als Standarddateisystem eingeführt wird bzw. eingeführt worden ist.

Geräte	APFS ab OS-Version
iMac, MacBook (Desktop PC, Laptop)	macOS 10.12.6 „Sierra“
iPad, iPhone (Tablet, Smartphone)	iOS 11.0
Apple Watch (Smartwatch)	watchOS 4.0
Apple TV (Fernseher)	tvOS 11.0

Tabelle 2.1: Einführung von APFS - Zusammenstellung aus [Apple macOS], [Apple iOS], [Apple watchOS] und [Apple tvOS]

2.4 Der Vorgänger HFS+

Viele Jahre wurde als Standard-Dateisystem bei Apple Geräten das hierarchische Dateisystem HFS bzw. HFS+ verwendet. Davon existieren verschiedene weitere Versionen,

z. B. HFS+ mit Journaling (jHFS+) oder für Mobilgeräte (HFSX).

2.4.1 Allgemeines

Hierarchisches Dateisystem bedeutet, dass mehrere Unterverzeichnisse unter dem root-Verzeichnis existieren, z. B. System, Applications, usr oder dev. Eine HFS+ Partition teilt sich in Sektoren, diese wiederum in Cluster, diese in Allocation Blocks, welche aus mehreren Blocks von je 512 Bytes bestehen. Alle mehrere Byte großen Werte eines HFS+ Volumes sind im Big Endian Format abgelegt, lediglich der GPT-Header speichert im Little Endian Format. Die Zeitstempel bei HFS+ sind 32 Bit Integer-Werte in Apple Mac Time (Sekunden seit dem 01.01.1904).

2.4.2 Datenträgerstruktur

In einer HFS+ Partition gibt es immer die Datenträgerstrukturen Volume Header, Allocation File, Extends Overflow File, Catalog File, Attributes File und Startup File. Der Volume Header enthält allgemeine Informationen zum Volume und dessen Struktur, z. B. Versionsnummern, verschiedene Zeitstempel, Größe, Anzahl und Belegung der Allocation Blocks sowie Position und Größe der Dateien Allocation File, Extends Overflow File, Catalog File, Attributes File und Startup File. Wichtig zum Finden von Dateien ist das Catalog File, welches Metainformationen und die Hierarchie zu allen Dateien und Verzeichnissen speichert. Dies passiert über eine B-Tree-Struktur.

2.4.3 B-Trees

Das Abspeichern und Wiederaufrufen von Dateien wird über B-Bäume organisiert, was gewährleisten soll, dass Dateien schnell und effizient gefunden werden. B-Trees sind balancierte Suchbäume, die aus Nodes (bzw. Knoten) bestehen. Jede Node hat eine ID, über welche sie adressiert wird und kann mehrere Einträge, bestehend aus Key und Daten, enthalten. Die Nodes haben verschiedene Level und können entweder Leaf Node, Index Node, Header Node oder Root Node sein. Dabei ist die Header Node immer die erste mit der ID 0 und auf Level 0. Sie enthält allgemeine Informationen zur jeweiligen B-Tree-Datei und die Root Node ID. Die Root Node hat immer den höchsten Level und kann entweder nur Index-Einträge oder nur Leaf-Einträge enthalten. Eine Index Node enthält Verweise auf andere Index Nodes oder auf Leaf Nodes. Leaf Nodes haben immer den Level 1 in der Hierarchie. Abbildung 2.4 zeigt eine beispielhafte B-Tree-Struktur, die Informationen zu drei Dateien speichern kann.

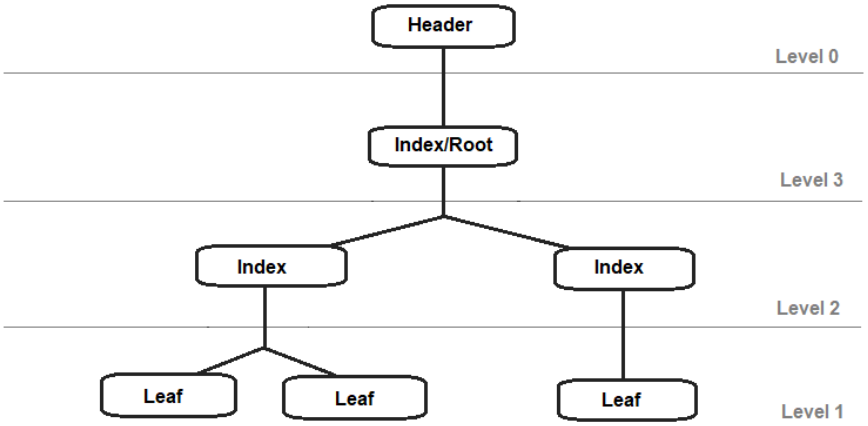


Abbildung 2.4: Beispielhafte B-Tree Struktur - eigene Darstellung

2.5 Konzept des Apple File System

Im Newsroom der Apple Website wurde im Juni 2017 zu verbesserten System-Technologien unter MacOS „High Sierra“folgendes angekündigt: „Apple Dateisystem (APFS) bietet verbesserte Leistung, Sicherheit und Zuverlässigkeit der Daten und bietet eine Grundlage für zukünftige Speicherinnovationen. Mit einer fortschrittlichen, für die heutigen massiven Speichertechnologien optimierten Architektur, macht APFS gängige Vorgänge wie das Kopieren von Dateien und Verzeichnissen unmittelbarer, hilft Daten vor Stromausfällen und Systemabstürzen zu schützen und hält Dateien mit nativer Verschlüsselung sicher und geschützt.“ [Apple Newsroom] Wie Leistung, Sicherheit und Zuverlässigkeit verbessert und gängige Vorgänge unmittelbarer werden sollen, wird auf Apples Entwickler-Seite spezifiziert. APFS soll für die Speicherung von Daten auf Flashspeichern optimiert sein und folgende neue Merkmale mitbringen: starke native Verschlüsselung, Copy-On-Write für Metadaten, gemeinsame Nutzung des physischen Plattenplatzes durch mehrere logische Laufwerke, Klone von Dateien oder Verzeichnissen, Schnappschüsse zur Datensicherung, atomares sicheres Speichern und verbesserte Dateisystem-Grundlagen. Laut Herstellerangaben ersetzt APFS ab macOS „High Sierra“ und iOS 10.3 das bis dahin verwendete HFS+ bzw. HFSX als Standard-Dateisystem [APFS Guide 1].

2.6 Umsetzung des Apple File Systems

Dieser Abschnitt erklärt die Funktionsweise der geplanten Neuerungen bei APFS.

2.6.1 Allgemeines

Das Apple Filesystem speichert sämtliche Angaben so wie die meisten gängigen Dateisysteme im Little-Endian Format. Zeitstempel sind 64-Bit-Werte und werden in Nanosekunden seit dem 01.01.1970 UTC angegeben [Cugu's Blog 2017].
Die Dateinamen sind UTF-8 codiert nach Unicode 9.0 Standard. Unter iOS wird zwischen Groß- und Kleinschreibung unterschieden, unter macOS standardmäßig nicht, kann aber auch dort eingestellt werden. In der Regel sind die Dateinamen unter APFS unempfindlich gegenüber verschiedenen Normalisierungen [APFS Guide 4].
APFS teilt die Partitionen wie auch HFS+ in gleichmäßige Teile bzw. Blocks. Die allokierten Einheiten sind im Allgemeinen 4096 Byte groß, was einem Allocation Block von acht Blocks bei HFS+ entspricht. Einige dieser 4 KiB großen Strukturen beginnen mit einem 32 Byte großen Block Header, welcher u. a. den Blocktyp beinhaltet, z. B. Volume Superblock, Node oder Spacemanager [Cugu's Blog 2017]. Diese Header befinden sich in Blöcken, welche Metadaten beinhalten oder durch das Dateisystem navigieren. Allocation Blocks, in denen Daten abgelegt sind, haben keinen solchen Header. Es werden Partitionsgrößen bis 2^{63} Allocation Blocks und Dateigrößen bis 2^{63} Byte unterstützt [APFS Guide 2].
Es wird aufgrund von flexibler Partitionierung mit Containern gearbeitet. Ein Container bekommt eine feste Größe an Plattenplatz zugewiesen, welche dann innerhalb des Containers je nach Bedarf auf die einzelnen Partitionen verteilt wird.

2.6.2 Copy-on-Write Verfahren statt Journaling

Statt einen Plattenabsturz nur zu dokumentieren, wie bei Dateisystemen mit Journaling, soll ein bei Apple neues Copy-on-Write Verfahren vor solchen Ausfällen schützen und dazu führen, dass permanent ein konsistentes Dateisystem vorliegt. Copy-on-Write existiert bereits bei einigen neueren Dateisystemen und bedeutet, dass Kopien mittels Verlinkungen erstellt und neue Blöcke erst dann physisch beschrieben werden, wenn an der Datei bzw. einer Kopie davon neue Inhalte geschrieben worden sind [Golem 2016]. Auf diese Art soll auch physischer Platz auf der Platte eingespart werden.

2.6.3 Klone

Klone sind Kopien von Dateien, die keinen weiteren Platz benötigen [APFS Guide 3]. Der Klon enthält lediglich die normalen Metainformationen, wie Zeitstempel und Speicherort, sowie die Verweise auf die von der ursprünglichen Datei belegten Blöcke, ohne die Daten in diesen Blöcken selbst an eine andere Stelle zu schreiben. Wird eine Datei verändert, so werden nur die Änderungen an eine neue freie Stelle im Speicher geschrieben. Die Blöcke, deren Inhalt sich dabei nicht ändert, werden nicht erneut ge-

schrieben sondern von der neuen Dateiversion darauf verwiesen. Abbildung 2.5 verdeutlicht dies. Die Datei MyPresentation.key im root-Verzeichnis zeigte ursprünglich auf die beiden linken Blöcke der SSD. Sie wurde dann ins Verzeichnis Archive kopiert. Die Datei Archive/MyPresentation.key enthält nun die Verweise auf die zwei linken Blöcke. Die im root-Verzeichnis gespeicherte Version wurde anschließend geändert. Die Änderung betrifft jedoch nur den zweiten beschriebenen Block. Deswegen enthält die Datei nun Verweise auf den ersten ursprünglichen Block und den neuen (rechten) Block.

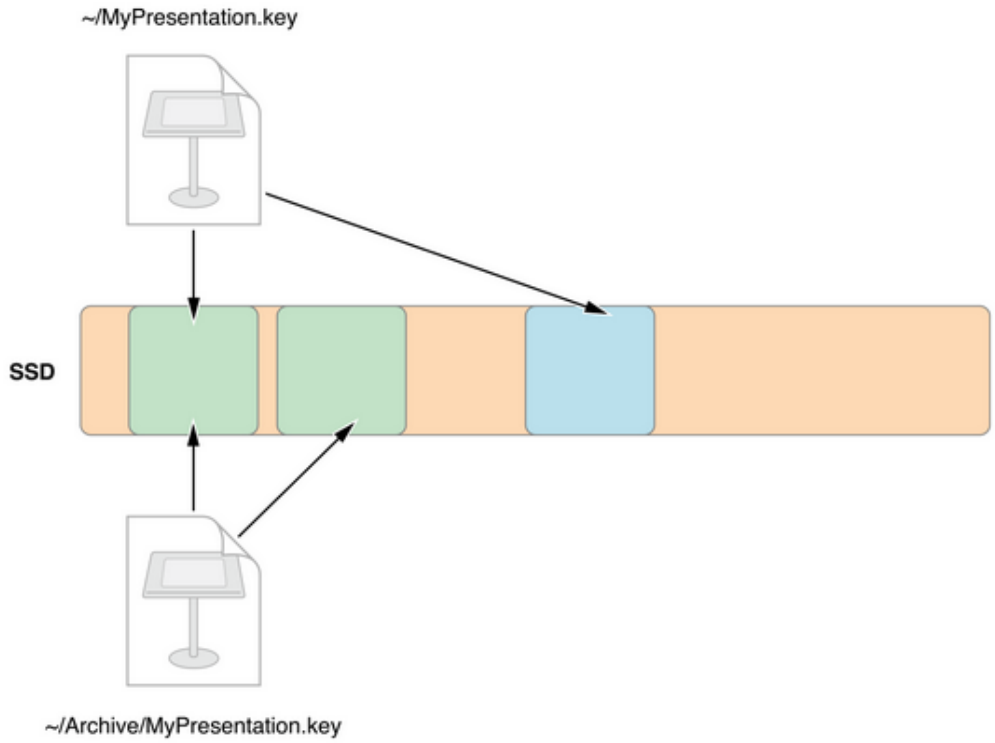


Abbildung 2.5: Funktion von Klonen - Darstellung von [APFS Guide 3]

2.6.4 Schnappschüsse

APFS erlaubt das Erstellen von Schnappschüssen. Das sind nur lesbare Dateien, die den Zustand des Dateisystems zu einem bestimmten Zeitpunkt speichern [APFS Guide 3]. So sollen Backups effizient, einfach und schnell ablaufen. Man kann darüber an einen bestimmten Stand zurückkehren und Änderungen nach dem Erstellen des Snapshots rückgängig machen. Abbildung 2.6 bezieht sich auf das unter Klone beschriebene Szenario. Der Snapshot wurde vor der Änderung an MyPresentation.key gemacht und verweist noch auf die alten Blöcke dieser Datei, sowie eine weitere Datei.

In der Entwicklervorschau von APFS ist das Erstellen von Snapshots bisher nur über das Terminal möglich. Dazu verwendet werden apfs_snapshot-Befehle, die in /System/Library/Filesystems/apfs.fs/Contents/Resources/ liegen. Tabelle 2.2 zeigt die Be-

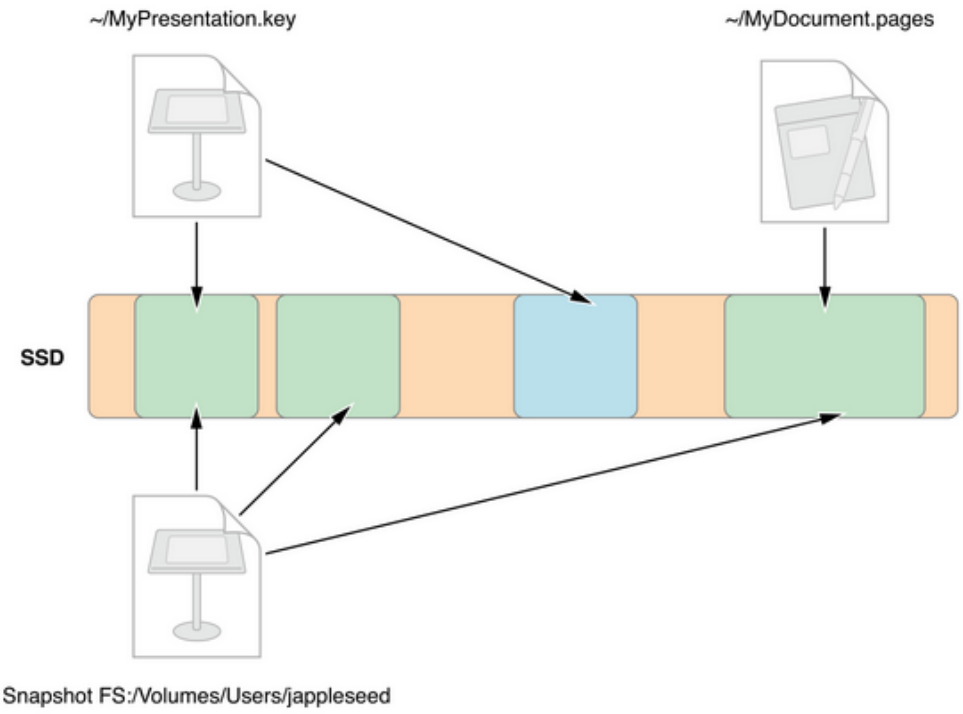


Abbildung 2.6: Funktion von Schnappschüssen - Darstellung von [APFS Guide 3]

fehle, die momentan zu Snapshots existieren. Der Partitionspfad wird immer am Ende des Befehls mit angegeben. In der Entwicklervorschau ist das Zurücksetzen der Partiti-

Befehl	**Effekt**
snapUtil -l <vol>	Auflistung aller Snapshots
snapUtil -c <snap> <vol>	Erstellen des Snapshots snap als Abbild von vol
snapUtil -r <snap> -n <new> <vol>	Umbenennung von snap in new
snapUtil -d <snap> <vol>	Löschen von Snapshot snap
snapUtil -b <snap> <vol>	Zurücksetzen von vol auf den Zustand von snap
snapUtil -s <snap> <vol> <mntpnt>	Mounten von snap am Mountpoint mntpnt

Tabelle 2.2: snapUtil-Befehle zu APFS-Snapshots - eigene Tabelle

on auf den Zustand eines Snapshots nur nach erneutem Mounten der Partition möglich. Außerdem wird davor gewarnt, dass apfs_snapshot nur zu Testzwecken verwendet werden soll und es möglicherweise später wieder entfernt wird. Sollten diese Befehle entfallen, dann wahrscheinlich nur, wenn sie durch andere ersetzt werden oder das Erstellen von Snapshots dann entweder über die Benutzerobefläche oder automatisiert stattfinden wird. Es ist unwahrscheinlich, dass diese Funktionalität gänzlich entfällt, da Apple in Verbindung mit APFS immer wieder die Schnelligkeit, Effizienz und Sicherheit betont. Snapshots als eine schnelle, einfache und sichere Möglichkeit Backups zu erstellen, leisten dabei einen wesentlichen Beitrag.

2.6.5 Verschlüsselung

Um Integrität und Vertraulichkeit der Nutzerdaten zu gewährleisten kann das Apple File System sowohl ganze Laufwerke als auch einzelne Dateien oder Metadaten verschlüsseln. Alles kann getrennt über verschiedene Schlüssel aber auch alles mit dem gleichen Schlüssel verschlüsselt werden. APFS nutzt dazu den Adavanced Encryption Standard in den Betriebsmodi CBC oder XTS [APFS Guide 3].

2.6.6 Sonstiges

Des Weiteren sollen Space Sharing, Fast Directory Sizing, Atomic Safe-Save Verfahren und Sparse Files dazu beitragen, dass APFS schnell und flexibel arbeitet und dabei möglichst nur so viel Speicherplatz wie wirklich notwendig belegt.
Durch Space Sharing also das Teilen des physischen Plattenplatzes durch mehrere logische Laufwerke entfällt der Zwang einer Partition von Beginn an eine feste Größe zuweisen zu müssen und neu zu partitionieren, wenn diese darüber hinaus wachsen soll. Es wird bei [APFS Guide 3] an folgendem Beispiel erklärt: Innerhalb eines Containers stehen 100 GB Speicher zur Verfügung und der Container beinhaltet zwei Partitioen - A und B. Volume A belegt bereits 10 GB und Volume B 20 GB. Für beide ist der freie, noch zur Verfügung stehende Speicherkapazität also 100-10-20 GB = 70 GB.
Fast Directory Sizing bedeutet lediglich, dass der durch ein Verzeichnis benötigte Speicherplatz zur Laufzeit berechnet und somit aktuell gehalten werden kann, wenn diese Option für das Verzeichnis zuvor ausgewählt wurde [APFS Guide 3].
Sicheres Speichern (Atomic Safe-Save) führt Umbenennungen in nur einer Transaktion aus, sodass es aus Perspektive des Benutzers so aussieht, dass die Operation entweder komplett oder überhaupt nicht statt gefunden hat [APFS Guide 3].
APFS erlaubt Sparse-Dateien, also Dateien bei denen die logische Größe höher ist als die physikalisch tatsächlich benötigte [APFS Guide 3]. Plattenplatz wird erst allokiert, wenn auch in die Null-Bereiche der Datei geschrieben wird. So kann wiederum etwas physikalischer Speicher gespart werden.

2.7 Datenträgerstrukturen bei APFS

In [Cugu's Blog 2017] werden folgende Strukturen beschrieben: Container Superblock, Spacemanager, Allocation Info File, Allocation File, B-Trees, Nodes, Checkpoints und Volume Superblock. Mit Ausnahme des Allocation Files besitzen alle diese Strukturen einen 32 Byte großen Header. Der Aufbau eines APFS Containers wird in Abbildung 2.7 dargestellt.

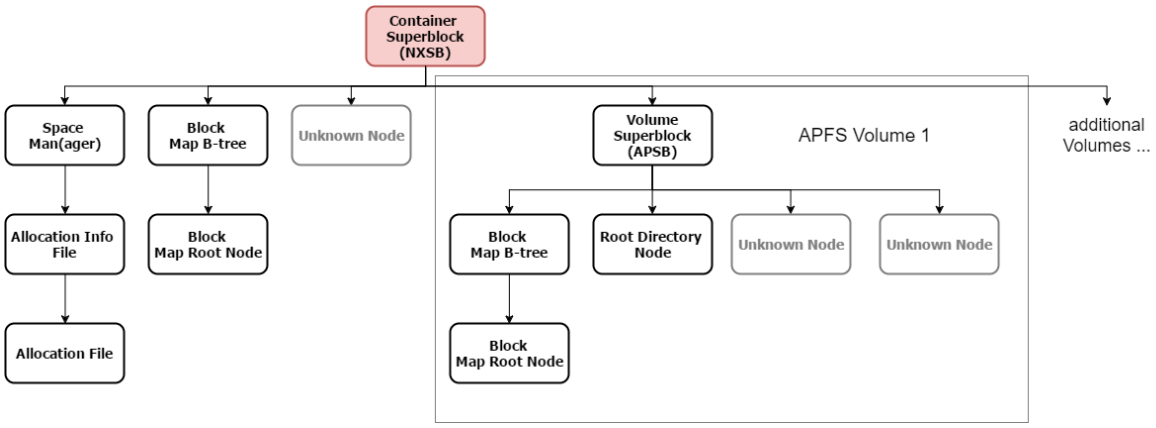


Abbildung 2.7: APFS Strukturen - Darstellung von [Cugu's Blog 2017]

2.7.1 Block Header

Im 32 Byte langen Block Header sind eine Checksumme, Block ID, Version, Blocktyp und Flags abgelegt. Die ersten acht Byte des Headers stellen die Checksumme dar, die nach Fletcher's Checksum Algorithmus berechnet wird [APFS Guide 4]. Es folgen je acht Byte für Block ID und Version. Dabei ist der Block mit dem größten Versionswert der aktuellste. Die Block ID dient zur Adressierung der Blöcke. Der Blocktyp ist zwei Byte lang, steht an Offset 0x18 des Headers und kann die in Tabelle 2.4 aufgelisteten Werte annehmen. Es folgen noch zwei Byte Flags, deren Wert 0x0000, 0x4000 oder 0x8000 sein kann. Ist der Wert 0x4000 gesetzt, handelt es sich bei der Block ID um die Position des Blocks ab Beginn des Containers [Cugu Kaitai]. Bei 0x1C des Block Headers kann die Art des Block-Inhalts abgelegt sein. Das wird beispielsweise bei Nodes verwendet, um zu klassifizieren, ob die Einträge der Node Informationen zu Dateien, Extents oder Anderem beinhalten. Tabelle 2.3 zeigt den Aufbau des Blockheaders, Tabelle 2.4 die Bedeutung der möglichen Werte für den Block Typ.

Offset	Länge	Beschreibung
0x00	8	Checksumme
0x08	8	Block ID
0x10	8	Version
0x18	2	Block Typ
0x1A	2	Flags
0x1C	2	Art des Inhalts

Tabelle 2.3: Aufbau des Blockheaders - eigene Tabelle nach [Cugu's Blog 2017]

2.7.2 Container Superblock

Am Anfang eines Datenträgers findet man mindestens einen Container Superblock. Dieser beginnt mit der Signatur NXSB und ermöglicht es überhaupt erst, auf das Dateisys-

Wert	Bedeutung
0x01	Container Superblock
0x02	Node
0x05	Spacemanager
0x07	Allocation Info File
0x0B	B-Tree
0x0C	Checkpoint
0x0D	Volume Superblock
0x11	unbekannt

Tabelle 2.4: Block Typen - eigene Tabelle nach [Cugu's Blog 2017]

tem zuzugreifen und Dateien zu finden. Zu diesem Zweck sind Pointer im Container Superblock gespeichert, die zum Einen auf den Spacemanager zeigen, welcher Informationen zur Belegung aller Allocation Blocks beinhaltet und zum Anderen auf die Block Map verweisen, welche Block IDs auf Blocknummern mappt. Man findet außerdem Informationen zu Blockgröße und Anzahl der Blöcke [Cugu's Blog 2017]. Wenn mehrere solcher Superblöcke existieren, so sind es verschiedene Versionen davon. Über den Block Header kann jedoch der aktuellste herausgefunden werden.
Die ersten vier Byte des Container Superblocks sind die Signatur NXSB. Darauf folgen vier Byte Blockgröße, welche standardmäßig 0x1000 ist, was 4096 Byte entspricht. An Offset 0x08 des Superblocks steht die Gesamtanzahl Blocks des Containers über acht Byte. Es folgen 16 Byte Padding oder reservierter Bereich.
Von den folgenden Bestandteilen des Container Superblocks ist wenig gesichert bekannt. Vermutlich gehören dazu verschiedene IDs und Versionsnummern sowie Counter und Offsets. An Offset 0x38 des Container Superblocks steht vermutlich eine acht Byte lange Block ID, bei 0x40 eine Versionsnummer über acht Byte, die um genau 1 größer ist als die Versionsnummer, die im jeweiligen Block Header steht. Es könnte sich also um die als nächstes folgende Version des Blocks handeln. An Offset 0x64 gibt es einen vier Byte großen Wert, der in Zusammenhang mit der Version steht. Ist die Versionsnummer in einem Block um 1 größer als in einem anderen, so ist eben benannter Wert um 4 größer. Er beginnt mit 2, wenn die Version 1 ist. An Offset 0x70 steht ein zweiter vier Byte großer Wert, der wiederum um 4 kleiner ist als der erste und in analoger Weise zum ersten von der Versionsnummer abhängig zu sein scheint. Bei [Cugu's Blog 2017] wird vermutet, dass sich an Offset 0x78 über acht Byte die ID des Spacemanagers befindet und die nächsten acht Byte auf einen Block Map B-Tree zeigen. Der Wert bei 0x80 entspricht allerdings der Anzahl belegter Blöcke vermindert um 2. Möglicherweise steht die Block Map also immer am Ende des Volumes. Tabelle 2.5 zeigt den möglichen Aufbau des Container Superblocks.

Offset	Länge	Beschreibung	Wert
0x00	4	Signatur	NXSB
0x04	4	Blockgröße	0x1000 = 4096 Byte
0x08	8	Gesamtanzahl Blöcke	
0x28	16	GUID	
0x38	8	unbekannte Block ID	
0x40	8	nächste Versionsnummer	aktuelle Version + 1
0x64	4	unbekannter Wert	Versionsnummer x 4 - 2
0x70	4	unbekannter Wert	Versionsnummer x 4 - 6
0x78	8	Block ID Spacemanager	0x400
0x80	8	Block Map Block	belegte Blöcke - 2
0x94	4	Anzahl Partitionen	
0x98	8	Offset Partition 1	
0xA0	8	Offset Partition 2	
...			

Tabelle 2.5: Aufbau des Container Superblocks - eigene Tabelle

2.7.3 Checkpoint

Zwischen allen Containersuperblöcken befindet sich jeweils ein Block Checkpoint. Der große Teil dieser Struktur ist von Version zu Version gleich. Lediglich ein Zähler, der entweder zwei oder vier mal auftaucht, verändert sich.
Ab Offset 0x08 findet man jeweils 32 Byte bisher nicht näher bestimmte IDs und Werte und darauf folgend den jeweiligen Counter über acht Byte. Im ersten Checkpoint findet man zwei dieser 40 Byte langen Strukturen, in allen anderen jeweils vier. Der Zähler wird checkpointübergreifend jedes Mal um 1 erhöht und ist demzufolge an der letzten Stelle im aktuellsten Checkpoint am größten.

2.7.4 Spacemanager

Der Spacemanager befindet sich am Anfang der Platte, wenige Allocation Blocks nach dem letzten Container Superblock. Er organisiert gemeinsam mit dem Allocation Info File und dem Allocation File die belegten und freien Blöcke des Containers. Man findet Informationen zu Blockgröße, Gesamtanzahl Blöcke, Anzahl freier Blöcke und zur Position des nächsten Allocation Info Files. Auch vom Spacemanager können wie beim Container Superblock mehrere Versionen vorliegen. Nach Beobachtungen der Autorin ist die Block ID des Spacemanagers immer 0x400.
Die ersten vier Byte geben erneut die Blockgröße an. An den Offsets 0x10 und 0x28 findet man jeweils acht Byte Gesamtzahl Blöcke und Anzahl freier Blöcke [Cugu's Blog 2017]. Im weiteren Verlauf gibt es verschiedene IDs, Counter, Offsets und andere Strukturen, über die aktuell noch nichts weiteres bekannt ist.

Offset	Länge	Beschreibung	Wert
0x00	4	Blockgröße	0x1000 = 4096 Byte
0x04	4	unbekannt	0x8000
0x08	4	unbekannt	0x7E
0x0C	4	unbekannt	0x1FB
0x10	8	Gesamtanzahl Blöcke	
0x18	8	unbekannt	
0x20	8	unbekannt	0x1
0x28	8	Anzahl freier Blöcke	
0x30	8	unbekannt	0x180
0x60	8	unbekannt	0x188
0x124	4	unbekannt	0x150
0x128	4	unbekannt	0x158
0x12C	4	unbekannt	0x160
0x130	8	aktuelle Version?	
0x138	8	vorhergehende Version?	
0x160	8	Offset zum Allocation Info File	

Tabelle 2.6: Aufbau des Spacemanagers - eigene Tabelle

2.7.5 Allocation Info File

Das Allocation Info File enthält wie der Spacemanager Informationen zur Anzahl gesamter und freier Blöcke und dient zusätzlich als eine Art Header für das Allocation File, welches selbst nur eine Bitmap-Struktur besitzt [Cugu's Blog 2017]. Es sind Länge, Version und Offset des Allocation Files im Info File gespeichert.
An den Offsets 0x04 und 0x08 findet man je vier Byte Länge und Version des Allocation File. Bei 0x18 und 0x1C stehen erneut die Anzahl gesamter und freier Blöcke. Den Offset zum Allocation File findet man bei 0x20 über vier Byte, was der nächste Block nach dem Allocation Info File ist. Der Aufbau des Allocation Info Files ist in Tabelle 2.7 dargestellt.

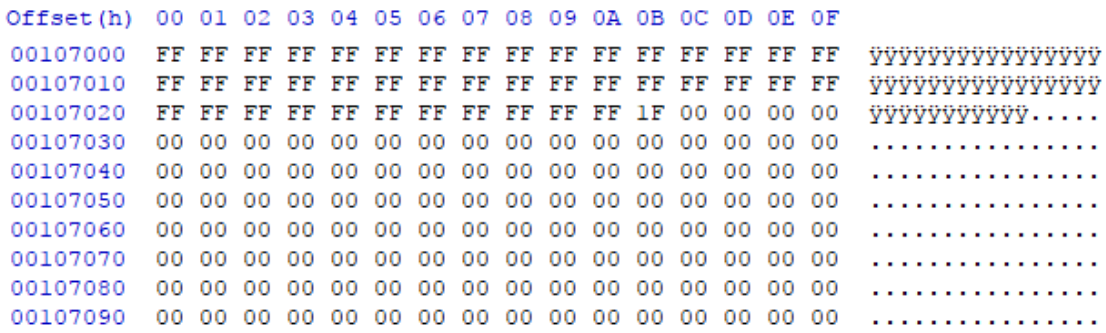
Offset	Länge	Beschreibung	Wert
0x04	4	Länge des Allocation File	
0x08	4	Version des Allocation File	
0x18	4	Gesamtanzahl Blöcke	
0x1C	4	Anzahl freier Blöcke	
0x20	4	Offset zum Allocation File	Block nach dem Info File

Tabelle 2.7: Aufbau des Allocation Info File - eigene Tabelle nach [Cugu's Blog 2017]

2.7.6 Allocation File

Das Allocation File hat eine Bitmap-Struktur und kennzeichnet die belegten Blöcke ab Beginn des Containers. Es kodiert nur die Anzahl belegter Blöcke ab Beginn, nicht deren

Position. Abbildung 2.8 zeigt eine solche Bitmap für eine Partition mit sehr wenigen Verzeichnissen und Dateien.



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
00107000	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00107010	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
00107020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	1F	00	00	00	00	ÿÿÿÿÿÿÿÿÿÿÿÿ....
00107030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00107090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Abbildung 2.8: Allocation File - eigener Screenshot

2.7.7 Volume Superblock

Für jede Partition gibt es einen Volume Superblock. Liegen mehrere mit gleicher Block ID vor, so handelt es sich um verschiedene Versionen. Die aktuellste kann, wie beim Conatiner Superblock, anhand des Block Headers herausgefunden werden. Im Volume Superblock einer Partition sind der Partitionsname, mindestens zwei Zeitstempel und diverse Pointer gespeichert. Die Signatur des Volume Superblocks ist APSB.
Der Volume Superblock beginnt mit der Signatur APSB, gefolgt von einem Zähler, der angibt, zur wievielten Partition des Containers der Volume Superblock gehört. Volume Superblöcke mit dem Wert 0 dieses Zählers gehören zur ersten Partition, mit dem Wert 1 zur zweiten Partition usw. Bis 0x38 folgen leere oder reservierte Bereiche. Zu den folgenden Strukturen ist bisher nur bekannt, dass der Bereich von 0x40 bis 0x5F für alle Versionen aller Volume Superblöcke eines Containers die gleichen Werte besitzt.
An Offset 0x60 steht der Offset zum Block Map B-Tree über acht Byte. Die nächsten acht Byte speichern die Block ID der Root-Directory Node der jeweiligen Partition [Cugu Kaitai]. Es folgen bei 0x70 und 0x78 zwei weitere Node-Offsets, von denen sich der erste jeweils zwischen den verschiedenen Versionen des Volume Superblocks unterscheidet, der zweite in jeder Version konstant bleibt. An Offsets 0x90, 0x98, 0xA0 und 0xC0 findet man vier weitere acht Byte große IDs, Counter oder Versionsnummer, zu denen aktuell nichts bekannt ist.
An Offset 0xD0 ist eine 16 Byte lange Volume GUID gespeichert. Darauf folgt ein erster Zeitstempel. Ab 0x240 steht in einem String-Wert 'newfs apfs (apfs-249.xx.xx)', wobei für xx Zahlen einzusetzen sind. Wofür diese Nummer steht, ist aktuell noch nicht bekannt. Bei einer leeren Partition entfällt dieser String. Bei 0x110 folgt ein zweiter Zeitstempel. Es können weitere Stringwerte und Zeitstempel folgen. An Offset 0x2A0 steht schließlich der Name der Partition. Tabelle 2.8 zeigt die bekannten Strukturen des Volume Superblocks.

Offset	Länge	Beschreibung	Wert
0x00	4	Signatur	'APSB'
0x04	4	Partitionsnummer	0x00 = 1. Partition
0x38	8	unbekannt	variabel zwischen Versionen
0x40	8	unbekannt	konstant in allen Versionen
0x48	24	unbekannt	
0x60	8	Block Map B-Tree	
0x68	8	ID des Root-Verzeichnisses	konstant in allen Versionen
0x70	8	unbekannte Block ID	variabel zwischen Versionen
0x78	8	unbekannte Block ID	konstant in allen Versionen
0x90	8	unbekannt	variabel zwischen Versionen
0x98	8	unbekannt	variabel zwischen Versionen
0xA0	8	unbekannt	variabel zwischen Versionen
0xc0	8	unbekannt	variabel zwischen Versionen
0xD0	16	Volume GUID	
0x0E	8	Zeitstempel letzte Änderung	
0xF0	32	String-Wert	newfs_apfs (apfs-xxx.xxx.xxx)
0x110	8	Zeitstempel Erstelldatum	
0xF0 + 48 x V	32	ggf. weitere String-Werte	
0x110 + 48 x V	8	gg. weitere Zeitstempel	
0x2A0	8	Name der Partition	

Tabelle 2.8: Aufbau des Volume Superblocks - eigene Tabelle

2.7.8 Node

Nodes enthalten entweder feste oder dynamische Einträge. Sie können eigenständig existieren oder als Teil eines B-Trees. Es gibt verschiedene Arten von Nodes, der Typ steht im Block Header an Offset 0x1C. Es gibt laut [Cugu Kaitai] unter anderem History und Location Nodes. Jede Node hat einen 24 Byte großen Header, an den sich die Entry Header direkt anschließen. Nach dem Node Header findet man für jeden Eintrag Pointer zum Node Key und Node Record [Cugu's Blog 2017]. Jeder Eintrag einer Node besteht demzufolge, wie in Abbildung 2.9 gezeigt, aus den drei Teilen Header, Key und Record. Die letzten 40 Byte einer Node gehören nicht zu den Records. Dort steht erneut die Anzahl der Einträge, sowie weitere Angaben, welche noch nicht eindeutig interpretiert werden konnten. Die Keys sowie die Records der Einträge einer Node müssen nicht zwangsläufig die gleiche Reihenfolge haben, wie die Header dazu. An welcher Stelle des Key- bzw. Record-Abschnitts sich die benötigten Daten befinden, wird über die Offsets im Entry Header referenziert.

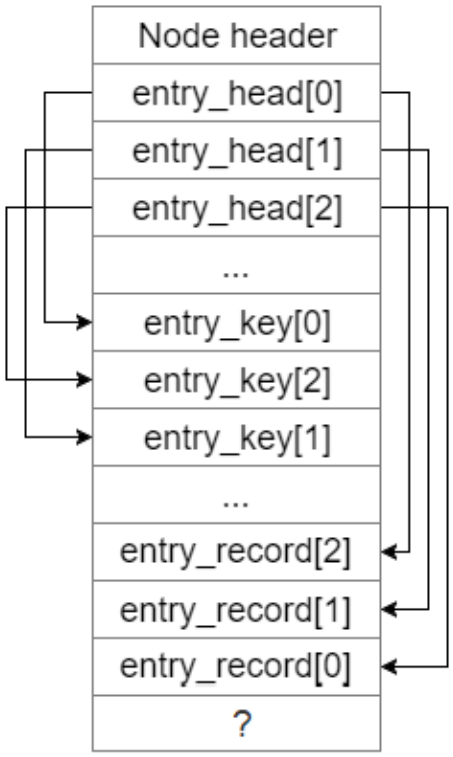


Abbildung 2.9: Aufbau einer Node - Darstellung von [Cugu's Blog 2017]

Node Header

Im Node Header findet man Angaben dazu, ob die Einträge eine feste oder dynamische Größe haben, wie viele Einträge es gibt und wo die Keys und Daten dieser Einträge liegen. Der Aufbau ist in Tabelle 2.9 dargestellt.

Offset	**Länge**	**Beschreibung**	**Wert**
0x00	4	Art der Einträge	0x03 = dynamisch; 0x07 = fest
0x04	4	Anzahl Einträge	
0x0A	2	Offset Beginn Entry Keys	
0x0C	2	Größe aller Entry Keys zusammen	
0x0E	2	Offset Ende Entry Records	
0x10	8	unbekannt	

Tabelle 2.9: Aufbau des Node Headers - eigene Tabelle nach [Cugu Kaitai]

Node Einträge

Node Entries bestehen jeweils aus Header, Key und Record. Dabei sind jeweils alle Entry Header, Entry Keys und Entry Records an einem Stück gespeichert. Die Offsets zu allen Teilen stehen im Node Header und sind als relative Adressen zu verstehen.
Feste Einträge können unter anderem History oder Location Entries sein [Cugu Kaitai]. Entry Header von festen Einträgen sind vier Byte groß und speichern die Offsets von Entry Key und Record. Entry Keys von festen Einträgen sind jeweils 16 Byte groß. In Abbildung 2.10 ist anhand einer Node mit zwei festen History-Einträgen dargestellt, wie man solche Nodes interpretiert. Die Positionen von Block Header, Node Header und Beginn der Entry Header sind mit 0x00, 0x20 und 0x38 vorgegeben. Entry Keys und Records können wie abgebildet über den Node Header gefunden werden.
Dynamische Einträge enthalten verschiedene Informationen, z. B. Datei- oder Verzeichnisnamen und dazugehörige Zeitstempel. Entry Header von dynamischen Einträgen sind acht Byte groß. Jeweils zwei Byte speichern Offset und Größe der Entry Keys sowie Offset und Größe der Daten. Entry Keys beginnen mit vier Byte ID, gefolgt von vier Byte Entry Typ. Dieser kann 0x00000000 für Location, 0x20000000 für Volume, 0x30000000 für Threads, 0x80000000 für Extents oder 0x90000000 für Namenseinträge sein. Des Weiteren kann der Entry Typ 0x40000000 oder 0x60000000 sein. Bei Namenseinträgen folgt auf den Entry Typ die Länge des Namens über zwei Byte und der Name über diese dynamische Länge.
Die Entry Records werden stets über die Offsets im Entry Header gefunden. Diese Offsets beziehen sich jedoch auf das Ende des Record-Abschnitts. Beispielsweise ist der Record-Offset 0x350 so zu verstehen, dass die Daten zum Eintrag an Offset 0xC88 des Allocation Blocks beginnen, da dies das Ergebnis aus folgender Rechnung ist: Blockgröße (0x1000) minus 40 Byte am Ende der Node (0x28) minus Offset Record (0x350) = Beginn Entry Record (0xC88).

2.7.9 B-Tree

B-Trees Blöcke enthalten nur an Offset 0x10 den Offset zur B-Tree Root Node [Cugu's Blog 2017], welche meistens direkt im nächsten Block liegt.

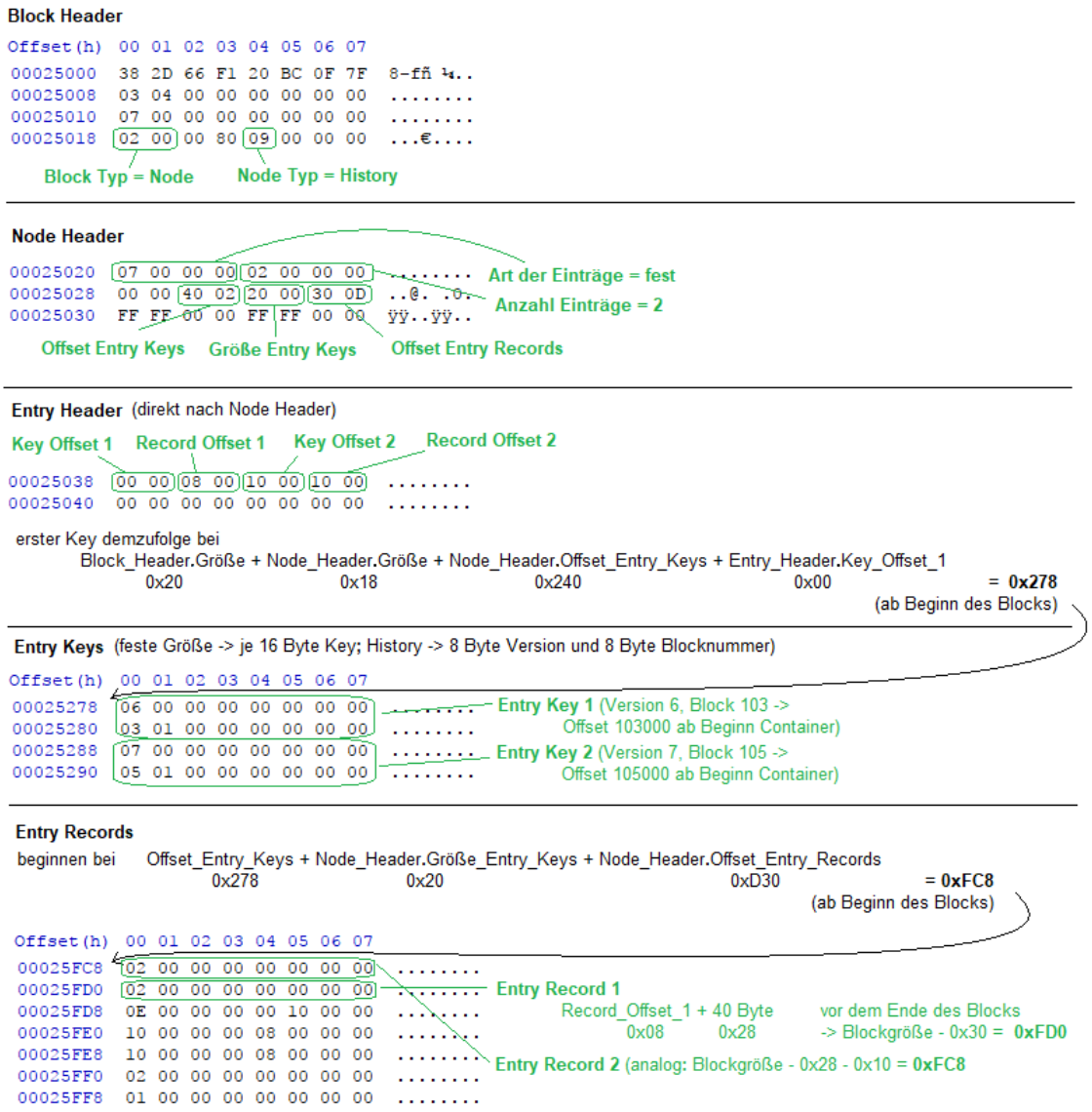


Abbildung 2.10: Node mit zwei festen Einträgen - eigener bearbeiteter Screenshot

3 Methoden

Es ist an einem Testimage gearbeitet worden. Das Image wurde unter macOS „High Sierra“erstellt und ist GPT partitioniert. Es existieren eine Datei Test1.txt sowie ein Ordner TestOrdner im root-Verzeichnis. Desweiteren wurde eine weitere Textdatei geloescht.txt angelegt und wieder gelöscht. In zwei weiteren Versionen des Images wurden zwei Snapshots erstellt sowie eine EPS-Datei hinzugefügt, die aufgrund ihrer Größe von ca. 17 KB mehrere Allocation Blocks belegt. Durch Betrachtung dieses Images im Hex-Editor HxD wurden mögliche Strukturen herausgefiltert und untersucht. Im Folgenden wird dieses Image mit der Nummer 1 bezeichnet.
Im Laufe der Arbeit sind zwei weitere Images zum Vergleich (im Folgenden Nummer 2 und 3) hinzugezogen worden, die unter [Github] frei verfügbar sind. Image 2 hat nur eine Partition ohne durch den Benutzer angelegte Dateien oder Verzeichnisse. Image 3 besitzt drei Partitionen und diverse Dateien und Verzeichnisse. Beide wurden unter macOS 10.12.3 Sierra erstellt.
Abbildung 3.1 zeigt das erste Image im Hex-Editor.

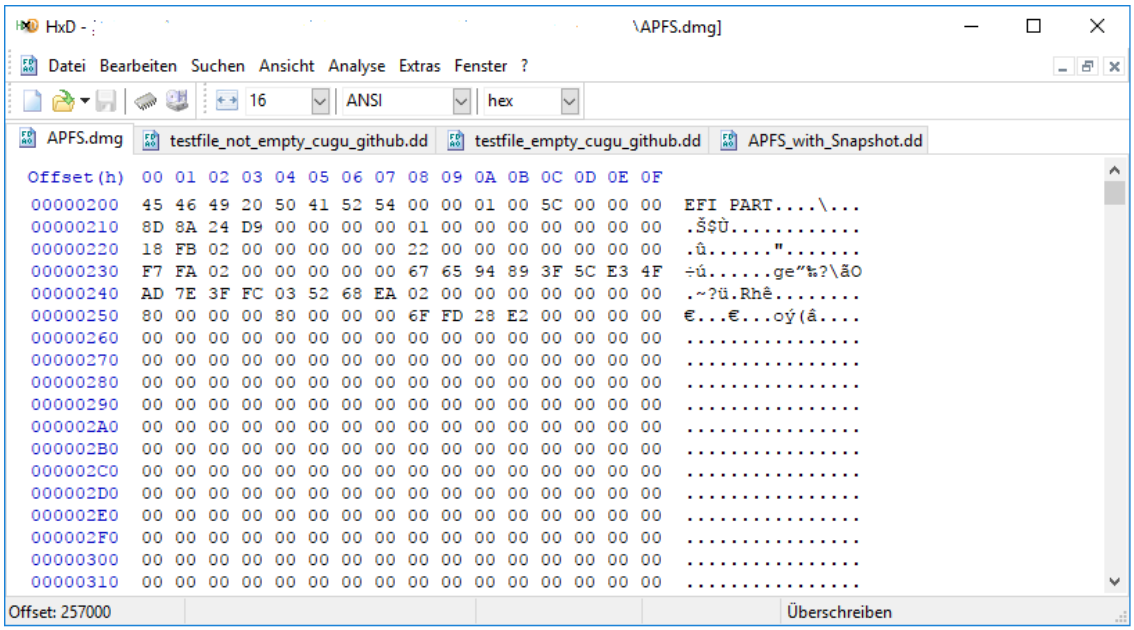


Abbildung 3.1: Image 1 im Hex-Editor HxD - eigener Screenshot

4 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Betrachtung für jedes der drei Images seperat zusammengefasst. Für jedes Image wurden allgemeine Informationen notiert und die Datenträgerstrukturen anhand der Erkenntnisse des Grundlagenteils analysiert. Darüber hinaus festgestellte Zusammenhänge oder Gemeinsamkeiten, die auf allgemeingültige Tatsachen schließen lassen, sind im 4. Abschnitt festgehalten.

4.1 Image 1

Für dieses Image gibt es einen zusätzlichen fünften Abschnitt über Snapshots, der bei den anderen beiden entfällt.

4.1.1 Allgemeines

Das Image ist GPT partitioniert mit folgendem Partitionstyp: EF57347C-0000-AA11-AA11-00306543ECAC. Dieser unterscheidet sich nur durch den ersten Teil von der Kennung für HFS+. Des Weiteren gewinnt man aus dem GPT-Header und dem ersten und einzigen Partitionseintrag folgende Informationen: Der erste nutzbare Sektor ist (standardmäßig) Sektor 34, der letzte 195.319; Die Partition erstreckt sich von logischem Sektor 40 bis 195.319 und ist eine Systempartition mit dem Namen disk image.
Aus der Angabe für den Startsektor und der Standardgröße von 512 Byte für einen Sektor gelangt man zum Offset 0x5000, an welchem die erste und einzige GPT-Partition beginnt. Die Allocation Blocks dieses APFS-Containers sind 4096 Byte groß. Das entspricht acht Sektoren. Am Anfang der Partition findet man fünf Blöcke mit der Kennung NXSB, welche sich an den Offsets 0x5000, 0x7000, 0x9000, 0xB000 und 0xD000 befinden.

4.1.2 Container Superblock

Der aktuellste Containersuperblock hat die Versionsnummer 1C und ist in Abbildung 4.1 dargestellt. Die wesentlichen Informationen sind markiert. Es ist erkennbar, dass der Container aus 24410 Blöcken zu je 4096 Byte besteht, demzufolge 95 MiB groß ist. Die ID des Spacemanagers ist 400. Im Container gibt es genau eine Partition, die an Block 402 beginnt, was den Volume Superblock darstellt.

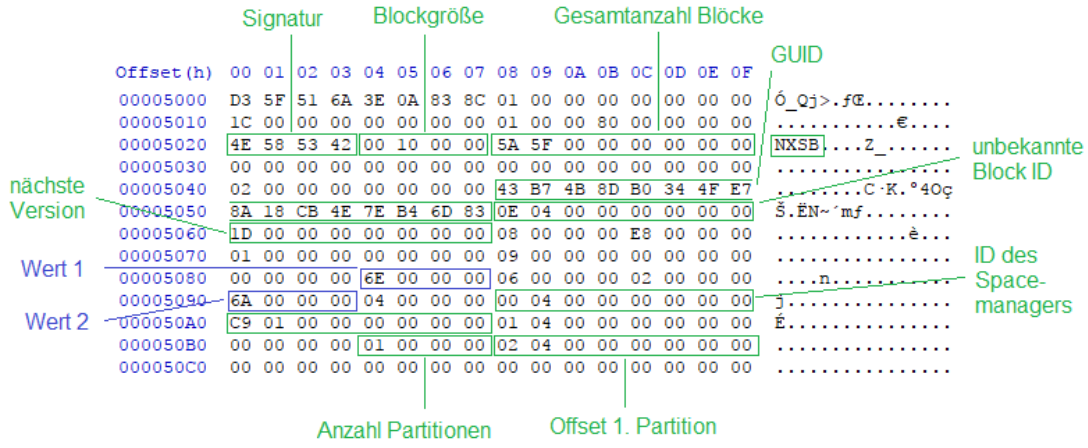


Abbildung 4.1: Image 1: Container Superblock - eigener bearbeiteter Screenshot

4.1.3 Spacemanager, Allocation Info File und Allocation File

Aus Spacemanager, Allocation Info File und Allocation File wird ersichtlich, dass 459 der 24410 Blöcken belegt sind. Die drei aktuellen Strukturen liegen an den Offsets 0x73000, 0x101000 und 0x102000.

4.1.4 Volume Superblock und Datei- und Verzeichniseinträge

Ab Offset 0x10D000 beginnt ein neuer Abschnitt, welcher Angaben zu Dateien und Verzeichnissen beinhaltet. An 14 Stellen findet sich dabei die Signatur APSB für den Volume Superblock. In diesem ist als Name für die Partition *newAPFS* angegeben. Außerdem findet man die Zeichenkette *newfs_apfs (apfs-249.60.20)*. Im aktuellsten Volume Superblock (Version 13) sind neben den zwei obligatorischen Zeitstempeln und eben genannter Zeichenkette jeweils vier weitere Zeitstempel und String-Werte angegeben. Die Zeichenketten sind beinahe gleich und beginnen mit *apfs_kext compiled*. Die Zeitstempel kodieren zum einen den 03.07.2017 und zum anderen den 05.09.2017. Dies sind die Daten, an denen das Image erstellt und letztmalig geändert wurde. Auf einen Volume Superblock folgt meist ein Block B-Tree sowie mindestens ein Node-Block.

4.1.5 Snapshots

Die Untersuchung der zwei Snapshots führte zur Erkenntnis, dass Snapshots intern als Nodes mit zwei Einträgen je Snapshot dargestellt sind. Diese Einträge enthalten den

Namen, zwei im betrachteten Beispiel exakt gleiche Zeitstempel und zwei Verweise auf andere Blöcke. Einer dieser zwei Verweise in einer Snapshot-Node zeigt auf den Volume Superblock, der zur Erstellung des Snapshots aktuell war. Der andere Pointer navigiert zu einer Node mit mehreren Einträgen, die wiederum auf Blöcke zeigen, welche mit 1F 8B 08 00 beginnen oder Dateiinhalt speichern. Außerdem scheint ein Snapshot nur Einträge zu enthalten, die Änderungen seit dem letzen Snapshot darstellen. Es ist jedoch auch möglich, dass ein neuerer Snapshot auf den jeweils vorhergehenden referenziert.

4.2 Image 2

4.2.1 Allgemeines

Das Image hat keinen GPT Header und beginnt offensichtlich direkt am Anfang der Partition. Die Blöcke sind auch hier 4096 Byte groß. Es existieren drei Versionen des Container Superblocks an den Offsets 0x0000, 0x2000 und 0x4000.

4.2.2 Container Superblock

Die aktuellste Version des Container Superblocks ist 2. Der Container ist lediglich 2560 Allocation Blocks groß, was 10 MiB entspricht. Der Block Map Block wird mit 0x53 angegeben. Die ID des Spacemanagers ist ebenfalls 400. Im Container gibt es genau eine Partition, die an Block 0x402 beginnt.

4.2.3 Spacemanager, Allocation Info File und Allocation File

Es gibt zwei Spacemanager, der aktuellste hat die Versionsnummer 2 und befindet sich an Offset 0xB000. Man findet darin folgende Angaben: von 2560 Allocation Blocks sind 2467 frei, demzufolge sind 93 Allocation Blocks belegt. Diese Angaben werden auch durch die jeweils aktuelle Version von Allocation Info File und Allocation File bestätigt, welche an Offsets 0x4F000 und 0x50000 beginnen.

4.2.4 Volume Superblock und Datei- und Verzeichniseinträge

Ab Offset 0x59000 findet man die Node zum root-Verzeichnis und darauf folgend genau einen Volume Superblock. In den Node-Einträgen zum Wurzelverzeichnis findet man Zeitstempel vom 01.06.2017. Die beiden obligatorischen Zeitstempel des Volume Superblocks unterscheiden sich nur im Nanosekundenbereich von dem des Root

Directories. Der String *newfs_apfs* ist nicht vorhanden. Es gibt auch keine weiteren Informationen und Zeitstempel in diesem Volume Superblock.

4.3 Image 3

4.3.1 Allgemeines

Das Image besitzt, wie Image 2, keine Angaben zur Partitionierung und beginnt am Anfang der ersten Partition. Die Blöcke sind ebenfalls 4096 Byte groß. Die Signatur NXSB findet sich hier an neun Stellen (0x0000, 0x2000, 0x4000, 0x6000, 0x8000, 0xA000, 0xC000, 0xE000 und 0x010000).

4.3.2 Container Superblock

Der aktuellste Container Superblock hat die Versionsnummer 8. Dieser Container ist 262.390 Allocation Blocks, also etwas mehr als 1 GiB, groß. Der Block, an dem sich die Block Map befindet, ist 0x57A. Die ID des Spacemanagers hat auch hier den Wert 400. Es existieren drei Volumes, welche an den Blöcken 0x402, 0x406 und 0x408 beginnen.

4.3.3 Spacemanager, Allocation Info File und Allocation File

Die aktuellste Version des Spacemanagers ist die Nummer 8. Von den 262.390 Allocation Blocks des Containers sind 260986 frei, demzufolge 1404 bereits belegt. Allocation Info File und Allocation File der Version 8 beginnen an den Offsets 0x513000 und 0x514000 und enthalten die gleichen Angaben zum Belegungsstatus des Containers.

4.3.4 Volume Superblock und Datei- und Verzeichniseinträge

Der Abschnitt mit den Verzeichnis- und Dateiinformationen beginnt bei Image 3 an Offset 0x529000; die Signatur APSB gibt es elf Mal. Jeweils vier der elf Volume Superblöcke gehören zu Partition 1 und 2, die restlichen drei zur dritten Partition. Die Partitionen heißen *myvol1*, *myvol2* und *thirdvolume* und liegen in den Versionen 6, 7 und 8 vor. Die Block Maps liegen an den Blöcken 565, 56E und 577. Die IDs der Root Directories sind 404, 407 und 409. Die Zeitstempel für Erstellzeit und letzte Änderung in den drei Volume Superblöcken stammen alle vom 05.04.2017 und liegen lediglich wenige Sekunden auseinander. Der String *newfs_apfs (apfs-249.30.8)* befindet sich in jedem der Volume Superblöcke.

4.4 Allgemeine Erkenntnisse

In jeder Partition ist eine 32 Zeichen lange ID für die jeweilige Partition abgelegt. Diese liegt meistens in einem Block direkt oder kurz nach dem ersten Volume Superblock.
Einige Block IDs sind bestimmten Strukturen fest zugewiesen. So haben die untersuchten Root Nodes stets die ID 404, die Volume Superblöcke der ersten Partition die 402 und die Spacemanager die ID 400. Auch die gefundenen Blocknummern 406 und 408 für die zweite und dritte Partition von Image 3 könnten festgelegte Standardwerte sein.
Jeweils alle Strukturen, die zusammen gehören, haben die gleiche Versionsnummer. Die aktuellste Version aller Datenträgerstrukturen sollte also in Container Superblock, Checkpoints, Block Map, Allocation Info File, Spacemanager und Root Node jeweils den selben Wert haben.
Bei einigen Datenträgerstrukturen ist die Anzahl der vorliegenden Versionen offenbar begrenzt. Diese kann zwischen verschiedenen Systemen oder Containern zwar verschieden sein, jedoch werden innerhalb eines Containers die Datenträgerstrukturen ab einer bestimmten Anzahl mit der neueren Version überschrieben. Dies gilt für Container Superblock, Allocation Info File und Allocation File. Volume Superblock, Spacemanager und beispielsweise die Root Node werden beibehalten und weitere Versionen in die folgenden freien Blöcke geschrieben. Möglicherweise wurden bei diesen Strukturen die Grenzen im Zuge dieser Analyse noch nicht erreicht.

5 Diskussion

5.1 Über diese Arbeit

Diese Arbeit enthält eine erste Strukturanalyse des Apple File Systems sowie einige theoretische Betrachtungen zu den neuen Features. Da APFS nach nur vier Jahren Entwicklung als Entwicklervorschau auf den Markt gekommen ist, kann gesagt werden, dass das Dateisystem zwar stabil läuft, jedoch immer noch in einer Art Beta-Testphase steckt. Dadurch kann es vorkommen, dass bestimmte Strukturen oder Eigenschaften noch hinzugefügt, verändert oder entfernt werden. Das führt dazu, dass es im Rahmen dieser Arbeit teilweise nicht möglich gewesen ist, bestimmte Features zu testen. Unter anderem war es unter macOS „Sierra“in den Versionen 10.12.0 bis 10.12.5 möglich APFS-Images zu erstellen, mit dem Update auf 10.12.6 entfiel diese Funktionalität. Images können dann nur noch gelöscht oder verändert werden. Des Weiteren war es in Betriebssystem-Version 10.12.6 nicht möglich ein Volume zu verschlüsseln oder ein verschlüsseltes Volume zu einem bestehenden Container hinzuzufügen. Lediglich der DiskUtility-Befehl decrpytVolume war verfügbar. Nach Bewertung der Autorin werden die Funktionen zur Verschlüsselung jedoch bald wieder vollumfänglich verfügbar sein, da Vertraulichkeit der User-Daten durch Verschlüsselung eines der zentralen Features bei APFS ist und Sicherheit und Datenschutz zu einem der wesentlichen Grundwerte von Apple Inc. gehört.
Bei der Datenträgeranalyse konnten viele Strukturen gefunden und interpretiert werden. Die Grundmerkmale der einzelnen Strukturen sind im Wesentlichen gut erfasst worden, es besteht jedoch Klärungsbedarf bei einzelnen Feldern bezüglich Größe, Bedeutung oder Zuordnung in einen Kontext. Beispielsweise kann es passieren, dass nur geklärt ist, ob es sich um eine ID oder einen Counter handelt, aber nicht worauf sich der Wert bezieht. Ein Beispiel dafür findet man an Offset 0x70 und 0x78 des Volume Superblocks. Dort sind Block IDs gespeichert, die auf Nodes verweisen, jedoch ist bisher nicht bekannt, um was für Nodes es sich dabei handelt. Teilweise ist es bisher sogar nur möglich zu sagen, ob sich der Wert in verschiedenen Versionen einer Struktur ändert oder nicht, jedoch nicht, was er bedeutet. Ein Beispiel dafür findet man ebenfalls im Volume Superblock, dieses Mal an den Offsets 0x90 und 0x98. Dort befinden sich Werte, die sich zwischen den einzelnen Versionen des Volume Superblocks unterscheiden, jedoch noch nicht interpretiert wurden.
Aufgrund der geringen Menge an Daten, die in beschriebenen Testimages anfallen, konnten diejenigen Features, die zu höheren Speicher- und Zugriffsgeschwindigkeiten sowie effizienter Speichernutzung führen sollen, nicht eindeutig nachvollzogen werden. Das kann umfassend erst in einem realen System mit komplexer Verzeichnisstruktur, verschiedener Anwendungssoftware und vielen Schreibvorgägngen intensiv getestet werden. Erste Tests zur Performance von APFS unter iOS 10.3 wurden bei [Golem 2017] bereits durchgeführt mit dem Ergebnis, dass sich die Geschwindigkeiten nicht

erheblich verbessern und auch stark vom Zustand der zugrunde liegenden Hardware abhängig ist. Ein Test von APFS unter macOS kann dabei durchaus zu anderen Ergebnissen kommen, da die Speichergröße und die Komplexität bei großen PCs deutlich größer ist.

5.2 Erfüllung der Erwartungen an das neue Dateisystem

Neben der bereits angesprochenen und theoretisch betrachteten Verschlüsselung, gibt es laut Herstellerangaben noch einige weitere interessante Features. APFS soll zum einen den zur Verfügung stehenden Speicherplatz schnell und effizient nutzen und zum anderen größtmögliche Sicherheit bieten und die Daten der Nutzer vor Abstürzen oder Fremdzugriffen schützen.
Copy-on-Write und die damit verbundenen Klone führen in der Theorie zu weniger Schreibvorgängen. Dadurch bedingt wird die Kapazität der Speichermedien besser ausgenutzt und die SSDs nutzen sich langsamer ab. In der Praxis bewirkt eine Veränderung an einem Block einer Datei oft aber auch die Veränderung aller darauf folgenden. Bei Löschung von nur einem Zeichen im ersten Block verändert sich nicht nur dieser, sondern alle anderen Blöcke ebenfalls. Das führt dazu, dass alle Blöcke der Datei neu geschrieben werden müssen. Dieses Verhalten war bei Veränderung von Nutzerdaten auf einem fast leeren Plattenimage zu beobachten. Die Einsparung an Schreibvorgängen und tatsächlich genutztem physikalischen Speicher ist somit aus Sicht der Autorin als mäßig bis gering einzustufen.
Schnelle Schreib- und Lesezugriffe werden nicht durch das Dateisystem alleine, sondern auch durch die zur Verfügung stehende Hardware realisiert. Da APFS für Flashspeicher optimiert ist und zukünftig wahrscheinlich hauptsächlich auf diesen laufen wird, ist es ohnehin schon schneller als beispielsweise HFS+ auf einer magnetischen Festplatte. Auf der Seite der Software sind die Veränderungen aber auch eher gering. Die verwendeten B-Trees kommen schon bei HFS+ zum Einsatz und besitzen eine logarithmische Komplexität. Demzufolge ist die Dauer für Such-, Lese- und Schreibzugriffe auch bei zunehmender Knotenanzahl im Baum relativ gering, sollte sich theoretisch jedoch nicht wesentlich von der bei HFS+ unterscheiden. Auch die unter 5.1 angesprochenen Tests auf iOS 10.3 zeigen keine deutlich schnelleren Systeme. Aufgrund fehlender weiterer Erkenntnisse zu dieser Thematik, ist es zum aktuellen Zeitpunkt noch fragwürdig, ob und in welchem Umfang durch das Apple File System tatsächlich Zeit gespart wird.
Die Sicherheit soll unter anderem durch Verschlüsselung, Copy-on-Write und Snapshots gegeben werden. Da die Verschlüsselung im Rahmen dieser Arbeit nicht praktisch getestet werden konnte und auch Copy-on-Write als Schutz vor einem Plattenabsturz nicht überprüfbar war, soll an dieser Stelle auf die Snapshots als effiziente Art der Erstellung von Backups eingegangen werden. Bisher ist die Erstellung von Snapshots nur

über das Terminal möglich, funktioniert dort aber einwandfrei. Um diese Funktion für die breite Masse an Nutzern attraktiv zu machen, sollte jedoch an einer Möglichkeit gearbeitet werden, Schnappschüsse über ein graphisches Menü zu erstellen oder automatisiert erstellen zu lassen zu bestimmten Zeitpunkten. Bei der Analyse der Images wurde festgestellt, dass zu jedem Snapshot zwei Zeitstempel abgelegt sind. Das wirft in Anbetracht der Tatsache, dass Schnappschüsse nicht veränderbar sind, die Frage auf, wofür beide Zeitstempel stehen. Da man das System auf den im Snapshot gespeicherten Zustand zurücksetzen kann, steht eine der Zeiten möglicherweise für den letzten Zeitpunkt, an dem der Snapshot gemounted war. Einer der Zeitstempel ist wahrscheinlich das Erstelldatum, der andere zeigt möglicherweise an, wann das System zum letzten Mal auf diesem Stand war. Bestätigt werden konnte diese Vermutung noch nicht, da beide Zeitstempel im betrachteten Beispiel gleich waren.
Abschließend wird zur Umsetzung des APFS-Konzepts festgehalten, dass Apple seinen Ankündigungen bisher nur teilweise gerecht werden kann. In der Praxis ist das Verhalten der einzelnen Neuerungen entweder abweichend von der Theorie oder noch nicht vollständig implementiert. Befehle für Verschlüsselung existieren, jedoch können nicht alle davon unter macOS 10.12.6 ausgeführt werden. Klone und das Copy-on-Write-Verfahren sind theoretisch nützlich und können teilweise auch in der Praxis überzeugen. Das ist jeweils vom vorliegenden Szenario abhängig. Werden beispielsweise mehrere exakte Kopien an verschiedenen Stellen abgelegt, funktioniert das Prinzip einwandfrei. Werden einzelne Klone verändert, kommt es darauf an, welche Stelle(n) und welche Blocks davon betroffen sind. Auch die Snapshots funktionieren nach aktuellen Erkenntnissen gut, lediglich eine automatisch einstellbare Implementierung von Seiten des Herstellers ist wünschenswert in Bezug zum eigentlich geplanten Zweck der Datensicherung. Die flexible Partitionierung und die nur kurz betrachteten Features Space Sharing und Fast Directory Sizing sind ebenfalls als positive Entwicklung zu betrachten, auch wenn die Auswirkungen dessen ebenfalls erst in einem System größeren Umfangs sichtbar werden.

5.3 Bewertung der Ergebnisse für die Forensik

Um die Frage zu diskutieren, ob das Apple File System die computerforensischen Untersuchungen an Apple-Geräten begünstigen oder erschweren wird, werden die einzelnen Features vorerst getrennt analysiert.
Ist die Verschlüsselung von Dateien und Partitionen erst einmal endgültig implementiert, wird das die forensische Arbeit behindern. Verschlüsselte Festplatten oder Teile davon stellen in der Computerforensik stets eine Herausforderung dar. Die verwendete bzw. geplante Verschlüsselung mittels AES ist außerdem als so sicher einzustufen, dass sie mit realem Zeitaufwand bisher nicht zu brechen ist. Um Daten aus verschlüsselten Volumes zu gewinnen, ist es demzufolge notwendig, das admin-Passwort in Erfahrung zu bringen oder das System in einem Zustand zu finden, in dem die verschlüsselten Teile gemountet und entschlüsselt sind. Problematisch kann es dann noch werden, wenn

einzelne Dateien seperat mit einem anderen Schlüssel verschlüsselt sind. Auch da gilt, dass ein Brechen der Verschlüsselung relativ aussichtslos ist, sofern man den Schlüssel nicht kennt.
Snapshots können computerforensische Auswertungen vereinfachen und sinnvoll eingesetzt werden, um ältere Dateiversionen zu finden oder Änderungen an den Dateien zeitlich nachvollziehen zu können. Inwieweit dieses Potential tatsächlich genutzt werden kann, ist momentan stark davon abhängig, wie internsiv der Anwender dieses Feature nutzt. Durch regelmäßige Backups können jedoch Änderungen am System relativ gut zurückverfolgt werden. Neben den Snapshots sind aber auch in den Nodes jeweils mehrere Zeitstempel abgelegt, die im Vergleich zu HFS+ 1.000.000.000 Mal genauer sind. Beispielsweise können sich in den Volume Superblöcken neben Zeiten für das Erstellen und die letzte Änderung der Partition weitere Zeitstempel finden. Wofür diese genau stehen, muss noch getestet werden.
Ebenfalls nicht abschließend geklärt werden konnte, nach welchen Kriterien alte Versionen der verschiedenen Datenträgerstrukturen überschrieben werden. Es scheint jedoch bei einigen Strukturen Begrenzungen in der Art zu geben, dass nur eine bestimmte Anzahl verschiedener Versionen gleichzeitig vorliegen kann. So gab es beim untersuchten Image1 stets nur fünf Versionen des Container Superblocks zur gleichen Zeit. Die Unterschiede zwischen den einzelnen Versionen liegen jedoch nur in den verschiedenen Pointern, z. B. auf die Block Map oder bei der Anzahl und Position der Volumes, wenn neue Partitionen hinzugefügt oder alte gelöscht werden. Bei gelöschten Dateien liegen die Inhalte auch nach dem Löschvorgang vor, wenn man den Datenträger als Image im Hex-Editor betrachtet. Jedoch gibt es keine Verweise mehr auf diese Blöcke in aktuellen Versionen der Datenträgerstrukturen und in Snapshots, die nach dem Löschen entstanden sind. Das bedeutet, dass gelöschte Dateien vom Dateisystem zwar nicht mehr angezeigt werden, sie aber beispielsweise durch Carving oder gezielte Suchbegriffe gefunden werden könnten. Das ist jedoch nicht nur bei APFS sondern den meisten Dateisystemen so und wird deswegen keine neuen Auswirkungen auf die forensische Arbeit an Apple-Computern haben.
Die weiteren Neuerungen bei APFS sind aus Anwendersicht interessant, wenn dadurch tatsächlich Speicherplatz und Zeit eingespart werden kann, die Partitionen flexibel wachsen und man diese nicht im Voraus exakt planen muss. Copy-on-Write und die Klone können Platz sparen und das Dateisystem intern übersichtlicher machen. Durch die Platzersparnis ist es eventuell möglich sehr viele verschiedene Versionen einer Datei auf dem System ablegen zu können und , ein sonstiger Nutzen oder Schaden bezüglich forensischer Auswertungen konnte nicht festgestellt werden. Ähnlich verhält es sich bei Space Sharing und Fast Directory Sizing. Zwar gibt es somit keine festen Positionen, an denen nach bestimmten Datenträgerstrukturen gezielt gesucht werden kann. Allerdings scheint es für Spacemanager, Volume Superblock und Root Node festgelegte IDs zu geben, über welche diese wichtigen Strukturen gefunden werden können.

5.4 Einordnung des Dateisystems

Der Vergleich mit anderen aktuell verwendeten Dateisystemen soll zeigen, wie Sicherheit und Schnelligkeit von APFS einzuordnen sind.

5.4.1 Entwicklung gegenüber HFS+

Gegenüber HFS+, welches unter macOS bisher verwendet wurde, sind kleine Fortschritte erkennbar. Beide Dateisysteme beruhen auf B-Trees, Inodes und Extents als wesentliche Strukturen für die Datenablage und Suche von Dateien. Demzufolge sind von Seiten dieser Implementierung keine großen Unterschiede zu vermerken. Die allokierten Einheiten sind ebenfalls gleich groß. Jedoch unterstützt APFS sowohl größere Partitionen als auch Dateien, was als Anpassung an die immer größer werdenden Speichermedien gesehen werden kann. Beide Dateisysteme haben einige feste Strukturen, wie das Allocation File und den Volume Header bei HFS+ respektive den Volume Superblock bei APFS. Unterschiedlich dabei ist jedoch, dass der Volume Header bei HFS+ eine feste Position hat, der Volume Superblock jedoch nicht. Dies ist wahrscheinlich aufgrund der Container-Struktur und flexiblen Partitionierung bei APFS so gelöst worden. Diese Art der Partitionsverwaltung ist wiederum ein Fortschritt gegenüber HFS+, da nicht neu partitioniert werden muss, wenn sich die Größen der Volumes ändern oder man beispielsweise eine neue Partition hinzufügen möchte. Inwieweit eventuelle Verbesserungen in der Geschwindigkeit tatsächlich vom Dateisystem ausgehen, wurde in dieser Arbeit nicht genauer betrachet, da hauptsächlich die Strukturanalyse und Forensik im Mittelpunkt der Betrachtungen stand. Ein schnelles und gut organisiertes System hat aber definitiv Vorteile, wenn beispielsweise im Rahmen von Ermittlungen nach festgelegten Begriffen gesucht werden soll.
Auch wenn die Verschlüsselung von Volumes und Dateien für die forensische Untersuchung von Computern immer eine große Herausforderung darstellt, ist die standarmäßige Implementierung der Dateiverschlüsselung aus Sicht der Anwender und der IT-Sicherheit eine erhebliche Verbesserung. Wenn der Benutzer nicht mehr den Umweg über File Vault FileVault gehen muss, werden zukünftig möglicherweise mehr Volumes verschlüsselt vorgefunden werden. Die Algorithmen hinter FileVault und der Verschlüsselung bei APFS sind jedoch gleich. Nach dem aktuellen Stand sind bei verschlüsselten Daten also keine großen Veränderungen im Vergleich zu HFS+ zu erwarten. Durch standarmäßige native Verschlüsselung ist es zwar einfacher, das Schutzziel der IT-Sicherheit der Vertraulichkeit zu erreichen. Da aber auch Ermittler theoretisch als Dritte in diesem Sinne gelten, wird es noch wichtiger, alle benötigten Passwörter zu Beginn der Ermittlungen in Erfahrung zu bringen.
Durch Klone und Copy-on-Write wird vor allem das Schutzziel der Verfügbarkeit angestrebt. Neben den Einsparungen an Speicherplatz soll Copy-on-Write auch vor Datenverlusten bei Plattenabstürzen vorbeugen. Solche Mechanismen gibt es bei HFS+ noch

nicht, allerdings hält ein Journal alle Änderungen am Dateisystem fest. Im Vergleich dazu ist bei APFS ebenfalls ein Fortschritt erzielt worden.
Die letzte wichtige Neuerung bei APFS liegt in den Snapshots. Sie ermöglichen eine einfache, schnelle und effiziente Art der Backuperstellung. So wird in wenigen Sekunden ein Abbild der jeweiligen Partition erstellt, welches den Zustand zu diesem Zeitpunkt festhält. Bei HFS+ sind solche Datensicherungen über TimeMachine möglich. Momentan ist es zwar lediglich über Terminal-Befehle möglich, solche Snapshots zu erstellen und das System auf so auf einen älteren Stand zurückzusetzen, jedoch wird angenommen, dass es in zukünftigen Versionen von APFS eine Implementierung von Snapshots über ein Menü geben wird. Schnappschüsse können damit die Notwendigkeit von TimeMachine ersetzen.

5.4.2 Parallelen zu BTRFS

BTRFS ist ein seit 2007 für Linux entwickeltes Dateisystem. Dieses zeigt ähnliche Eigenschaften wie APFS, vor allem bezüglich der Dateigrößen, B-Trees, Copy-on-Write-Verfahren und Schnappschüsse. Zusätzlich werden dort jedoch auch Prüfsummen für Daten und Metadaten unterstützt, bei APFS gibt es solche Checksummen nur in den Datenträgerstrukturen für die Metadaten. Fragwürdig ist, warum Apple bei seinem neuen Dateisystem nicht auch Prüfsummen für Nutzerdaten unterstützt, weil durch diese das Schutzziel der Datenintegrität erreicht werden kann. Alles in Allem ist BTRFS dem Apple File System recht ähnlich, aber möglicherweise auch schon ein klein wenig weiter, was Datensicherheit und Effizienz betrifft.

5.5 Fazit und Ausblick

Es sollte untersucht werden, wie Dateien gespeichert werden und wo relevante Daten und Metadaten liegen. Dateien können ähnlich wie beim Vorgänger HFS+ über B-Trees gefunden werden. In den Nodes finden sich auch Metadaten, wie beispielsweise Zeitstempel, Namen und Versionen. Desweiteren ist interessant, wie man auf gelöschte oder veränderte Dateien zugreifen kann und inwieweit Veränderungen zeitlich nachvollzogen werden können. Wie lange gelöschte Dateien noch vorliegen und gefunden werden können oder Änderungen verfolgt werden können, ist bisher nicht eindeutig zu bestimmen. Durch Schnappschüsse ist es prinzipiell aber möglich, solche Änderungen zu sehen und das System gegebenenfalls auf einen älteren Zustand zurückzusetzen. Der wichtigste Punkt ist jedoch die Frage nach Nutzen und Schaden für die forensische Arbeit. Geht es beispielweise darum, das Dateisystem nach Begriffen zu durchsuchen, kommt dem Ermittler das Apple File System dahingehend entgegen, dass Dateien nicht redundant vorliegen und die Daten effizient gespeichert sind. Außerdem helfen

Schnappschüsse dem Forensiker wie bereits erwähnt. Eine Herausforderung könnte die Verschlüsselung darstellen, die zukünftig standardmäßig implementiert sein wird. Da die Entwicklung von APFS noch nicht komplett abgeschlossen ist, kann es in näherer Zukunft passieren, dass verschiedene Features abweichend von der bisherigen Planung hinzugefügt, entfernt oder verändert werden. Aufgrund dessen und um Lücken in der bisherigen Analyse zu füllen, bietet die Untersuchung von APFS in nächster Zeit noch viele Möglichkeiten. Interessant wäre beispielsweise auch die Arbeit an einem Fallbeispiel und die Implementierung von APFS in großen Forensik-Suites wie EnCase oder XWay Forensics.

Kaitai Struct Struct File

```
meta:
  id: apfs
  license: MIT
  encoding: UTF-8
  endian: le

seq:
  - id: block0
    type: block
    size: 4096

instances:
  block_size:
    value: _root.block0.body.as<containersuperblock>.block_size
#  random_block:
#    pos: 0 * block_size   # enter block number here to jump direct-
                           ly that block in the WebIDE
#    type: block           # opens a sub stream for making position-
                           ing inside the block work
#    size: block_size

types:

# block navigation

  ref_block:
    doc: |
      Universal type to address a block: it both parses one u8-sized
      block address and provides a lazy instance to parse that block
      right away.
    seq:
      - id: value
        type: u8
    instances:
      target:
        io: _root._io
        pos: value * _root.block_size
        type: block
        size: _root.block_size
    -webide-representation: 'Blk {value:dec}'
```

```
# meta structs

  block_header:
    seq:
      - id: checksum
        type: u8
        doc: Flechters checksum, according to the docs.
      - id: block_id
        type: u8
        doc: ID of the block itself. Either the position of the block
        or an incrementing number starting at 1024.
      - id: version
        type: u8
        doc: Incrementing number of the version of the block
        (highest == latest)
      - id: type_block
        type: u2
        enum: block_type
      - id: flags
        type: u2
        doc: 0x4000 block_id = position, 0x8000 = container
      - id: type_content
        type: u2
        enum: content_type
      - id: padding
        type: u2

  block:
    seq:
      - id: header
        type: block_header
      - id: body
        #size-eos: true
        type:
          switch-on: header.type_block
          cases:
            block_type::containersuperblock: containersuperblock
            block_type::rootnode: node
            block_type::node: node
            block_type::spaceman: spaceman
            block_type::allocationinfofile: allocationinfofile
            block_type::btree: btree
            block_type::checkpoint: checkpoint
```

```
            block_type::volumesuperblock: volumesuperblock


# containersuperblock (type: 0x01)

  containersuperblock:
    seq:
      - id: magic
        size: 4
        contents: [NXSB]
      - id: block_size
        type: u4
      - id: num_blocks
        type: u8
      - id: padding
        size: 16
      - id: unknown_64
        type: u8
      - id: guid
        size: 16
      - id: next_free_block_id
        type: u8
      - id: next_version
        type: u8
      - id: unknown_104
        size: 32
      - id: previous_containersuperblock_block
        type: u4
      - id: unknown_140
        size: 12
      - id: spaceman_id
        type: u8
      - id: block_map_block
        type: ref_block
      - id: unknown_168_id
        type: u8
      - id: padding2
        type: u4
      - id: num_volumesuperblock_ids
        type: u4
      - id: volumesuperblock_ids
        type: u8
        repeat: expr
```

```
        repeat-expr: num_volumesuperblock_ids

# node (type: 0x02)

  node:
    seq:
      - id: type_flags
        type: u2
      - id: leaf_distance
        type: u2
        doc: Zero for leaf nodes, > 0 for branch nodes
      - id: num_entries
        type: u4
      - id: unknown_40
        type: u2
      - id: ofs_keys
        type: u2
      - id: len_keys
        type: u2
      - id: ofs_data
        type: u2
      - id: meta_entry
        type: full_entry_header
      - id: entries
        type: node_entry
        repeat: expr
        repeat-expr: num_entries

  full_entry_header:
    seq:
      - id: ofs_key
        type: s2
      - id: len_key
        type: u2
      - id: ofs_data
        type: s2
      - id: len_data
        type: u2

  dynamic_entry_header:
    seq:
      - id: ofs_key
        type: s2
```

```
      - id: len_key
        type: u2
        if: (_parent._parent.type_flags & 4) == 0
      - id: ofs_data
        type: s2
      - id: len_data
        type: u2
        if: (_parent._parent.type_flags & 4) == 0

## node entries

  node_entry:
    seq:
      - id: header
        type: dynamic_entry_header
    instances:
      key:
        pos: header.ofs_key + _parent.ofs_keys + 56
        type: key
        -webide-parse-mode: eager
      data:
        pos: _root.block_size - header.ofs_data - 40 * (_parent.type_
        flags & 1)
        type:
          switch-on: '(((_parent.type_flags & 2) == 0) ? 256 : 0) +
          key.type_entry.to_i * (((_parent.type_flags & 2) == 0) ? 0
          : 1)'
          cases:
            256: pointer_record # applies to all pointer records, i.e.
            any entry data in index nodes
            entry_type::location.to_i: location_record
            entry_type::inode.to_i: inode_record
            entry_type::name.to_i: named_record
            entry_type::thread.to_i: thread_record
            entry_type::hardlink.to_i: hardlink_record
            entry_type::entry6.to_i: t6_record
            entry_type::extent.to_i: extent_record
            entry_type::entry12.to_i: t12_record
            entry_type::extattr.to_i: extattr_record
        -webide-parse-mode: eager
    -webide-representation: '{key}: {data}'

## node entry keys
```

```
key:
  seq:
    - id: key_low # this is a work-around for JavaScript's inabili-
    ty to hande 64 bit values
      type: u4
    - id: key_high
      type: u4
    - id: content
      #size: _parent.header.len_key-8
      type:
        switch-on: type_entry
        cases:
          entry_type::location: location_key
          entry_type::inode: inode_key
          entry_type::name: named_key
          entry_type::hardlink: hardlink_key
          entry_type::extattr: named_key
          entry_type::extent: extent_key
  instances:
    key_value:
      value: key_low + ((key_high & 0x0FFFFFFF) << 32)
      -webide-parse-mode: eager
    type_entry:
      value: key_high >> 28
      enum: entry_type
      -webide-parse-mode: eager
  -webide-representation: '({type_entry}) {key_value:dec} {content}'

location_key:
  seq:
    - id: block_id
      type: u8
    - id: version
      type: u8
  -webide-representation: 'ID {block_id:dec} v{version:dec}'

history_key:
  seq:
    - id: version
      type: u8
    - id: block_num
      type: ref_block
```

```
    -webide-representation: '{block_num} v{version:dec}'

  inode_key:
    seq:
      - id: block_num
        type: ref_block
    -webide-representation: '{block_num}'

  named_key:
    seq:
      - id: len_name
        type: u1
      - id: flag_1
        type: u1
      - id: unknown_2
        type: u2
        if: flag_1 != 0
      - id: dirname
        size: len_name
        type: strz
    -webide-representation: '"{dirname}"'

  hardlink_key:
    seq:
      - id: id2
        type: u8
    -webide-representation: '#{id2:dec}'

  extent_key:
    seq:
      - id: offset # seek pos in file
        type: u8
    -webide-representation: '{offset:dec}'

## node entry records

  pointer_record: # for any index nodes
    seq:
      - id: pointer
        type: u8
    -webide-representation: '-> {pointer:dec}'

  history_record: # ???
```

```
    seq:
      - id: unknown_0
        type: u4
      - id: unknown_4
        type: u4
    -webide-representation: '{unknown_0}, {unknown_4}'

  location_record: # 0x00
    seq:
      - id: block_start
        type: u4
      - id: block_length
        type: u4
      - id: block_num
        type: ref_block
    -webide-representation: '{block_num}, from {block_start:dec}, len
    {block_length:dec}'

  thread_record: # 0x30
    seq:
      - id: parent_id
        type: u8
      - id: node_id
        type: u8
      - id: creation_timestamp
        type: u8
      - id: modified_timestamp
        type: u8
      - id: changed_timestamp
        type: u8
      - id: accessed_timestamp
        type: u8
      - id: flags
        type: u8
      - id: nchildren_or_nlink
        type: u4
      - id: unknown_60
        type: u4
      - id: unknown_64
        type: u4
      - id: bsdflags
        type: u4
      - id: owner_id
```

```
      type: u4
    - id: group_id
      type: u4
    - id: mode
      type: u2
    - id: unknown_82
      type: u2
    - id: unknown_84
      type: u4
    - id: unknown_88
      type: u4
    - id: filler_flag
      type: u2
    - id: unknown_94
      type: u2
    - id: unknown_96
      type: u2
    - id: len_name
      type: u2
    - id: name_filler
      type: u4
      if: filler_flag == 2
    - id: name
      type: strz
    - id: unknown_remainder
      size-eos: true
  -webide-representation: '#{node_id:dec} / #{parent_id:dec} "{name}"'

hardlink_record: # 0x50
  seq:
    - id: node_id
      type: u8
    - id: namelength
      type: u2
    - id: dirname
      size: namelength
      type: str
  -webide-representation: '#{node_id:dec} "{dirname}"'

t6_record: # 0x60
  seq:
    - id: unknown_0
      type: u4
```

```
    -webide-representation: '{unknown_0}'

  inode_record: # 0x20
    seq:
      - id: block_count
        type: u4
      - id: unknown_4
        type: u2
      - id: block_size
        type: u2
      - id: inode
        type: u8
      - id: unknown_16
        type: u4
    -webide-representation: '#{inode:dec}, Cnt {block_count:dec} *
    {block_size:dec}, {unknown_4:dec}, {unknown_16:dec}'

  extent_record: # 0x80
    seq:
      - id: size
        type: u8
      - id: block_num
        type: ref_block
      - id: unknown_16
        type: u8
    -webide-representation: '{block_num}, Len {size:dec}, {unknown_16:
    dec}'

  named_record: # 0x90
    seq:
      - id: node_id
        type: u8
      - id: timestamp
        type: u8
      - id: type_item
        type: u2
        enum: item_type
    -webide-representation: '#{node_id:dec}, {type_item}'

  t12_record: # 0xc0
    seq:
      - id: unknown_0
        type: u8
```

```
    -webide-representation: '{unknown_0:dec}'

  extattr_record: # 0x40
    seq:
      - id: type_ea
        type: u2
        enum: ea_type
      - id: len_data
        type: u2
      - id: data
        size: len_data
        type:
          switch-on: type_ea
          cases:
            ea_type::symlink: strz # symlink
            # all remaining cases are handled as a "bunch of bytes",
            thanks to the "size" argument
    -webide-representation: '{type_ea} {data}'


# spaceman (type: 0x05)

  spaceman:
    seq:
      - id: block_size
        type: u4
      - id: unknown_36
        size: 12
      - id: num_blocks
        type: u8
      - id: unknown_56
        size: 8
      - id: num_entries
        type: u4
      - id: unknown_68
        type: u4
      - id: num_free_blocks
        type: u8
      - id: ofs_entries
        type: u4
      - id: unknown_84
        size: 92
      - id: prev_allocationinfofile_block
```

```
        type: u8
      - id: unknown_184
        size: 200
    instances:
      allocationinfofile_blocks:
        pos: ofs_entries
        repeat: expr
        repeat-expr: num_entries
        type: u8

# allocation info file (type: 0x07)

  allocationinfofile:
    seq:
      - id: unknown_32
        size: 4
      - id: num_entries
        type: u4
      - id: entries
        type: allocationinfofile_entry
        repeat: expr
        repeat-expr: num_entries

  allocationinfofile_entry:
    seq:
      - id: version
        type: u8
      - id: unknown_8
        type: u4
      - id: unknown_12
        type: u4
      - id: num_blocks
        type: u4
      - id: num_free_blocks
        type: u4
      - id: allocationfile_block
        type: u8

# btree (type: 0x0b)

  btree:
    seq:
      - id: unknown_0
```

```
        size: 16
      - id: root
        type: ref_block

# checkpoint (type: 0x0c)

  checkpoint:
    seq:
      - id: unknown_0
        type: u4
      - id: num_entries
        type: u4
      - id: entries
        type: checkpoint_entry
        repeat: expr
        repeat-expr: num_entries

  checkpoint_entry:
    seq:
      - id: type_block
        type: u2
        enum: block_type
      - id: flags
        type: u2
      - id: type_content
        type: u4
        enum: content_type
      - id: block_size
        type: u4
      - id: unknown_52
        type: u4
      - id: unknown_56
        type: u4
      - id: unknown_60
        type: u4
      - id: block_id
        type: u8
      - id: block
        type: ref_block

# volumesuperblock (type: 0x0d)

  volumesuperblock:
```

```
    seq:
      - id: magic
        size: 4
        contents: [APSB]
      - id: unknown_36
        size: 92
      - id: block_map_block
        type: ref_block
        doc: 'Maps node IDs to the inode Btree nodes'
      - id: root_dir_id
        type: u8
      - id: inode_map_block
        type: ref_block
        doc: 'Maps file extents to inodes'
      - id: unknown_152_blk
        type: ref_block
      - id: unknown_160
        size: 80
      - id: volume_guid
        size: 16
      - id: time_updated
        type: u8
      - id: unknown_264
        type: u8
      - id: created_by
        size: 32
        type: strz
      - id: time_created
        type: u8
      - id: unknown_312
        size: 392
      - id: volume_name
        type: strz

# enums

enums:

  block_type:
    1: containersuperblock
    2: rootnode
    3: node
    5: spaceman
```

```
  7: allocationinfofile
  11: btree
  12: checkpoint
  13: volumesuperblock
  17: unknown

entry_type:
  0x0: location
  0x2: inode
  0x3: thread
  0x4: extattr
  0x5: hardlink
  0x6: entry6
  0x8: extent
  0x9: name
  0xc: entry12

content_type:
  0: empty
  9: history
  11: location
  14: files
  15: extents
  16: unknown3

item_type:
  4: folder
  8: file
  10: symlink

ea_type:
  2: generic
  6: symlink
```

Literaturverzeichnis

[APFS Guide 1] Apple Inc.: Apple File System Guide. Introduction.
URL: https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/Introduction/Introduction.html (zuletzt abgerufen 18.07.17, 13:07)

[APFS Guide 2] Apple Inc.: Apple File System Guide. Volume Format Comparison.
URL: https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/VolumeFormatComparison/VolumeFormatComparison.html#//apple_ref/doc/uid/TP40016999-CH8-DontLinkElementID_22 (2017, zuletzt abgerufen 25.07.2017, 14:32)

[APFS Guide 3] Apple Inc.: Apple File System Guide. Features.
URL: https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/APFS_Guide/Features/Features.html#//apple_ref/doc/uid/TP40016999-CH5-DontLinkElementID_7 (2017, zuletzt abgerufen 25.08.2017, 18:14)

[APFS Guide 4] Apple Inc.: Apple File System Guide. Frequently Asked Questions.
URL: https://developer.apple.com/library/ content/documentation/FileManagement/Conceptual/APFS_Guide/FAQ/FAQ.html (2017, zuletzt abgerufen 22.08.2017, 17:16)

[Apple iOS] Apple Inc.: What's New in iOS. iOS 11.0.
URL: https://developer.apple.com/library/content/releasenotes/General/WhatsNewIniOS/Articles/iOS_11_0.html#//apple_ref/doc/uid/TP40017637-SW7 (2017, zuletzt abgerufen 06.09.17:27)

[Apple macOS] Apple Inc.: What's New in macOS. macOS 10.13.
URL: https://developer.apple.com/library/content/releasenotes/MacOSX/WhatsNewInOSX/Articles/macOS_10_13_0.html#//apple_ref/doc/uid/TP40017638-SW8 (2017, zuletzt abgerufen 06.09.2017, 17:29)

[Apple Newsroom] Apple Inc.: Newsroom. macOS High Sierra liefert verbesserte Technologien für Speicher, Video und Grafik.
URL: https://www.apple.com/de/newsroom/2017/06/macos-high-sierra-delivers-advanced-technologies-for-storage-video-and-graphics/ (2017, zuletzt abgerufen 20.07.2017, 11:38)

[Apple tvOS] Apple Inc.: What's New in tvOS. tvOS 11.0.
URL: https://developer.apple.com/library/content/releasenotes/General/WhatsNew

inTVOS/Articles/tvOS_11_0.html#//apple_ref/doc/uid/TP40017635-SW7 (2017, zuletzt abgerufen 06.09.2017, 17:25)

[Apple watchOS] Apple Inc.: What's New in watch OS. watchOS 4.0.
URL: https://developer.apple.com/library/content/releasenotes/General/WhatsNew InwatchOS/Articles/watchOS_4_0.html#//apple_ref/doc/uid/TP40017636-SW1 (2017, zuletzt abgerufen 06.09.2017, 17:24)

[ArsTechnica2017] ArsTechnica: Testing out snapshots in Apple's next-generation APFS file system.
URL: https://arstechnica.com/gadgets/2017/02/testing-out-snapshots-in-apples-next-generation-apfs-file-system/ (2017, zuletzt abgerufen 07.09.2017, 11:37)

[Cugu's Blog 2017] Cugu's Blog. APFS filesystem format.
URL: https:// blog.cugu.eu/post/apfs/ (2017, zuletzt abgerufen 03.08.2017, 10:35)

[Cugu Kaitai] Cugu: Kaitai struct file.
URL: https://github.com/cugu/ apfs.ksy/blob/master/apfs.ksy (zuletzt abgerufen 23.09.2017, 12:04)

[Github] GitHub - cugu/apfs.ksy. APFS filesystem format for Kaitai Struct.
URL: https://github.com/cugu/apfs.ksy (2017, zuletzt abgerufen 25.07.2017, 11:02)

[Golem 2016] Golem: APFS: Apple erstellt eigenes modernes Dateisystem.
URL: https://www.golem.de/news/apfs-apple-erstellt-eigenes-modernes-dateisystem-1606-121496.html (14.06.2016, zuletzt abgerufen 26.08.2017, 13:02)

[Golem 2017] Golem: APFS unter iOS 10.3 im Test. Schneller suchen und ein bisschen schneller booten. URL: https://www.golem.de/news/apfs-unter-ios-10-3-im-test-schneller-suchen-und-ein-bisschen-schneller-booten-1703-126964.html (2017, zuletzt abgerufen 20.09.2017, 12:34)

[Kingston] Kingston Technology: XTS Verschlüsselung.
URL: https:// support.kingston.com/de/usb/encrypted_security/xts_encryption (zuletzt abgerufen 09.08.2017, 16:11)

[Macprime1 2009] Macprime: Apple History. Die PowerPC-Krise.
URL: https://www.macprime.ch/applehistory/geschichte/die-powerpc-krise (2009, zuletzt abgerufen 20.07.2017, 10:50)

[Macprime2 2009] Macprime: Apple History.
URL: https://www.macprime.ch/applehistory/geschichte (2009, zuletzt abgerufen 15.08.2017, 14:27)

[Moritz/Steffens/Steffens, 2010] Th. Moritz/H.-J. Steffens/P. Steffens: Prüfungstrainer Informatik. 500 Fragen und Antworten für das Bachelor-Studium. 1. Auflage. Heidelberg: Spektrum Akademischer Verlag, 2010.

[TechTerms] TechTerms: Partition Definition.
URL: https://techterms.com/ definition/partition (zuletzt abgerufen 28.08.2017, 17:16)

[Ubuntuusers] Ubuntuusers.de: Wiki. BTRFS-Dateisystem URL: https://wiki.ubuntuusers.de/Btrfs-Dateisystem/ (2017, zuletzt abgerufen 13.09.2017, 11:28)

[WhatIs: AES] WhatIs.com: Advanced Encryption Standard.
URL: http://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard (zuletzt abgerufen 28.08.2017, 13:34)

[WhatIs: DES] WhatIs.com: Data Encryption Standard.
URL: http://searchsecurity.techtarget.com/definition/Data-Encryption-Standard (zuletzt abgerufen 28.08.2017, 13:37)

[WhatIs: Rijndael] WhatIs.com: Rijndael.
URL: http://searchsecurity.techtarget.com/ definition/Rijndael (zuletzt abgerufen 28.08.2017, 14:10)

[Wilkinson 2012] M. Wilkinson: MBR_GPT_cheatsheet.
URL: http://www.writeblocked.org/resources (2012, zuletzt abgerufen 20.07.2017, 10:20)

[XTS] Wikipedia: Disk encryption.
URL: https://en.wikipedia.org/wiki/Disk_encryption_theory#Xor-encrypt-xor_.28XEX.29 (zuletzt abgerufen: 28.08.2017, 11:13)

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 09. Oktober 2017

Glossar

BTRFS BTRFS ist die Abkürzung für B-Tree File System. Das umgangssprachlich auch ButterFS oder BetterFS genannte Dateisystem wurde von Oracle Corporation für Linux entwickelt. Es soll das bisher hauptsächlich bei Linux verwendete ext ersetzen und basiert auf B-Trees zur Dateiverwaltung. BTRFS implementiert Features wie Copy-on-Write, Snapshots und Prüfsummen sowie effiziente Speicherung und Verwaltung von Daten und Metadaten..

Datenträgerimages Ein Datenträgerimage oder kurz Image ist das Abbild eines Datenträgers. Ohne die Interpretation durch Forensik Suites kann man solche Datensicherungen als Folge von Hexadezimalzahlen in einem Hex-Editor betrachten..

DiskUtility DiskUtility ist ein Programm, dass unter dem Apple-Betriebssystem macOS für die Verwaltung der Speichermedien zuständig ist. Über DiskUtility werden beispielsweise Partitionen erstellt, bearbeitet oder gelöscht..

EPS EPS ist die Abkürzung für Encapsulated Post Script und ein Format für Bilddateien. Es kann Vektor-, Bitmap- und Textdaten enthalten. Die Inhalte sind in der Seitenbeschreibungssprache Post Script gespeichert..

Extent Ein Extent ist ein zusammenhängender Speicherbereich, der reservierten Platz für eine Datei repräsentiert. Da es sich um einen Bereich aufeinander folgender Allocation Blocks handelt, kann er als Paar von zwei Werten dargestellt werden - der Blocknummer des Anfangsblocks und der Länge des Extents..

FAT FAT ist die Abkürzung für File Allocation Table (Dateizuordnungstabelle). Dieses von Microsoft entwickelte Dateisystem kommt hauptsächlich auf mobilen Datenträgern zur Anwendung. Es gibt davon mehrere Varianten, beispielsweise FAT16 und FAT32..

FileVault FileVault ist eine Funktion des Apple-Betriebssystems macOS, welche die Benutzerdaten unter Verwendung von AES-XTS verschlüsseln kann..

Forensik Suites Forensik Suites sind umfangreiche Software-Tools, mit denen Datenträgerimages erstellt, eingelesen und ausgewertet werden können. Sie implementieren unter anderem Funktionen wie Suchen, Rekonstruktion von Dateien, Metadatenextraktion, Abgleich mit Hashlisten und Berichterstellung. Bekannte Vertreter sind XWay Forensics, EnCase Forensics und das Forensik Toolkit..

GPT GPT steht für GUID Partition Table und ist der am weitesten verbreitete Standard für Partitionstabellen von permanenten Massenspeichermedien..

GUID GUID ist die Abkürzung für Global Unique Identifier und bezeichnet eine 16 Byte große Identifikationsnummer, welche unter anderem als eindeutige Partitionskennung oder zum Bestimmen des Partitionstypen bei GPT verwendet wird..

HDD HDD ist die Abkürzung für Hard Disk Drive und bezeichnet Festplatten, die aus mehreren rotierenden magnetischen Platten bestehen..

HFS HFS ist die Abkürzung für Hierarchical File System. Dieses hierarchische Dateisystem wurde von Apple 1985 eingeführt und ab 1998 von HFS+ (HFS Plus) abgelöst. Eine weitere Version ist HFSX (HFS Extended) und wird auf Apples Mobilgeräten verwendet..

Inode Inodes (Index Nodes) sind wichtige Datenstrukturen bei unixartigen Betriebs- und Dateisystemen. In der Inode sind die Metadaten, wie Pfad, Erstelldatum und Name sowie Verweise auf die eigentlichen Daten einer Datei gespeichert..

Kaitai Struct Kaitai Struct ist eine Beschreibungssprache für Binärdateien. Die bei der Analyse von Binärstrukturen entstehenden .ksy-Dateien können in Quelldateien für verschiedene Programmiersprachen umgewandelt werden..

KiB KiB steht für Kibibyte. 1 Kibibyte entspricht genau 1024 Byte, das meistens synonym dazu verwendete Kilobyte (KB) jedoch nur 1000 Byte. Analog dazu werden in dieser Arbeit MiB (Mebibyte) und GiB (Gibibyte) statt MB (Megabyte) und GB (Gigabyte) verwendet..

Komplexität Als Komplexität eines Algorithmus bezeichnet man den größtmöglichen Aufwand, den er bei den ungünstigsten Eingabedaten besitzt. Es werden Rechenkomplexität und Speicherkomplexität unterschieden, also der Bedarf an Rechenzeit und Speicherplatz. Dieser Bedarf wächst bei verschiedenen Algorithmen oder Datenstrukturen mit größer werdender Menge an Eingabedaten unterschiedlich schnell. Angegeben wird dieser Bedarf in der O-Notation, z. B. als $O(\log_2(n))$ für logarithmischen Aufwand oder O(n) für linearen Aufwand..

MBR MBR ist die Abkürzung für Master Boot Record und bezeichnet einen Standard für Partitionstabellen von permanenten Massenspeichermedien..

NTFS NTFS ist die Abkürzung für New Technology Filesystem. Dieses Dateisystem ist das auf Microsoft-Rechnern am häufigsten verwendete. Dateien werden über die Master File Table (MFT) organisiert..

Schutzziel der IT-Sicherheit Die wesentlichen Schutzziele der IT-Sicherheit sind Verfügbarkeit, Integrität und Vertraulichkeit der Daten. Es geht dabei darum, dass die Daten jederzeit verfügbar sind, unverändert und korrekt vorliegen und nur dem Benutzer zugänglich sind, für den sie bestimmt sind..

SSD SSD ist die Abkürzung für Solid State Drive und bezeichnet elektronische Speichermedien, die aus vielen aneinander geschalteten Halbleiterbausteinen bestehen - den Floating Gate Transistoren. Diese elektrischen Bausteine haben nur eine begrenzte Anzahl an möglichen Beschreibe- und Löschvorgängen, bevor sie sich abnutzen, weswegen SSDs einem Alterungsprozess unterliegen. In einem Flashspeicher ist eine Page die kleinste beschreibbare Einheit, ein Block, welcher aus mehreren Pages besteht, die kleinste löschbare Einheit..

TimeMachine TimeMachine ist ein Programm von Apple für macOS, welches unter anderem das Erstellen von Backups und finden alter Dateiversionen ermöglicht..

XOR XOR bedeutet exklusives Oder. In der klassischen Logik wird die XOR-Verknüpfung genau dann wahr, wenn entweder Aussage A oder Aussage B wahr sind, jedoch nicht beide gleichzeit wahr oder unwahr. In der Informatik entsteht so aus der Verknüfung von 0 xor 1 oder 1 xor 0 der Zustand 1, aus 0 xor 0 oder 1 xor 1 der Zustand 0. Anwendung findet das unter anderem in diversen Blockverschlüsselungsmodi, bei denen beispielweise ein Klartextblock mit dem vorhergehenden Geheimtextblock XOR-verknüpft wird..