HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Herr
Elias Ullrich

Analyse des Messengerdienstes WhatsApp aus forensischer Sicht

2018

Fakultät **Angewandte Computer- und Biowissenschaften**

BACHELORARBEIT

Analyse des Messengerdienstes WhatsApp aus forensischer Sicht

Autor:
Elias Ullrich

Studiengang:
Allgemeine und Digitale Forensik

Seminargruppe:
FO15w4-B

Erstprüfer:
Prof. Dr. rer. nat. Christian Hummert

Zweitprüfer:
Dipl.-Inf. Andreas Sommer

Mittweida, 2018

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

Forensic analysis of the messenger service WhatsApp

Author:
Elias Ullrich

Course of Studies:
General and Digital Forensic Science

Seminar Group:
FO15w4-B

First Referee:
Prof. Dr. rer. nat. Christian Hummert

Second Referee:
Dipl.-Inf. Andreas Sommer

Mittweida, 2018

Bibliographische Angaben

Ullrich, Elias: Analyse des Messengerdienstes WhatsApp aus forensischer Sicht. 119 Seiten, 21 Abbildungen, 45 Tabellen, Hochschule Mittweida, University of Applied Sciences, Fakultät Angewandte Computer- und Biowissenschaften.

Bachelorarbeit, 2018

Referat

In dieser Bachelorarbeit wird der Messengerdienst WhatsApp auf Mobilgeräten mit Android-Betriebssystem aus digitalforensischer Perspektive beleuchtet. Dazu werden technische Aspekte der Funktionsweise sowie von der Anwendung auf dem Gerät gespeicherte Dateien und die enthaltenen forensischen Artefakte ebenso diskutiert wie mehrere Möglichkeiten, WhatsApp betreffende Daten zu extrahieren. Des Weiteren werden sowohl das WhatsApp-interne als auch das Android-Systemlog analysiert, um den erstellten Einträgen die zugrunde liegenden Nutzeraktivitäten zuordnen zu können. Anschließend wird ein Programm vorgestellt, das die gewonnenen Erkenntnisse nutzt, um automatisiert aus den gegebenen Logdateien Ereignisse zu extrahieren und aufzubereiten.
Die Arbeit soll für forensische Untersuchungen, bei denen WhatsApp eine Rolle spielt, sowohl die Informationen als auch ein Werkzeug bereitstellen, mit dem die Aktivitäten des Nutzers nachvollzogen und wichtige Spuren gefunden werden können.

Bibliographic Information

Ullrich, Elias: Forensic analysis of the messenger service WhatsApp. 119 pages, 21 figures, 45 tables, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences.

Bachelor Thesis, 2018

Abstract

In this Bachelor Thesis we take a closer look at the messenger service WhatsApp on mobile devices powered by the Android operating system from a digital forensics point of view. We discuss the technical foundations of the app's functionality and the forensic artefacts that are contained in files stored on the device. Additionally, multiple methods for extracting WhatsApp related data from such devices are presented. Both, WhatsApp's own logfile and Android's system log are analyzed in order to identify correlations between log entries and various user activities. Using the outcome of the above mentioned analysis, we present an application for extracting and refining events from supplied log files in an automated manner.
This work is intended to be an information source for forensic examinations bearing relation with the WhatsApp application and provides a tool to retrace the user activities in order to discover important forensic evidence.

Danksagungen

An dieser Stelle möchte ich mich bedanken bei all denen, die mich während der Erstellung dieser Arbeit unterstützt und zu einem wesentlichen Teil dazu beigetragen haben, dass sie in dieser Form überhaupt möglich ist.

Zuerst gilt mein Dank dem Landeskriminalamt Thüringen, besonders den Sachverständigen im Dezernat Forensische IuK, die mir in der gesamten Zeit der Recherche, Forschung und des Schreibens dieser Bachelorarbeit gute und freundliche Kollegen waren und stets mit hilfreichen Ratschlägen und Hinweisen zur Seite standen. Insbesondere möchte ich mich bei Silvio Sterner bedanken, der mir das Thema der forensischen WhatsApp-Analyse vorgeschlagen hat und viele Ideen beisteuerte, die in der digitalen Forensik von Bedeutung sind und näherer Untersuchung bedürfen, sowie bei Andreas Sommer, dessen Unterstützung ich besonders in Bezug auf das Programmieren und einen wissenschaftlichen Ausdruck schätzen gelernt habe.
Außerdem danke ich all jenen freiwilligen Helfern, die Entwürfe meiner Arbeit sowohl auf inhaltliche als auch auf sprachliche oder formale Mängel überprüft haben und mir hilfreiche Verbesserungsvorschläge geben konnten. Besonders meinem Papa und meinem Cousin möchte ich danken, die trotz vieler eigener Verpflichtungen meine Arbeit gelesen haben, aber auch erneut einigen Kollegen aus dem Thüringer Landeskriminalamt, die ihre Zeit dafür einsetzten.
Weiterhin bedanke ich mich bei meiner Familie und vor allem meiner Verlobten, die mich bei der Erstellung unterstützt und auf mich Rücksicht genommen sowie mich immer wieder motiviert (und zur richtigen Zeit auch abgelenkt) haben.

Zuletzt möchte ich Jesus Christus, meinem Heiland und Erlöser, danken, der mich immer wieder mit Wissen und guten Ideen beschenkt und ohne den ich nie in der Lage gewesen wäre, diese Arbeit zu verfassen.

I Inhaltsverzeichnis

II Abbildungsverzeichnis

III Tabellenverzeichnis

IV Listingverzeichnis

V Abkürzungsverzeichnis

$\mathcal{ID}_{Ereignis}$ ..... textuelle Repräsentation eines Ereignistyps für die Identifizierung

$\mathcal{T}_{Emu}$ ..... emuliertes Android-Gerät

$\mathcal{T}_{WA}$ ..... auszulesendes Telefon mit installiertem WhatsApp

$\mathcal{V}_{inst}$ ..... auf dem auszulesenden Telefon installierte WhatsApp-Version

ADB ..... Android Debug Bridge

AES ..... Advanced Encryption Standard

App ..... Application Software (Softwareanwendung)

CBC ..... Cipher Block Chaining

CSV ..... Comma Separated Values

DNS ..... Domain Name System

E-Mail ..... Electronic Mail

ECDH ..... Elliptic Curve Diffie-Hellman

GCM ..... Galois/Counter Mode

GIF ..... Graphics Interchange Format

GSM ..... Global System for Mobile Communications

GUI ..... Graphical User Interface

HKDF ..... HMAC-based Key Derivation Function

HMAC ..... Hash based Message Authentication Code

HTML ..... Hyper Text Markup Language

HTTP ..... Hypertext Transfer Protocol

IM ..... Instant Messaging

IP ..... Internet Protocol

IV ..... Initialisierungsvektor

JID ..... Jabber ID

JPEG ..... Joint Photographs Experts Group

LE ..... Little Endian

MD5 ..... Message Digest 5

MMS ..... Multimedia Messaging Service

PDF ..... Portable Document Format

PID ..... Process Identifier

PIN ..... Personal Identification Number

QR-Code ..... Quick Response Code

RAM ..... Random Access Memory

Regex ..... Regular Expression

RFC ..... Request For Comments

SASL ..... Simple Authentication and Security Layer

SD Card ..... Secure Digital Memory Card

SHA ..... Secure Hash Algorithm

SIM ..... Subscriber Identity Module

SMS ..... Short Message Service

SP ..... Signal Protocol

SSD ..... Solid State Drive

TCP ..... Transmission Control Protocol

TID ..... Thread ID

TLS ..... Transport Layer Security

UPI ..... Unified Payment Interface

USB ..... Universal Serial Bus

UTF-8 ..... Unicode Transformation Format – 8 Bit

WAL ..... Write Ahead Log

WATE ..... WhatsApp Timeline Extractor

WLAN ..... Wireless Local Area Network

XML ..... Extensible Markup Language

XMPP ..... Extensible Messaging and Presence Protocol

1 Einleitung

Fast zwei Milliarden Menschen weltweit nutzen zur Kommunikation Messenger-Apps auf ihrem Smartphone [eMa17]. In Deutschland ist dabei „WhatsApp“, ein im Februar 2009 entwickelter US-amerikanischer Messenger, der beliebteste Dienst zum Versand von Kurzmitteilungen [eMa15; Ols14]. Die Applikation hat vor allem bei jungen, aber vermehrt auch bei älteren Menschen einen hohen Stellenwert im digitalen Leben, bei einigen Smartphone-Herstellern ist sie sogar schon bei der Auslieferung des Telefons vorinstalliert.
Außerdem ist zu beobachten, dass sich zwischenmenschliche Kommunikation immer mehr in den digitalen Raum verlagert [LM90] - Anrufe, Briefe oder persönliche Gespräche werden vermehrt durch Nachrichten per E-Mail, in sozialen Netzwerken oder eben in Messenger-Apps ersetzt. Das heißt zwangsläufig, dass diese Medien (und damit auch WhatsApp) immer mehr sehr persönliche und vertrauliche Inhalte enthalten. Doch nicht nur textuelle Nachrichten werden mit WhatsApp übertragen. Da es außerdem die Möglichkeit bietet, Anrufe zu tätigen sowie Sprachaufnahmen, Bilder, Videos und andere Dateien auszutauschen, landen vermehrt auch solche, oft persönliche, Daten auf den Servern des Anbieters.

Dass WhatsApp auch in der Forensik eine hohe Bedeutung hat [Tha13], ist also nicht von der Hand zu weisen. Die Wahrscheinlichkeit ist groß, dass Menschen, die an Strafprozessen beteiligt sind, den Dienst ebenfalls nutzen. Planungen und Absprachen zwischen mehreren Tätern sowie die Kommunikation zwischen Opfer und Täter erfolgen mitunter vollständig über diese Anwendung, aber auch Medien und andere Dateien, die für eine Straftat relevant sind, werden verschickt.
In vielen Fällen kann eine forensische Untersuchung von Mobiltelefonen im Allgemeinen und den Spuren von WhatsApp im Speziellen also zu wertvollen Hinweisen und Beweisen führen. Dabei sind nicht nur verschickte Nachrichten oder Dateien von Bedeutung, sondern auch verschiedene Logeinträge und Metadaten lassen Rückschlüsse auf die Benutzung der Anwendung zu.

Das Ziel dieser Bachelorarbeit ist, aus forensischer Sicht einen Überblick über WhatsApp auf dem mobilen Betriebssystem Android zu geben sowie verschiedene Methoden zur Extraktion von Daten in Bezug auf den Messenger vorzustellen. Des Weiteren soll ein Zusammenhang zwischen WhatsApp betreffenden Einträgen in verschiedenen

Logdateien und in der Realität erfolgten Handlungen hergestellt werden, was durch die Entwicklung eines eigenen Programms automatisiert wird.

Zuerst werden in Kapitel 2 die Funktionsweise und die verschiedenen Möglichkeiten der App erläutert, anschließend wird auf die Umsetzung des Nachrichtenversands und die eingesetzte Verschlüsselung eingegangen. Auch die von WhatsApp auf dem Gerät gespeicherten Dateien und ihre Speicherorte werden erläutert. Ein besonderes Augenmerk liegt dabei auf forensisch relevanten Spuren, die auf dem Telefon hinterlassen werden und Rückschlüsse auf die Benutzung zulassen.
Im 3. Kapitel werden verschiedene Methoden vorgestellt, mit denen WhatsApp-Daten aus Android-Geräten extrahiert werden können, ohne dass eine physikalische Sicherung vorliegt und ohne Root-Zugang zu besitzen. Nach den verwendeten Methoden zum Analysieren relevanter Logdateien wird außerdem ein Programm vorgestellt, das aus diesen Logs, basierend auf den Ergebnissen der vorangegangenen Analyse, automatisiert wichtige Ereignisse extrahiert.
Beispiele für die Ergebnisse der Extraktionsmethoden aus Kapitel 3 sowie die mit realen Ereignissen korrelierenden Logeinträge werden in Kapitel 4 angeführt. Des Weiteren wird das Ergebnis eines Genauigkeitstests des entwickelten Programms zur Logfile-Analyse präsent.
Schließlich wird in Kapitel 5 die Arbeit zusammengefasst und in den wissenschaftlichen Kontext gestellt. Zudem wird ein Ausblick darauf gegeben, wie das Thema WhatsApp-Forensik zukünftig noch weiter erforscht werden kann.

2 Grundlagen

In diesem Kapitel werden die funktionalen und technischen Grundlagen von WhatsApp erläutert.
Da eine solche Anwendung regelmäßigen Aktualisierungen und Veränderungen unterliegt, wurden alle Informationen zum August 2018 validiert, es steht aber nicht fest, wie lange diese aktuell bleiben.
Die verwendete WhatsApp-Version ist 2.18.191 (veröffentlicht im Juni 2018) auf Android 5.1 (siehe auch Abschnitt 3.1).

2.1 Die Funktionalität von WhatsApp

WhatsApp ist eine Instant Messaging (IM)-Applikation, die für die mobilen Betriebssysteme Android, iOS, Windows Phone und (bis Ende 2018) Nokia S40 verfügbar ist [Wha18b; Wha18c]. Außerdem wird auch die Nutzung am Computer unterstützt. Hierzu gibt es eine offizielle Web-Version [Wha18d], die über jeden Webbrowser aufgerufen werden kann, sowie native Applikationen für Windows (ab Version 8) und Mac OS X (ab Version 10.9) [Wha18c].

Der IM-Dienst unterstützt eine Vielzahl an Funktionen, die im Laufe der Zeit immer mehr erweitert wurden und im Folgenden aufgeführt und kurz erläutert werden.

Text- und Sprachnachrichten. Kern der Anwendung sind die textuellen Kurzmitteilungen, die Nutzer untereinander verschicken und empfangen können. Dabei erlaubt WhatsApp den Einsatz von sog. Emoticons, also bunten Symbolen zum Ausdruck des aktuellen Befindens oder Ersetzen von schriftlichem Inhalt.
Des Weiteren lassen sich Audioaufnahmen mit Hilfe des im Telefon eingebauten Mikrofons direkt in WhatsApp erstellen und als Sprachnachrichten versenden. Diese Funktion wurde 2013 eingeführt [Kou13].

Medien- und Dateiversand. Weniger als ein Jahr nach Veröffentlichung der App führte WhatsApp die Funktion zum Versenden von Fotos, Videos und Audiodateien ein, wie es vorher nur mit MMS-Nachrichten möglich war [Kou09].
Seit 2016 ist es auch möglich, zuerst nur PDF-Dokumente und später auch andere Dateien mit dem Messenger zu verschicken [Per16].

Löschen von Nachrichten. Versandte Nachrichten können aus dem entsprechenden Chat wieder entfernt werden. Dabei unterstützt WhatsApp die Funktionen „Für mich löschen“ und "Für alle löschen“. Im ersten Fall wird nur die eigene Kopie der Nachricht entfernt, wobei die Inhalte gänzlich vom Telefon gelöscht werden, durch Betätigen der letzteren Funktion wird die Nachricht auf den Geräten aller Kommunikationspartner durch den Hinweis „Diese Nachricht wurde gelöscht“ ersetzt und damit unzugänglich gemacht [Wha18a].

Einzel- und Gruppenchats. Nicht nur der Versand von Nachrichten zu einem einzelnen Kommunikationspartner ist möglich. Seit 2011 gibt es Gruppenchats, zu denen mehrere WhatsApp-Benutzer hinzugefügt werden können und die von einem oder mehreren Administratoren verwaltet werden. Dabei wird eine neue Nachricht automatisch an alle Mitglieder der Gruppe verteilt [Kou11].

Sprach- und Videoanrufe. 2015 bekam WhatsApp eine neue Funktion, mit der sich, ähnlich der normalen Anrufe im GSM-Netz, Sprachanrufe tätigen lassen. Im nächsten Jahr kamen Videoanrufe hinzu, wie sie von anderen Diensten wie Microsoft Skype bekannt sind [Kou16b; Ols15]. Dabei erfolgen beide Varianten über das Internet - entweder im WLAN oder über das mobile Datennetz.

Standort teilen. Nachdem zuerst nur der aktuelle Standort in Form von Koordinaten und einem Link zum Anzeigen auf einer Karte an andere gesendet werden konnte, wurde 2017 eine Funktion namens „Live-Standort“ eingeführt, mit der die aktuelle Position geteilt werden kann und sich während einer eingestellten Zeit automatisch aktualisiert [Kou17a]. So kann der Empfänger (bzw. die Empfängergruppe) genau nachvollziehen, wo sich der Absender während dieser Zeit aufhält.

WhatsApp Web. Einen Zugriff auf fast alle Funktionen, die ein Benutzer der IM-App auf seinem Telefon hat, bietet WhatsApp über die 2015 eingerichtete Website `web.whatsapp.com`. Sobald der Nutzer sein Telefon durch Scannen eines QR-Codes mit dem Online-Dienst verbindet, laufen alle Aktionen in der App und auf WhatsApp Web synchron ab. Diese Verbindung bleibt solange bestehen, wie beide Seiten mit dem Internet verbunden sind [Kou15].
Die native Windows- bzw. OS X-Anwendung von WhatsApp basiert auf dem gleichen Prinzip wie die Website [Kou16a].

Statusmeldungen. Ursprünglich wurde WhatsApp entwickelt, um seinen „Status“ mit anderen zu teilen, z.B. eine kurze Beschreibung der aktuellen Beschäftigung o.Ä. Seit Februar 2017 ist es möglich, für diesen Status beliebig viele Texte, Fotos, Videos oder GIF-Animationen, die mit verschiedenen Effekten bearbeitet werden können, zu veröffentlichen [Kou17b]. Dabei kann in der Grundeinstellung der Status einer Person von jedem Nutzer gesehen werden, der deren Telefonnummer in seinem Kontaktbuch gespeichert hat. Der Zugriff ist jedoch auf ausgewählte Kontakte beschränkbar. Die Statusmeldungen sind jeweils für 24 Stunden sichtbar und werden danach automatisch entfernt.

Geldzahlungen. Seit Anfang 2018 wird, zunächst nur in Indien, eine neue Funktionalität getestet, mit der sich, einmal eingerichtet, über WhatsApp auch Geldzahlungen versenden lassen. Wenn beide Kommunikationspartner die Funktion aktiviert und eingerichtet haben, kann durch eine solche Zahlungs-Nachricht über einen Dienst namens „UPI“ (Unified Payment Interface) Geld vom einen auf das andere Bankkonto übertragen werden [Pet18].

Die genannten Chatmöglichkeiten unterliegen jeweils technischen oder festgelegten Einschränkungen, wie z.B. der maximalen Anzahl von Zeichen je Nachricht. In Tabelle 2.1 sind die jeweiligen Einschränkungen dargestellt.

Chatfunktion	**Maximum**	**Einheit**
Textnachricht	65536	Zeichen
Sprachnachricht	> 230	Minuten*
Gruppenchat	256	Mitglieder
Live-Standort	8	Stunden
Statusmeldung	> 200	gleichzeitig aktiv

* Die Länge von Sprachnachrichten kann wahrscheinlich sehr groß sein oder ist gar nicht begrenzt. Es ist keine Maximallänge bekannt.

Tabelle 2.1: Technische Einschränkungen bei WhatsApp
Quelle: eigene Arbeit

2.2 Technische Umsetzung

2.2.1 Registrierung

Im Unterschied zu vielen anderen Messenger-Diensten identifiziert WhatsApp seine Nutzer anhand einer Telefonnummer mit Landesvorwahl [Sah14]. Das heißt, wer sich bei dem IM-Dienst anmeldet, muss nach Installation der App zuerst seine Telefonnummer angeben. Dies muss nicht zwangsläufig die Nummer des Smartphones sein, auf dem die App läuft. Somit lassen sich auch Geräte ohne GSM-Modul (z.B. Tablets) bei WhatsApp registrieren.
Nachdem die Nummer eingegeben wurde, wird der Nutzer vor die Wahl gestellt, ob er für die Registrierung eine SMS oder einen Anruf mit einem Verifizierungscode empfangen möchte. Auf dem gewählten Weg empfängt er einen sechsstelligen Zahlencode, den er in der Anwendung angeben muss. Nachdem dieser überprüft wurde, ist das Telefon für diese Telefonnummer registriert. Dabei kann allerdings immer nur ein Gerät gleichzeitig mit einer Nummer gekoppelt sein.
Nach der Registrierung liest WhatsApp (bei Erteilen der entsprechenden Berechtigung) automatisch alle Kontakte des Telefons und synchronisiert diese mit der Datenbank des Anbieters, um stets jedem Nutzer anzeigen zu können, welche seiner Kontakte die App nutzen [Sah14].

2.2.2 Nachrichtenversand

Der Versand von Nachrichten erfolgt über das Internet mit Hilfe einer modifizierten Variante des Protokolls *XMPP* (Extensible Messaging and Presence Protocol) [Gup16; Hof14].

2.2.2.1 XMPP allgemein

Das Protokoll basiert auf der Auszeichnungssprache *XML* (Extensible Markup Language [Bra+06]) und bietet eine (annähernd) echtzeitfähige Übertragung von strukturierten Daten. Als Zeichenkodierung kommt laut dem RFC [Sai11] bei XMPP ausschließlich UTF-8 zum Einsatz.
Die Kommunikation über XMPP erfolgt in einem Client-Server-Netzwerk, wobei einzelne Entitäten (Nutzer, Hosts und Ressourcen) mit einer sog. *JabberID*[1], kurz

[1]Das Protokoll XMPP entstand in Verbindung mit dem IM-Server „Jabber“, daher der Name.

JID adressiert werden. Diese setzt sich nach dem Schema `<Nutzer>@<Domäne>/<Ressource>` zusammen, wobei die Domäne die Adresse des XMPP-Servers ist und die Ressource ein einzelnes verbundenes Gerät des jeweiligen Nutzers kennzeichnet. Das Domain Name System (DNS) wird zur Namensauflösung genutzt (vgl. [SST09]).

Eine XMPP-Verbindung wird oft mittels TLS-Verschlüsselung geschützt und verwendet TCP als Transportprotokoll. Im Folgenden wird der Ablauf einer solchen Verbindung grob dargestellt (nach [Sai11]).

1. Herstellen einer TCP-Verbindung zu gegebener IP-Adresse und Port
2. Öffnen je eines XML-Streams (bestehende Verbindung, Datenstrom) über diese TCP-Verbindung in beide Richtungen
3. Einrichten einer TLS-Verschlüsselung und Authentifizierung über SASL, einem Framework für Authentifizierung und Sicherheitsfeatures [MZ06] (sofern verfügbar).
4. Anbindung der korrekten Ressource an den Stream
5. Austausch einer beliebigen Anzahl von XML-Elementen (sog. *XML-Stanzas*) mit dem Kommunikationspartner
6. Abbau des XML-Streams und der TCP-Verbindung

Der entscheidende Teil der Verbindung ist Punkt 5. Hier ist die Verbindung hergestellt, und ein Stream von XML-Stanzas kann in beide Richtungen versandt werden. Ein solches Stanza kann verschiedene Daten enthalten, wie z.B. den Inhalt einer Nachricht, die an den Client verschickt wurde. Ein Beispiel eines solchen Stanzas ist in Listing 2.1 (entnommen aus [SST09]) zu sehen.
Dadurch, dass die Verbindung über die gesamte Sitzung bestehen bleibt und auch bei „Funkstille“ zwischen Client und Server nicht abgebaut wird, erreichen neue Nachrichten sehr schnell ihre Empfänger. Hierbei wird die Echtzeitfähigkeit von XMPP deutlich.

```
1 <message from="queen@wonderland.lit" to="↩
  ↪ madhatter@wonderland.lit">
2   <body>Off with his head!</body>
3 </message>
```

Listing 2.1: Beispiel eines XML-Stanzas für eine neue Nachricht

2.2.2.2 Einsatz von XMPP bei WhatsApp

Wie bereits erwähnt, nutzt WhatsApp zur Übertragung von Nachrichten das Protokoll XMPP, wobei jedes Gerät für eine XMPP-Ressource steht. Theoretisch könnte jeder Nutzer mehrere Geräte gleichzeitig besitzen, der Messenger erlaubt jedoch immer nur ein registriertes Gerät je Telefonnummer (siehe Abschnitt 2.2.1).
Vom Aufbau der XMPP-Verbindung über das Versenden und Empfangen von Nachrichten bis zum Abbau der Verbindung werden die von WhatsApp übermittelten Daten verschlüsselt. Nähere Informationen dazu sind in Abschnitt 2.2.3 nachzulesen, weshalb hier auf diese Verschlüsselung nicht weiter eingegangen wird.

Da der Absender über den Status seiner Nachricht informiert wird[2], muss er nach erfolgreicher Zustellung eine Rückmeldung erhalten. Der Ablauf einer Nachrichtenzustellung läuft dabei nach dem folgenden Schema ab, wobei jede Nachricht mittels einer eindeutigen *Nachrichten-ID*, bestehend aus 32 Hexadezimalzahlen, identifiziert wird (nach [Gup16; Hof14]):

1. Verfassen und Abschicken der Nachricht durch den Absender
2. Übermitteln eines XML-Stanzas mit der neuen Nachricht zum WhatsApp-Server über die bestehende XMPP-Verbindung
3. Rückmeldung über die Ankunft beim Server
4. Vorhalten der Nachricht in einer Warteschlange, bis der Empfänger mit dem Internet verbunden ist und eine XMPP-Verbindung mit dem Server aufgebaut hat
5. Zustellung eines XML-Stanzas mit der neuen Nachricht an den Empfänger
6. Löschen der Nachricht auf dem WhatsApp-Server
7. Rückmeldung über die Auslieferung an den Absender
8. Rückmeldung des Empfängers an den Server, wenn die neue Nachricht gelesen wurde und Weiterleiten dieser Information an den Absender

Da XML-Stanzas nur textuelle Daten transportieren können, werden Mediendateien wie Bilder, Videos oder Sprachnachrichten gesondert auf einen HTTP-Server hochgeladen. Als Nachricht an den Empfänger wird der entsprechende Link zum Herun-

[2] Über folgende Status bekommt der Nutzer Bescheid: *Ankunft der Nachricht beim Server*, *Ankunft der Nachricht beim Empfänger* und (sofern beidseitig in den Einstellungen erlaubt) *Lesen der Nachricht durch den Empfänger*.

terladen und ein Base64[3]-kodiertes Vorschaubild gesendet [Hof14].

Adressiert werden WhatsApp-Nutzer, wie bei XMPP üblich, mittels einer *JID*, die stets nach der folgenden Form gebildet wird (der Telefonnummer wird jeweils keine 0 vorangestellt) [Sah14].

- bei einzelnen Nutzern: `<Ländercode><Telefonnummer>@s.whatsapp.net` (z.B. *49123456789@s.whatsapp.net*)
- bei Gruppenchats: `<Ländercode><Telefonnummer>-<Epoch-Zeitstempel der Gruppenerstellung>@g.us` (z.B. *49123456789-1426340220@g.us*), wobei die Nummer immer die des Gruppenerstellers ist, auch wenn dieser nachträglich die Gruppe verlässt

2.2.3 Ende-zu-Ende-Verschlüsselung

Seit April 2016 bietet WhatsApp verschlüsselte Nachrichtenübertragung an [KA16]. Seitdem werden alle übermittelten Daten mit dem *Signal Protocol*, kurz SP, das von der Organisation „Open Whisper Systems“ entwickelt wurde, Ende-zu-Ende-verschlüsselt [Wha17].
Das bedeutet, dass eine zu versendende Nachricht schon auf dem Gerät des Absenders verschlüsselt und erst dann über den WhatsApp-Server an den Empfänger übermittelt wird, der sie dann auf seinem eigenen Gerät wieder entschlüsselt. Es kann also weder ein Außenstehender, der den Netzwerkverkehr mitschneidet, noch WhatsApp selbst, die Nachricht im Klartext lesen.

Dieser Abschnitt soll einen kurzen Überblick über die Funktionsweise des verwendeten Algorithmus bei der Verschlüsselung einer Textnachricht geben (nach [Wha17]). Wie die Verschlüsselung bei weiteren Nachrichtentypen wie Statusmeldungen, Medien- oder Gruppennachrichten (siehe auch Abschnitt 2.1) funktioniert, ist dem von WhatsApp veröffentlichtem Whitepaper [Wha17] zu entnehmen.
An dieser Stelle sei außerdem auf die in Abschnitt 5.1.1 diskutierten Schwächen der Verschlüsselung hingewiesen.

[3]Das Base64-Verfahren kodiert Binärdaten als Text aus ASCII-Zeichen. Dadurch ist es möglich, solche Daten in Umgebungen zu nutzen, die ausschließlich textuelle Inhalte erlauben [Jos06].

2.2.3.1 Vorbereitungen bei der Registrierung

Wenn ein Telefon bei WhatsApp registriert wird, generiert dieses verschiedene privat-öffentliche Schlüsselpaare. Darunter sind der `Identity Key`, der eindeutig für ein bestimmtes Nutzerkonto ist, ein `Signed Pre Key`, der vom `Identity Key` signiert wurde, und mehrere `One-Time Pre Keys`, die jeweils nur einmal verwendet werden und bei Bedarf neu generiert werden. Bei der Generierung aller drei Keys kommt die Elliptische Kurve[4] *Curve25519* zum Einsatz.
Die öffentlichen Schlüssel der genannten Paare werden bei der Registrierung an den WhatsApp-Server gesendet, damit dieser sie später an die entsprechenden Kommunikationspartner weiterleiten kann.

Zusammengefasst wird der beschriebene Prozess in Abbildung 2.1.

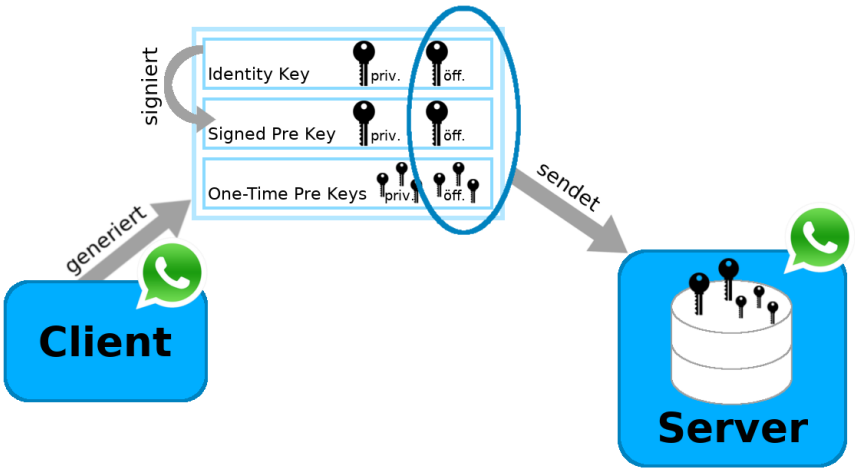


Abbildung 2.1: Schematischer Ablauf der Schlüsselerzeugung bei der Registrierung
Quelle: eigene Arbeit, Icons von `pngimg.com` und `Freepik auf www.flaticon.com`

2.2.3.2 Aufbau einer verschlüsselten Sitzung

Will ein WhatsApp-Nutzer einem seiner Kontakte eine Nachricht schreiben, muss er zuerst eine verschlüsselte Sitzung zu ihm aufbauen. Sobald der Kontakt mit dem Internet verbunden ist, baut dieser ebenfalls eine verschlüsselte Sitzung auf. Diese bleiben solange bestehen, bis einer der beiden Kommunikationspartner WhatsApp neu installiert oder ein neues Telefon registriert, erkennbar an der in Abbildung 2.2 dargestellten Nachricht, die allen Chatpartnern angezeigt wird.

[4]Spezielle mathematische Kurvenfunktionen, die elliptischen Kurven, kommen bei der asymmetrischen Verschlüsselung zum Einsatz, da sie besondere Eigenschaften bieten, die für die Kryptografie von Vorteil sind. Sie werden, wie z.B. auch der diskrete Logarithmus, als Einwegfunktionen eingesetzt [Mil86].

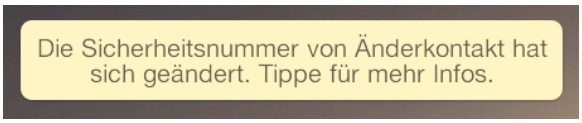


Abbildung 2.2: Hinweis auf neue verschlüsselte Sitzung im WhatsApp-Chat
Quelle: eigene Arbeit

Beim Aufbau einer Verbindung fordert der entsprechende Client den `Identity Key`, den `Signed Pre Key` und einen `One-Time Pre Key` seines Kontaktes beim WhatsApp-Server an. Danach generiert er aus diesen und seinem eigenen `Identity Key` sowie einem weiteren generierten *Curve25519*-Schlüssels das `master_secret` der Verbindung mit Hilfe des ECDH (*Elliptic Curve Diffie-Hellman* [SYS65])-Protokolls.
Aus dem `master_secret` wird schließlich der `Root Key`, daraus der erste `Chain Key` (beide 256 Bit groß) und aus diesem schließlich für jede einzelne Nachricht ein `Message Key` unter Nutzung der Schlüsselableitungsfunktion *HKDF* generiert (mehr zur Verschlüsselung von Nachrichten im folgenden Abschnitt).
Der `Message Key` wiederum ist 80 Byte groß und besteht aus je 32 Byte *AES 256*-Schlüssel zum symmetrischen Verschlüsseln und *HMAC-SHA 256*-Schlüssel zum Signieren der Nachricht sowie einem 16-Byte-Initialisierungsvektor für den *CBC* (Cipher Block Chaining)-Modus.

Da der Kommunikationspartner beim Aufbau der verschlüsselten Sitzung nicht mit dem Internet verbunden sein muss, aber der Absender schon Nachrichten verschicken kann, werden, bis der Empfänger antwortet, an alle gesendeten Nachrichten der `Identity Key` des Absenders sowie der zusätzlich generierte Schlüssel angehängt. So kann der Empfänger zuerst seine eigene verschlüsselte Sitzung aufbauen und die o.g. korrespondierenden Schlüssel generieren.

Abbildung 2.3 gibt eine Übersicht über das in diesem Abschnitt beschriebene Verfahren der Schlüsselgenerierung.

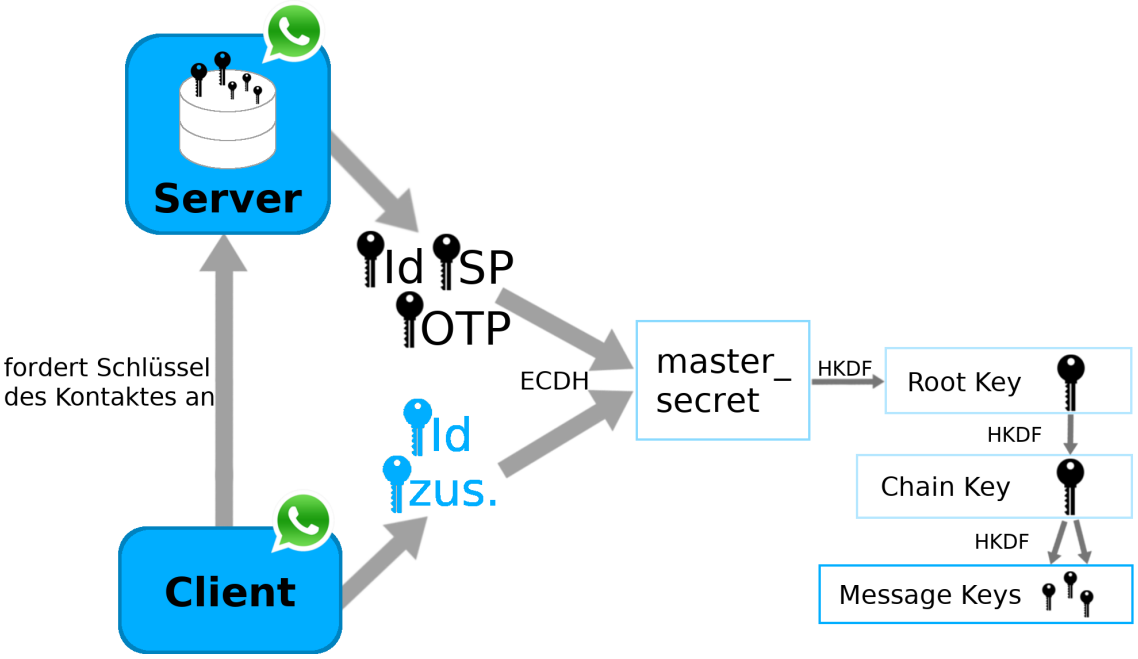


Abbildung 2.3: Schematischer Ablauf des Aufbaus einer verschlüsselten Sitzung
Symbole: `Id` – Identity Key, `SP` – Signed Pre Key, `OTP` – One-Time Pre Key, `zus.` – zusätzlich generierter Schlüssel
Quelle: eigene Arbeit, Icons von `pngimg.com` und `Freepik auf www.flaticon.com`

2.2.3.3 Versand einer Textnachricht

Jede Nachricht wird mit einem eigenen `Message Key` unter Verwendung des *AES 256*-Verfahrens im *CBC*-Modus verschlüsselt und mit dem *HMAC-SHA 256*-Schlüssel signiert. Des Weiteren wird für jede Nachricht ein öffentlicher *Curve25519*-Schlüssel bereitgestellt.
Der `Message Key` wird stets von dem aktuellen `Chain Key` abgeleitet, wobei nach einer solchen Ableitung sofort eine Aktualisierung des `Chain Keys` erfolgt. Dadurch unterliegt dieser einer ständigen Veränderung und von einem `Message Key` kann nicht auf zurückliegende Werte des `Chain Keys` geschlossen werden. Diese Eigenschaft wird „Forward Secrecy“ [Wha17] genannt.
Der Empfänger der Nachricht kann sie mit seinem eigenen passenden `Message Key` entschlüsseln und die Signatur überprüfen.

Antwortet dieser auf die letzte Nachricht, wird ein komplett neuer *Curve25519*-Schlüssel ausgehandelt sowie ein neuer `Root Key` und `Chain Key` generiert. Dies geschieht unter Nutzung der *Curve25519*-Schlüssel der aktuellen Nachrichten und des alten `Root Keys`.
Die verwendeten Schlüssel werden also ständig geändert, was zu einer erhöhten Si-

cherheit führt.
Genaue Informationen über die Berechnung der Schlüssel können [Wha17] entnommen werden, ein Überblick über das Ableiten und Aktualisieren der verschiedenen Schlüssel ist in Abbildung 2.4 zu finden.

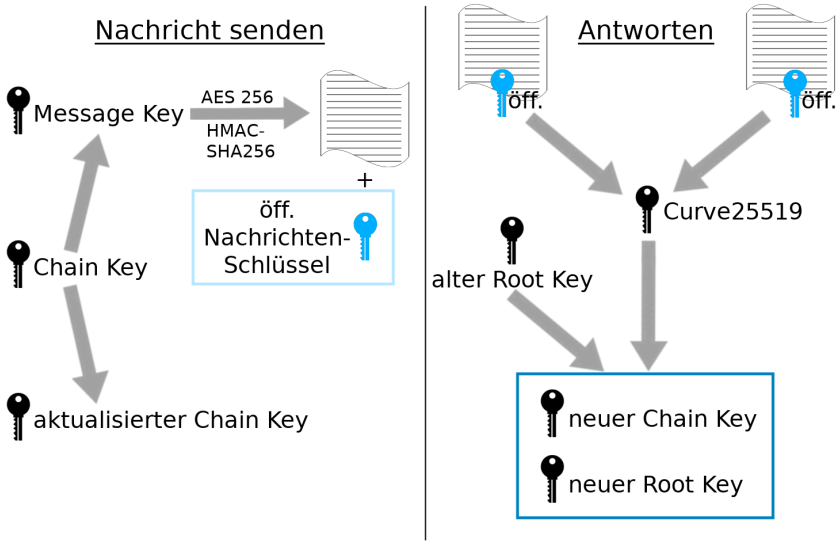
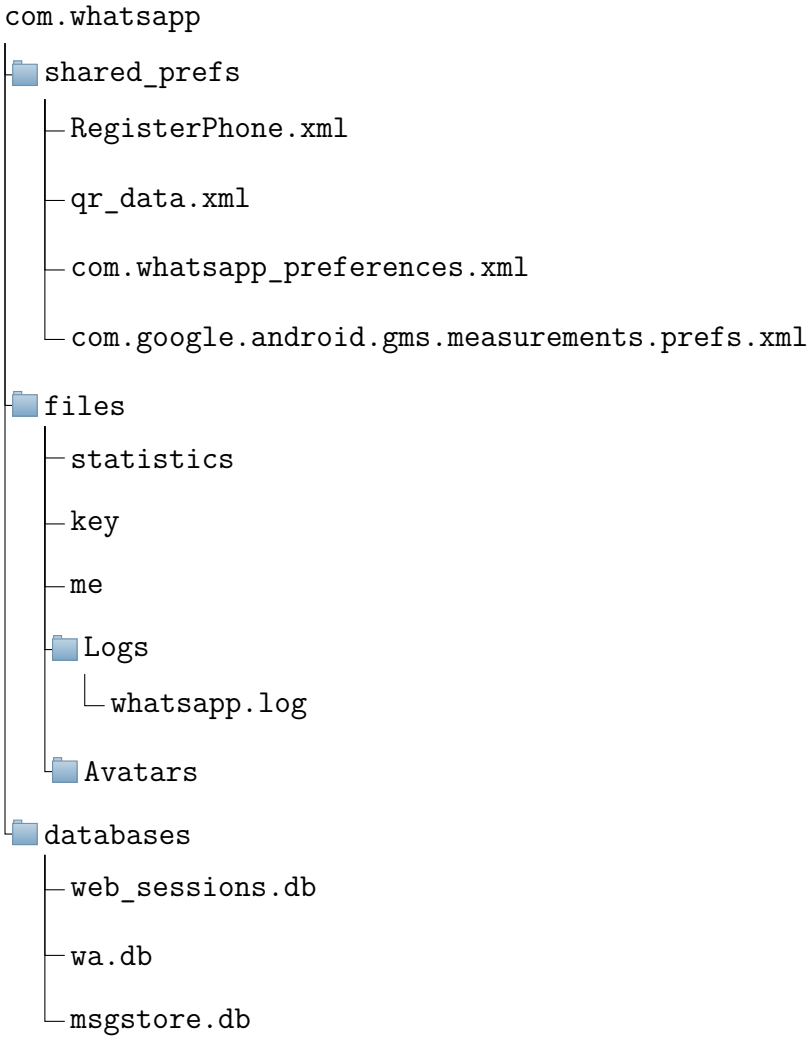


Abbildung 2.4: Schematischer Ablauf der Schlüsselableitung beim Versand von Textnachrichten
Quelle: eigene Arbeit, Icons von `Freepik auf www.flaticon.com`

2.2.4 Gespeicherte Daten

Nach der Installation legt WhatsApp verschiedene Dateien im Speicher des Telefons ab. Wird das Nutzerkonto registriert (siehe Abschnitt 2.2.1), kommen weitere hinzu und auch im laufenden Betrieb werden durch den Messenger immer wieder Dateien aktualisiert bzw. neu angelegt.
Im Folgenden werden die Speicherorte aller forensisch relevanten Dateien bzw. Ordner aufgeführt und ihr Zweck erläutert (erweitert und aktualisiert nach [TT15]), wobei beachtet werden muss, dass der Ordner `/data/data` auf der Android-Systempartition liegt und dem „normalen“ Nutzer nicht direkt zugänglich ist. Lediglich der Benutzer *root* hat vollen Zugriff auf alle Daten.
Weitere Möglichkeiten, auch ohne Rootzugriff auf diese Weise geschützte Dateien auszulesen werden in Kapitel 3 diskutiert.

Wichtige Dateien im WhatsApp Daten-Verzeichnis sind:



Weiterhin befinden sich folgende Ordner im internen Speicher bzw. auf der eingesetzten SD-Karte:

- `Media/`
- `Databases/`

2.2.4.1 registration.RegisterPhone.xml

Speicherort	`/data/data/com.whatsapp/shared_prefs/registration.RegisterPhone.xml`
Dateityp	XML

Tabelle 2.2: Eigenschaften der Datei `registration.RegisterPhone.xml`
Quelle: eigene Arbeit

Schon während der Registrierung von WhatsApp auf einem Telefon wird diese Datei erstellt, da sie Informationen über die registrierte Telefonnummer enthält. Darin befinden sich u.A. die in Listing 2.2 sichtbaren XML-Tags mit der Telefonnummer (ohne vorangestellte 0) und dem Ländercode (hier 49 für Deutschland).

```
1 <map>
2   <string name="com.whatsapp.registration.RegisterPhone.↩
  ↪ phone_number">123456789</string>
3   [...]
4   <string name="com.whatsapp.registration.RegisterPhone.↩
  ↪ country_code">49</string>
5 </map>
```

Listing 2.2: Ausschnitt aus der Datei `registration.RegisterPhone.xml`

2.2.4.2 qr_data.xml

Speicherort	`/data/data/com.whatsapp/shared_prefs/qr_data.xml`
Dateityp	XML

Tabelle 2.3: Eigenschaften der Datei `qr_data.xml`
Quelle: eigene Arbeit

Diese Datei wird erstellt, sobald sich das Telefon erstmalig mit WhatsApp Web verbindet. Sie enthält jeweils die Daten, die in dem letzten, für den Aufbau solch einer Verbindung gescannten, QR-Code enthalten waren. Meldet der Nutzer das Telefon von allen WhatsApp-Web-Verbindungen ab, wird der Inhalt dieser Datei gelöscht, allerdings nur, wenn durch die Funktion „Von allen Computern abmelden“ alle Verbindungen gleichzeitig beendet werden.
Die Datei ist insofern interessant, als dass sich anhand der enthaltenen *Browser ID* der Browser identifizieren lässt, der zuletzt für eine neue Verbindung zu WhatsApp Web verwendet wurde, selbst wenn sich der Nutzer auf diesem Browser schon wieder abgemeldet hat (mehr dazu im Abschnitt 2.2.4.10).

2.2.4.3 com.whatsapp_preferences.xml

Speicherort	`/data/data/com.whatsapp/shared_prefs/com.whatsapp_preferences.xml`
Dateityp	XML

Tabelle 2.4: Eigenschaften der Datei `com.whatsapp_preferences.xml`
Quelle: eigene Arbeit

In dieser Datei speichert WhatsApp viele Einstellungen und Metriken ab, die von forensischem Interesse sein können, wobei nicht immer alle Einträge vorhanden sind. Beispiele sind in Listing 2.3 zu sehen. Die *Browser IDs* der letzten WhatsApp Web-Verbindungen werden wieder nur bis zum Betätigen des „Von allen Computern abmelden“-Buttons gespeichert.

```
1  <map>
2    [...]
3    <!-- JID des letzten Gesprächspartners -->
4    <string name="previous_call_peer_id">49123456789@s.↩
   ↪ whatsapp.de</string>
5    [...]
6    <!-- registrierte Telefonnummer -->
7    <string name="registration_jid">49123456789</string>
8    [...]
9    <!-- aktuelle Statusinformation -->
10   <string name="my_current_status">Hey there! I am using ↩
   ↪ WhatsApp.</string>
11   [...]
12   <!-- selbst gewählter Name, der neben der Telefonnummer ↩
   ↪ angezeigt wird -->
13   <string name="push␣name">test name</string>
14   [...]
15   <!-- alle gespeicherten Browser IDs von WhatsApp Web-↩
   ↪ Verbindungen -->
16   <string name="web_session_verification_browser_ids">9↩
   ↪ fEB47sdf86sdf9GIOD7==,EBltB47sEf86s5f9+/O0f==</string>
17   [...]
18 </map>
```

Listing 2.3: Ausschnitt aus der Datei `com.whatsapp_preferences.xml`

2.2.4.4 com.google.android.gms.measurements.prefs.xml

Speicherort	`/data/data/com.whatsapp/shared_prefs/com.google.android.gms.measurements.prefs.xml`
Dateityp	XML

Tabelle 2.5: Eigenschaften der Datei `com.google.android.gms.measurements.prefs.xml`
Quelle: eigene Arbeit

In dieser XML-Beschreibung finden sich Informationen über die Nutzung des Messengers, z.B. die letzte Nutzungsdauer (in Millisekunden) zwischen Starten und Beenden der App oder der Epoch-Zeitstempel des erstmaligen Starts. Die entsprechenden XML-Tags tragen die Namen `time_active` und `first_open_time`. Auch der Zeitstempel des letzten Minimierens findet sich unter `last_pause_time`.

2.2.4.5 statistics

Speicherort	`/data/data/com.whatsapp/files/statistics`
Dateityp	Binär

Tabelle 2.6: Eigenschaften der Datei `statistics`
Quelle: eigene Arbeit

WhatsApp speichert verschiedene Statistiken in dieser Binärdatei ab. Unter anderem werden hier die Zahl der verschickten und empfangenen Text- bzw. Mediennachrichten oder der Statusmeldungen abgelegt. Die Zahlen sind alle 8 Byte groß und werden in der Little-Endian (LE)-Byteordnung abgespeichert. Im Anhang B, Tabelle B.3 sind die Struktur der Datei und damit die Positionen der einzelnen Einträge dargestellt.

2.2.4.6 key

Speicherort	`/data/data/com.whatsapp/files/key`
Dateityp	Binär

Tabelle 2.7: Eigenschaften der Datei `key`
Quelle: eigene Arbeit

Die `key`-Datei enthält verschiedene Informationen, die mit der Verschlüsselung der Datenbank-Backups zusammenhängen (Weiteres dazu im Abschnitt 2.2.4.14).
Die Struktur der Datei wird im Anhang B, Tabelle B.1 dargestellt. Der wichtigste Inhalt ist dabei der Schlüssel (32 Byte lang, ab Offset 0x7E) der AES-verschlüsselten Datenbank, der nach erstmaliger Generierung auf dem WhatsApp-Server für den späteren Gebrauch lokal abgespeichert wird.
Weitere Informationen über den Aufbau der `key`-Datei und die Verschlüsselung können [Dai+17] entnommen werden.

2.2.4.7 me

Speicherort	`/data/data/com.whatsapp/files/me`
Dateityp	Binär

Tabelle 2.8: Eigenschaften der Datei `me`
Quelle: eigene Arbeit

Die registrierte Telefonnummer findet man auch in der Binärdatei `me` wieder (ab Offset 0x65). Diese kann zur Verifizierung dienen oder genutzt werden, wenn andere Dateien nicht zugreifbar sind.

2.2.4.8 whatsapp.log

Speicherort	`/data/data/com.whatsapp/files/Logs/whatsapp.log`
Dateityp	Text

Tabelle 2.9: Eigenschaften der Datei `whatsapp.log`
Quelle: eigene Arbeit

Die `whatsapp.log`-Datei ist eine Textdatei, in die WhatsApp bei jeder Aktion, die es ausführt, einen Eintrag mit entsprechendem Zeitstempel (auf Millisenkunden genau) ablegt. Da anhand dieser Einträge nicht nur die Arbeit der Anwendung, sondern insbesondere auch die Aktivität des Nutzers nachvollzogen werden kann, hat eine solche Logdatei für den Forensiker einen sehr großen Wert.
Außer der aktuellen Logdatei, die `whatsapp.log` heißt, werden ältere Einträge als Backup im *GZIP*-Format komprimiert und im selben Verzeichnis (`files/Logs`) abgelegt. Die Backups umfassen immer ganze Tage (je nach Größe des Logs nur einen oder mehrere Tage) von 00:00:00.000 Uhr bis 23:59:59.999 Uhr und werden nach dem

Datum des letzten Eintrags benannt. Nach einem Schema, welches im Rahmen dieser Arbeit nicht weiter untersucht wurde, werden die ältesten Logdateien jedoch auch wieder gelöscht, sodass immer nur die aktuellsten Archive vorliegen.

Nachfolgend ist der generelle Aufbau einer Zeile im Log zu sehen, wobei `lvl` für das Loglevel (z.B. `I` für Information oder `W` für Warnung) sowie `TID` für die Thread ID steht und jeweils einer der drei Buchstaben `D`, `M` oder `W` vorkommt:

```
YYYY-MM-DD HH-mm-SS.sss LL_<lvl> <D/M/W> [<TID>:Thread] <Nachricht>
```

Um von Einträgen in der Logdatei auf reale Nutzeraktivitäten schließen zu können, wird sie im weiteren Verlauf der Arbeit analysiert und in Abschnitt 4.2.1 Verbindungen zwischen einzelnen Zeilen und Ereignissen hergestellt. Unter anderem gibt das Log Hinweise auf das Schreiben, Empfangen und Löschen einer Nachricht.

2.2.4.9 Avatars

Speicherort	`/data/data/com.whatsapp/files/Avatars`
Dateityp	Verzeichnis mit Bilddateien

Tabelle 2.10: Eigenschaften des Verzeichnisses `Avatars`
Quelle: eigene Arbeit

Hierbei handelt es sich um ein Verzeichnis, das mehrere JPEG-Bilddateien mit der Endung `.j` enthält. Dies sind die aktuellen Profilfotos aller WhatsApp-Kontakte, die jeweils 96x96 Pixel groß sind und als Dateinamen die jeweilige *JID* besitzen. Auch Profilbilder von WhatsApp-Gruppen finden sich in diesem Ordner, benannt nach dem in Abschnitt 2.2.2.2 erklärten Schema. Ändert ein Kontakt sein Foto oder wird das Gruppenbild gewechselt, wird die Aktualisierung auch in diesem Ordner vorgenommen.

2.2.4.10 web_sessions.db

Speicherort	`/data/data/com.whatsapp/databases/web_sessions.db`
Dateityp	SQLite

Tabelle 2.11: Eigenschaften der Datei `web_sessions.db`
Quelle: eigene Arbeit

Die Datenbank beinhaltet Informationen über Verbindungen zwischen dem Telefon und dem WhatsApp Web-Client. Sie wird erstellt, sobald die erste Verbindung aufgebaut wird (durch Scannen des angezeigten QR-Codes mit dem Telefon) und enthält immer nur die Verbindungen, die auch in der App sichtbar sind.
Die Datenbank enthält drei Tabellen: `android_metadata`, `sqlite_sequence` und `sessions`, wobei vor allem Letztere interessant ist. Sie enthält die Spalten `_id`, `browser_id`, `secret`, `token`, `os`, `browser_type`, `lat`, `lon`, `accuracy`, `place_name`, `last_active`, `timeout`, `expiration` und `fservice`.

Nicht immer werden alle Eigenschaften eines Eintrags ausgefüllt, aber auf jeden Fall ist immer die `browser_id` vorhanden, mit der sich eine Verbindung einem bestimmten Webbrowser eindeutig zuordnen lässt, denn im lokalen Speicher des Browsers[5] ist dessen von WhatsApp vergebene *Browser ID* abgelegt. Einen weiteren Hinweis auf einen verbundenen Browser geben die Spalten `os` und `browser_type`, wobei beachtet werden muss, dass deren Inhalt aus dem an WhatsApp Web übermittelten *User Agent* kommt, der sehr leicht gefälscht werden kann.
`secret` und `token` werden von WhatsApp Web genutzt, um eine sichere Datenübertragung zu ermöglichen und anhand des Epoch-Zeitstempels in der Spalte `last_active` kann bestimmt werden, wann das Telefon zuletzt mit diesem Browser verbunden war. Es muss allerdings berücksichtigt werden, dass ein realer Browser über die Zeit mitunter verschiedene *Browser IDs* haben kann, z.B. wenn die gesetzten Cookies gelöscht werden.

Ein Beispiel für den Inhalt der Tabelle `sessions` ist im Anhang A, Abbildung A.1 zu finden.

2.2.4.11 wa.db

Speicherort	`/data/data/com.whatsapp/databases/wa.db`
Dateityp	SQLite

Tabelle 2.12: Eigenschaften der Datei `wa.db`
Quelle: eigene Arbeit

[5] Webseiten können im lokalen Speicher des Browsers, mit dem sie besucht werden, Daten, meist Schlüssel-Wert-Paare, ablegen. Je nach Browser ist der lokale Speicher an einem unterschiedlichen Speicherort zu finden. Bei Firefox 61.0.1 ist er z.B. in der SQLite-Datenbank `webappsstore.sqlite` im aktuellen Profilverzeichnis (bei Windows unter `%userprofile%\AppData\Roaming\Mozilla\Firefox\Profiles` bzw. bei Linux im Ordner `~/.mozilla/firefox`) zu finden.

In der Datenbank `wa.db` speichert WhatsApp alle Informationen zu den vorhandenen Kontakten und Gruppen. Die wichtigste Tabelle heißt `wa_contacts` und hat insgesamt 26 Spalten. Da nicht alle davon forensisch relevant sind, werden im Folgenden die bedeutendsten Informationen beschrieben.

`jid`	Dies ist die *JID* des Kontaktes, die nach dem in Abschnitt 2.2.2.2 genannten Schema aufgebaut ist. Auch, wenn der Kontakt selbst nicht WhatsApp benutzt, wird eine *JID* eingetragen.
`is_whatsapp_user`	Die Information, ob der Kontakt WhatsApp benutzt, findet sich 0-1-codiert in dieser Spalte.
`status`	Hier steht die aktuelle Statusinformation des Nutzers.
`status_timestamp`	In dieser Spalte steht ein Epoch-Zeitstempel der letzten Statusänderung (oder 0, wenn der Kontakt kein WhatsApp benutzt).
`number`	Die Telefonnummer des Kontaktes wird so eingetragen, wie sie in den Telefonkontakten vorgefunden wird. Im Falle einer WhatsApp-Gruppe findet man stattdessen die *JID* des Erstellers.
`display_name`	In dieser Spalte findet sich der Name des Kontaktes, wie er auch im Kontaktverzeichnis des Telefons steht bzw. der aktuelle Name der Gruppe.

2.2.4.12 msgstore.db

Speicherort	`/data/data/com.whatsapp/databases/msgstore.db`
Dateityp	SQLite

Tabelle 2.13: Eigenschaften der Datei `msgstore.db`
Quelle: eigene Arbeit

Diese Datenbank ist die zentrale Stelle für die Speicherung aller gesendeten und empfangenen Nachrichten. Damit WhatsApp gesamte Chatverläufe auch anzeigen kann, wenn das Telefon nicht mit dem Internet verbunden ist und weil sie erst beim Empfänger entschlüsselt werden (siehe Abschnitt 2.2.3), speichert der IM-Dienst alle Nachrichten lokal ab.
Die SQLite-Datenbank besteht aus vielen Tabellen, die relevantesten sind im Folgenden aufgeführt.

`chat_list` Diese Tabelle spiegelt alle auf dem Telefon vorhandenen Konversationen wider, sowohl Einzel- als auch Gruppenchats. Zu jeder *JID* (in der Spalte `key_remote_jid`) finden sich verschiedene Informationen, z.B. ob der Chat archiviert wurde (in der Spalte `archived`) oder die Anzahl ungelesener Nachrichten (`unseen_message_count`).
Bei Gruppenchats steht hier außerdem auch der Gruppenname (`subject`) und der Erstellungszeitstempel (`creation`).

`group_participants` Hier stehen zu jeder Gruppe, an der der Nutzer teilnimmt, alle aktuellen Mitglieder. Auch die Administratoren der Gruppe sind vermerkt.

`messages` In dieser Tabelle werden alle Nachrichten mit zugehörigen Metadaten gespeichert. Da sie sehr wichtig ist, wird im folgenden Abschnitt weiter darauf eingegangen.

`status_list` Informationen über Statusmeldungen finden sich hier. Einer `key_remote_jid` werden hier unter anderem der Epoch-Zeitstempel der letzten Meldung (`timestamp`), die Anzahl der ungesehenen Status (`unseen_count`) und die Gesamtanzahl der aktuell verfügbaren Status (`total_count`) zugeordnet.

Die Tabelle `messages` Zu jeder Nachricht gibt es einen Eintrag in dieser Tabelle, selbst zu den automatisch von WhatsApp angezeigten wie z.B. verpasste Anrufe. Sie sind allerdings nicht nach Chats, sondern nach dem Zeitstempel sortiert. In der folgenden Übersicht werden die wichtigsten Spalten erläutert.

`key_remote_jid` Hier steht die *JID* des Kommunikationspartners bzw. der Gruppe.

`key_from_me` Der Wert in dieser Spalte zeigt an, ob die Nachricht gesendet oder empfangen wurde.

`data` Bei Textnachrichten findet sich hier der Nachrichteninhalt.

`timestamp` Dabei handelt es sich um den Epoch-Zeitstempel des Versendens (im Falle einer ausgehenden Nachricht) bzw. des Empfangens (bei eingehenden Nachrichten).

`media_wa_type` Die Zahl in dieser Spalte beschreibt den Typ der Nachricht. Dabei konnten bisher folgende Typen zugeordnet werden.
0 – Textnachrichten, 1 – Bilder, 2 – Sprachnachrichten, 3 – Videos,

	8 – Anrufe (auch wenn diese nicht im Chatverlauf auftauchen), 9 – Dokumente, 10 – Nachrichten zu verpassten Anrufen, 15 – leerer Eintrag für Nachrichten, die mit der Funktion „Für alle löschen“ gelöscht wurden.
`media_caption`	Falls eine Mediendatei mit Beschriftung verschickt wurde, wird diese hier vermerkt.
`thumb_image`	Diese Spalte enthält (sofern verfügbar) Binärdaten, in denen sich oft nützliche Informationen finden lassen. Beispiele dafür sind ein verkleinertes Vorschaubild und der Speicherort bei Bildern, ein textueller Hinweis auf einen Wechsel der Telefonnummer, das Hinzufügen bzw. Entfernen von Mitgliedern oder das Ändern von Profilbildern in Gruppenchats.
`remote_resource`	Im Falle von Gruppenchats steht hier der Autor der Nachricht.

2.2.4.13 Media

Speicherort	`/sdcard/WhatsApp/Media`
Dateityp	Verzeichnis mit Unterverzeichnissen

Tabelle 2.14: Eigenschaften des Verzeichnisses `Media`
Quelle: eigene Arbeit

Im Ordner `Media` werden alle gesendeten und empfangenen Mediendateien, Dokumente und Sprachnachrichten sowie temporär verfügbare Statusmeldungen abgespeichert. Für jedes Dateiformat findet sich deshalb ein Unterordner mit dem entsprechenden Namen, z.B. `WhatsApp Audio` oder `WhatsApp Images`. Darin wiederum gibt es einen Ordner namens `Sent`.
Eine Ausnahme dazu stellen allerdings die Sprachnachrichten dar, bei denen für jede Kalenderwoche ein Unterordner im Verzeichnis `WhatsApp Voice Notes` erstellt wird, worin dann alle gesendeten und empfangenen Sprachnachrichten dieser einen Woche gemeinsam gespeichert werden.
Nimmt man Bilder oder Videos direkt in der Messenger-App auf, werden diese im entsprechenden Medienordner (z.B. `WhatsApp Video`) gespeichert. Werden sie dann verschickt, erfolgt eine zusätzliche Speicherung im `Sent`-Unterordner. Bricht man den Vorgang allerdings vor dem Senden ab, wird die Datei verworfen. Vor dem Versenden skaliert WhatsApp die Bilder, um Datenverkehr einzusparen. Dabei beträgt die maximale Breite und Höhe unter Beibehaltung der Seitenverhältnisse je 1280 Pixel.

Der Dateiname von Mediendateien enthält in der Regel einen Präfix für den Dateityp[6], das Erstellungsdatum und eine täglich neu hochzählende Nummer, z.B. `IMG-20180501-WA0008.jpg`. Eine Ausnahme dazu bilden verschickte Dokumente, die den originalen Dateinamen behalten.

Statusmeldungen, die als Foto vorliegen, speichert WhatsApp auf dem Telefon im Ordner `Media/.Statuses` mit einem Dateinamen ab, der aus 32 Hexadezimalzahlen besteht. Normalerweise werden Status nach 24 Stunden den Kontakten nicht mehr angezeigt und auch die Datei auf den jeweiligen Geräten wird wieder gelöscht. Allerdings wurden auch permanent gespeicherte Statusfotos entdeckt.

2.2.4.14 Databases

Speicherort	`/sdcard/WhatsApp/Databases`
Dateityp	Verzeichnis mit Datenbank-Backups

Tabelle 2.15: Eigenschaften des Verzeichnisses `Databases`
Quelle: eigene Arbeit

In diesem Verzeichnis befinden sich AES-verschlüsselte Backups der Nachrichtendatenbank `msgstore.db`. Das Verfahren zur Verschlüsselung wurde im Laufe der Zeit immer wieder geändert (erkennbar an der Dateiendung), wobei in dieser Arbeit auf die aktuelle Version Bezug genommen wird, die den AES im *Galois/Counter-Mode* (GCM) verwendet. Dabei erfolgt die Verschlüsselung mit einem 256-Bit-Schlüssel, der bei der Installation der App festgelegt wird [Dai+17] (im Gegensatz zu früheren Versionen, bei denen auf jedem Gerät immer der gleiche Schlüssel zum Einsatz kam [CSW12]).
In der Datei werden die verschlüsselten Daten von einem 67 Byte großen Header und einem 20 Byte großen Trailer eingeschlossen. Diese Struktur wird im Anhang B, Tabelle B.2 dargestellt.
Es gibt stets ein jüngstes Backup mit dem Namen `msgstore.db.crypt12` und mehrere ältere Versionen, die das Datum der Sicherung im Dateinamen enthalten. Nach einer bestimmten Zeit werden die ältesten Dateien auch wieder gelöscht, wegen fehlender Relevanz wurde das zugrundeliegende Schema im Rahmen dieser Arbeit allerdings nicht weiter untersucht.

Mehr Informationen zur Verschlüsselung der Datenbanken und zur Wiederherstellung von historischen Nachrichtenbeständen finden sich im Abschnitt 3.1.4.

[6]Die Präfixe der Mediendateien sind `IMG` für Bilder, `VID` für Videos oder GIF-Animationen und `PTT` für Sprachnachrichten bzw. `AUD` für Audiodateien

3 Methoden

3.1 Extraktion von WhatsApp-Daten

Am Beginn jeder digitalforensischen Untersuchung steht eine (möglichst) unveränderte Sicherung der betroffenen Originaldaten. Im Idealfall handelt es sich dabei um eine vollständige physische Sicherung des gesamten Datenträgers inklusive aller Sektoren, die keine oder gelöschte Daten enthalten. Durch den Einsatz eines sog. *Writeblockers* werden dabei alle Schreibzugriffe verhindert [BSI11].

Für klassische Datenträger wie Festplatten, Solid State Drives (SSDs) oder Speicherkarten ist dies in der Regel auch möglich.
Bei Smartphones oder anderen Mobilgeräten, wie z.B. Smartwatches, die Daten auf internen Speicherchips ablegen, ist oft schon der Zugriff auf diese Daten eingeschränkt bzw. nur mit erhöhtem Aufwand möglich. Entsprechend sind forensische Datensicherungen in diesem Fall meist zwangsläufig mit einer Veränderung von Daten verbunden. Außerdem ist eine physische Kopie des gesamten Speichers nicht immer möglich, weshalb immer wieder auf alternative Methoden zurückgegriffen werden muss, um fallrelevante Daten zu extrahieren.

In diesem Abschnitt werden deshalb verschiedene Möglichkeiten aufgezeigt, wie forensisch relevante WhatsApp-Daten (z.B. die in Abschnitt 2.2.4 genannten) ohne physische Sicherung aus Mobiltelefonen extrahiert werden können, eine weitere Methode zum Extrahieren der Nachrichtendatenbank kann zudem im Abschnitt 5.2.3 gefunden werden.
Dabei ist allerdings immer *mindestens der Zugang zum Gerät notwendig* (entsperrter Bildschirm). Da aber die Veränderung der Daten so gering wie möglich gehalten werden soll, wird bei allen genannten Methoden auf das „Rooten“, d.h. das Erlangen von Superuser-Rechten mit uneingeschränktem Dateizugriff, verzichtet.
Einige dieser Methoden wurden in dieser Arbeit genutzt, um die benötigten Dateien für die Analyse der Logdateien (siehe Abschnitt 3.2.1) und das Erstellen des entsprechenden Programms zum Verarbeiten (siehe Abschnitt 3.2.2) zu erhalten.

Für jede Methode wurde beispielhaft eine Extraktion der betroffenen WhatsApp-Daten von einem Testgerät vorgenommen, auf dem der IM-Dienst installiert und

verschiedene Aktivitäten (Nachrichten versenden und empfangen, Anrufe tätigen, Statusmeldungen veröffentlichen etc.) durchgeführt wurden, um Beispielinhalte zu erzeugen, wobei die folgende Testumgebung zum Einsatz kam:

Testgerät

Modell:	ZTE Blade L110
Betriebssystem:	Android 5.1
CPU:	ARM Cortex-A7
RAM:	512 MB
WhatsApp-Version:	2.18.191 (Juni 2018)

Auswerterechner 1

Modell:	Acer Aspire XC-115
Betriebssystem:	Windows 8.1
CPU:	AMD A4-6210
RAM:	4,00 GB

Auswerterechner 2

Modell:	mh SERVICE OxyCube
Betriebssystem:	Windows 8.1
CPU:	Intel Core i7-6700K
RAM:	16,00 GB

Die entstandenen Ergebnisse werden in Abschnitt 4.1 aufgeführt.

3.1.1 Export von Chatverläufen als Textdatei

WhatsApp bietet dem Nutzer die Möglichkeit, über eine eingebaute Funktion einen einzelnen „Chat [zu] exportieren“[1].
Die hier beschriebene Methode bedient sich dieser Funktion, um Chatverläufe zu extrahieren. Im Folgenden wird beschrieben, wie ein Export über E-Mail durchgeführt

[1] Für den Export werden verschiedene, auf dem Gerät verfügbare Varianten angeboten, z.B. als WhatsApp-Nachricht oder E-Mail, über Bluetooth oder als Datei mit Hilfe eines installierten Dateimanagers.
Diese generelle Exportmöglichkeit ist ab Version 2.18.178 verfügbar, davor war es nur möglich, den Chatverlauf als E-Mail zu exportieren.

werden kann, da diese Möglichkeit auch bei älteren WhatsApp-Versionen verfügbar ist. Weitere Möglichkeiten können jedoch, sofern vorhanden, mit weniger Aufwand und Datenveränderung verbunden sein.

Um vom Internet getrennt zu bleiben und keine fallbezogenen Daten auf fremde Server (der E-Mail-Anbieter) hochladen zu müssen, wird hier insbesondere auf die Nutzung eines eigenen Mailservers eingegangen.

Voraussetzungen Im beschriebenen Szenario wird ein eigenes Netzwerk-Setup benötigt, das die folgenden Komponenten umfasst:

- WLAN-fähiger Router, der nicht mit dem Internet verbunden ist
- Computer mit installiertem Mailserver[2]
- Telefon $\mathcal{T}_{WA}$ mit zu extrahierenden Nachrichtenverläufen (Verbindung über WLAN)
- ein weiteres Gerät, mit dem die versandte E-Mail vom Server herunter geladen wird, z.B. ein Auswerterechner

Vorgehen Nachdem das Netzwerk wie beschrieben aufgebaut wurde, muss das entsprechende E-Mail-Konto auf dem Gerät $\mathcal{T}_{WA}$ eingerichtet werden. Dabei hat sich gezeigt, dass die Einrichtung in Androids Standard-E-Mail-App am einfachsten war. Nach der Einrichtung kann der Nachrichtenverlauf über die WhatsApp-Funktion „Chat exportieren“[3] im gewünschten Chat auf dem Telefon $\mathcal{T}_{WA}$ versandt werden.
Dabei kann ausgewählt werden, ob zu dem Chat gehörige Mediendateien angehängt werden sollen, wobei allerdings nicht der gesamte Chatverlauf exportiert wird. Alle benötigten Mediendateien können später auch manuell aus dem Verzeichnis `/sdcard/WhatsApp/Media/` kopiert werden (siehe auch Abschnitt 2.2.4.13), müssen aber auch per Hand den entsprechenden Nachrichten zugeordnet werden.
Die versandte E-Mail kann später mit einem weiteren Gerät vom Mailserver heruntergeladen werden.

Ergebnis Mit dieser Methode erlangt man eine Klartextdatei mit allen Nachrichten des gewählten Chats, die WhatsApp standardmäßig `WhatsApp Chat mit <Kontakt-`

[2]Der Mailserver muss in der Lage sein, E-Mails von einem auf dem Telefon registrierten Konto zu empfangen und an ein anderes Konto auszuliefern. Zum Testen wurde z.B. der Open-Source-Server „Citadel“ [Unc18] unter Linux Raspbian [Ras18] auf einem Raspberry Pi mit mehreren eingerichteten E-Mail-Konten genutzt.

[3]Die Funktion ist bei geöffneten Chatverlauf erreichbar über den Eintrag „Mehr“ im Dreipunkt-Menü.

name>.txt nennt, sowie möglicherweise weitere an die E-Mail angehängte Dateien. Deren Anzahl ist aber durch die maximale, vom E-Mail-Server bzw. der Clientsoftware festgelegte, Größe einer E-Mail begrenzt.
Die Inhalte der Textdatei können entweder direkt ausgewertet oder z.B. mit Hilfe eines Skriptes für die Auswertung in eine übersichtliche Form gebracht werden. Im Rahmen dieser Arbeit wurde zu diesem Zweck das Python3-Skript `wa_txt2html.py` mit dem Quellcode aus Listing C.2 (Anhang C) entwickelt, welches den textuellen Inhalt in eine HTML-Seite umwandelt und die Nachrichten (wie bei Chatprogrammen üblich) auf verschiedene Seiten anordnet. Das Skript kann mit dem folgenden Kommandozeilenbefehl gestartet werden, wobei alle Optionen bis auf die Eingabedatei optional sind und der Kontaktname der Person angegeben werden kann, aus dessen Sicht der Chat dargestellt wird.

```
python wa_txt2html.py -i <Textdatei> -o <HTML-Datei> -m <Kontaktname>
```

Des Weiteren ist eine Hilfeseite verfügbar, die mit dem folgenden Kommando aufgerufen werden kann:

```
python wa_txt2html.py -h
```

3.1.2 Nachrichtenwiederherstellung durch Neuregistrierung

Wenn ein Nutzer WhatsApp auf seinem Telefon deinstalliert, ohne die Anwendungsdaten zu löschen, und die App später erneut aufspielt, bietet es nach der Registrierung an, die vorhandenen Chats wiederherzustellen.
Dabei ist das Vorhandensein des verschlüsselten Nachrichtenbackups (also der Datei `/sdcard/WhatsApp/Databases/msgstore.db.crypt12`) für eine funktionierende Wiederherstellung ausreichend. Die zur Entschlüsselung notwendige `key`-Datei holt sich der Messengerdienst auf Grundlage der Zusatzinformationen in der Backup-Datei vom WhatsApp-Server.

Diese Tatsache wird in der hier beschriebenen Methode ausgenutzt. Dabei wird allerdings das frei zugängliche Backup vom auszulesenden Gerät $\mathcal{T}_{WA}$ auf ein weiteres reales bzw. auf dem PC emuliertes Android-Gerät kopiert.
Auf diese Weise erhält man Zugriff auf ein komplett registriertes WhatsApp-Konto in einer kontrollierbaren Umgebung. Es kann also auf alle gewünschten (inklusive der geschützten) Daten zugegriffen werden, sofern diese Umgebung dies zulässt, wie es beim beschriebenen Fall des Android-Emulators möglich ist.

Voraussetzungen Je nachdem, wie die Methode angewandt wird, sind verschiedene Komponenten notwendig.
Darf $\mathcal{T}_{WA}$ keine Verbindung zum Mobilfunknetz herstellen, wird ein weiteres Telefon benötigt, um die SMS mit dem Registrierungscode zu empfangen. In diesem Fall muss zwingend die PIN der SIM-Karte bekannt sein, andernfalls kann $\mathcal{T}_{WA}$ selbst dazu genutzt werden und wenn das Gerät bereits eingeschaltet ist, ist die SIM-PIN evtl. nicht mehr notwendig.
In jedem Fall ist ein Android-Gerät notwendig, auf dem das WhatsApp-Konto registriert wird. Dies kann entweder ein reales Smartphone sein, oder, wie im Folgenden angenommen, ein Android-Emulator (nachfolgend $\mathcal{T}_{Emu}$ genannt) auf einem klassischen Computer. Im Testfall wurde der in der App-Entwicklungsumgebung „Android Studio“ [Goo18b] enthaltene Emulator sowie der Auswerterechner 2 (siehe Seite 28) verwendet, da für die Hardwarebeschleunigung der Emulation ein Intel-Prozessor notwendig ist.

Vorgehen Zuerst muss sichergestellt werden, dass $\mathcal{T}_{Emu}$ richtig auf den Prozess vorbereitet ist. Dazu zählt im Einzelnen:

- WhatsApp ist nicht installiert
- es befinden sich keine alten WhatsApp-Daten mehr auf dem Telefon
- der Ordner für verschlüsselte Nachrichtenbackups `/sdcard/WhatsApp/Databases` ist vorhanden

Wenn die Registrierungs-SMS mit einem anderen Telefon empfangen werden soll, muss die SIM-Karte aus $\mathcal{T}_{WA}$ dazu in das zweite Gerät eingesetzt werden.
Nachdem die gewünschte Backupdatei[4] von $\mathcal{T}_{WA}$ auf $\mathcal{T}_{Emu}$ in den o.g. `Databases`-Ordner kopiert wurde, wird WhatsApp auf dem Zieltelefon installiert.

Bei der Registrierung muss zwangsläufig die gleiche Telefonnummer wie auf $\mathcal{T}_{WA}$ genutzt werden. Nachdem der sechsstellige Code auf dem gewünschten Gerät empfangen und in die App eingetragen wurde, sucht WhatsApp automatisch nach vorhandenen Backups. Die danach präsentierte und in Abbildung 3.1 dargestellte Meldung bestätigt man mit „Wiederherstellen“.

Ergebnis Nachdem das verschlüsselte Backup, wie oben beschrieben, wiederhergestellt wurde, können alle gewünschten Daten von $\mathcal{T}_{Emu}$ kopiert und ausgelesen werden, da der geschützte Speicherbereich im Emulator zugänglich ist.

[4] Sowohl die Verwendung und Wiederherstellung des neusten Backups `msgstore.db.crypt12` als auch von älteren Versionen, z.B. `msgstore-2018-06-01.1.db.crypt12` ist möglich.

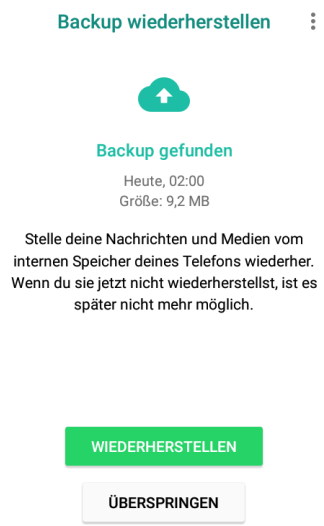


Abbildung 3.1: präsentierte Meldung bei gefundenem Backup
Quelle: eigene Arbeit

Dabei muss allerdings berücksichtigt werden, dass auf dem neuen Gerät keine Kontaktinformationen verfügbar sind, WhatsApp also nur die Telefonnummern anzeigt.

Die unverschlüsselte Nachrichtendatenkbank lässt sich z.B. über die *Android Debug Bridge*[5] mit dem folgenden Kommandozeilenbefehl im aktuellen Verzeichnis abspeichern:

```
adb pull /data/data/com.whatsapp/databases/msgstore.db
```

3.1.3 Komplettes App-Backup über ADB

Diese Methode bedient sich der Möglichkeit, über die Android Debug Bride ein komplettes Backup aller Daten einer bestimmten Anwendung zu erstellen. Die Erlaubnis dafür kann jedoch die jeweilige App selbst verwalten, WhatsApp entzieht sie ab Version 2.11.444 und liefert ein leeres Archiv, wenn ein Backup angefordert wird.

Da sich über ADB allerdings auch Apps installieren lassen, gibt es die Möglichkeit, die vorhergehende WhatsApp-Version (2.11.431) zu installieren (*Downgrade*[6]) und

[5] Die Android Debug Bridge (ADB) dient zur Kommunikation zwischen einem Computer und einem Android-Gerät. Sie ist hauptsächlich für die Entwicklung von Apps vorgesehen und erlaubt die Steuerung des Mobilgerätes über verschiedene Kommandos. Damit ein über USB angeschlossenes Gerät auf diese Weise angesprochen werden kann, muss auf diesem jedoch „USB-Debugging“ in den Entwickleroptionen aktiviert sein. Mehr über die ADB und USB-Debugging kann [Goo18a] und [TT15] entnommen werden.

[6] App-Downgrades über ADB werden ab Android 7 zurückgewiesen, wodurch diese Methode nicht mehr funktioniert. Mehr dazu im Abschnitt 4.1.5.

das Backup dennoch zu bekommen.
Der Prozess des Downgrades und Backups mit den notwendigen Vorbereitungen wird in einem Shellskript automatisiert, das der Nutzer „EliteAndroidApps“ auf GitHub veröffentlicht hat (zu finden unter [Eli16b]). Im Folgenden wird mit Hilfe dieses Skripts, das für Windows und Linux verfügbar ist, ein komplettes Backup der WhatsApp-Daten erstellt.

Voraussetzungen Sowohl auf dem Telefon, von dem die WhatsApp-Daten extrahiert werden sollen ($\mathcal{T}_{WA}$), als auch auf dem Rechner, auf dem das Shellskript ausgeführt wird, müssen einige Bedingungen gegeben sein:

- $\mathcal{T}_{WA}$ hat USB-Debugging aktiviert
- minimale Android-Version: 4.0
- auf dem Rechner installierte Software: Java, ADB und die ADB Treiber für $\mathcal{T}_{WA}$

Vorgehen Prinzipiell ist der gesamte Prozess unter Nutzung des „WhatsApp Key/ DB Extractor“-Skriptes sehr einfach. Da dieses aber standardmäßig nur einige wenige Dateien[7] aus dem Archiv speichert und danach das gesamte Backup automatisch löscht, muss es zuvor zur kompletten Extraktion aller Daten manuell angepasst werden. Dazu wird eine Zeile auskommentiert, was beispielhaft für die Linux-Version (Bourne Again Shell) der Datei in Listing 3.1 gezeigt wird.

```
133 echo -e "Restore␣complete\n\nCleaning␣up␣temporary␣files␣..."
134 rm tmp/whatsapp.ab
135 # rm tmp/whatsapp.tar (Diese Zeile auskommentieren)
136 rm -rf tmp/apps
137 rm tmp/$apkname
138 echo -e "Done\n\nOperation␣complete\n"
```

Listing 3.1: Ausschnitt aus der Datei `WhatsAppKeyDBExtract.sh` nach Bearbeitung

Danach kann das Skript gestartet und den Anweisungen gefolgt werden. Unter Linux muss es dazu ausführbar sein, was z.B. durch das folgende Kommando erreicht wird.

```
chmod a+x WhatsAppKeyDBExtract.sh
```

Dabei muss beachtet werden, dass das Backup auf $\mathcal{T}_{WA}$ bestätigt werden muss, und

[7] Das Skript speichert nur die Dateien `key`, `msgstore.db`, `wa.db`, `axolotl.db` und `chatsettings.db` im Unterordner `extracted` ab.

das Programm abbricht, wenn dies zu lange dauert.

Zur Erläuterung werden nachfolgend die einzelnen Arbeitsschritte der Shelldatei zusammengefasst.

1. Backup der installierten Version $\mathcal{V}_{inst}$ von WhatsApp
2. Installation von WhatsApp 2.11.431 (ab Android 6 muss zuvor die vorhandene Installation entfernt werden, ohne die Anwendungsdaten zu löschen)
3. Backup der WhatsApp-Daten über ADB
4. Extraktion der Dateien aus dem Backup als tar-Archiv (unter Nutzung des Java-Programms „Android Backup Extractor“ von GitHub-Nutzer „nelenkov“ [Ele18])
5. Entpacken einiger Dateien (siehe Fußnote 7) und Löschen aller temporären Dateien (wenn nicht auskommentiert, wie in Listing 3.1)
6. Wiederherstellen der WhatsApp-Version $\mathcal{V}_{inst}$

Ergebnis Nachdem der Vorgang beendet wurde, finden sich alle extrahierten Daten im Archiv `tmp/whatsapp.tar` wieder. Darin befindet sich der WhatsApp-Daten-Ordner `apps/com.whatsapp`, der alle in Abschnitt 2.2.4 erwähnten Dateien enthält. Dabei sind die Namen der Unterverzeichnisse allerdings abgekürzt[8].

3.1.4 Manuelle Entschlüsselung des Nachrichtenbackups

Liegen die verschlüsselten Backups der Nachrichtendatenbank `msgstore.db` vor und hat man Zugriff auf die zugehörige `key`-Datei, können die Dateien auch „per Hand“ entschlüsselt werden. Dadurch lassen sich auch historische Nachrichtenbestände wiederherstellen, in denen möglicherweise noch Nachrichten vorhanden sind, die später gelöscht wurden.
Nachdem der geheime Schlüssel aus der Datei `key` extrahiert wurde, kann das Backup entschlüsselt und danach entpackt werden, um an die lesbare SQLite-Datenbank zu gelangen.

[8] In der verkürzten Schreibweise steht `sp` für `shared_prefs`, `f` für `files` und `db` für `databases`.

Voraussetzungen Neben den benötigten Dateien (`/sdcard/WhatsApp/Databases/msgstore*.db.crypt12` und `/data/data/com.whatsapp/files/key`) wird zur Entschlüsselung eine Java-Installation benötigt, damit der genutzte Quelltext kompiliert und ausgeführt werden kann.

Vorgehen Alle hier folgenden Informationen über den Aufbau der Dateien und den Entschlüsselungsprozess wurden 2017 auf einer Sicherheitskonferenz in Singapur vorgestellt und in [Dai+17] veröffentlicht.

Zum Entschlüsseln der AES-GCM-verschlüsselten Datenbank wird einerseits der 256 Bit große Schlüssel und andererseits der 128 Bit große Initialisierungsvektor (IV) benötigt. Wie den Tabellen B.1 und B.2 im Anhang B zu entnehmen ist, kann der Schlüssel in der `key`-Datei ab Offset 0x7E gefunden werden. Der IV ist hier allerdings ausgenullt, muss also der `*.crypt12`-Datei (Offset 0x33) entnommen werden.

Alle notwendigen Schritte zur Entschlüsselung werden in der Java-Klasse aus Listing C.1 im Anhang C (modifiziert nach [Dai+17]) durchgeführt. Das heißt, nachdem der Quelltext mit den korrekten Dateipfaden versehen wurde, kann die Datei kompiliert und ausgeführt werden.

Ergebnis Die SQLite-Datenbank `msgstore.db` liegt in unverschlüsselter Form vor.

3.2 Analyse von Logdateien in Bezug auf WhatsApp

Oft geht es in der Forensik darum, die Aktivitäten einer Person in einem bestimmten Zeitraum zu rekonstruieren bzw. eine Behauptung über das Verhalten dieser Person zu bestätigen oder zu widerlegen.
In Bezug auf WhatsApp könnten solche Aktivitäten z.B. das Versenden einer Nachricht an einen bestimmten Empfänger oder das Löschen von relevanten Mitteilungen sein. Auch der Nachweis, ob eine Person die WhatsApp-Anwendung, und damit das Telefon, genutzt hat, obwohl dies rechtlich nicht gestattet ist (z.B. während einer Autofahrt) kann forensisch von Bedeutung sein.

Eine Möglichkeit, Beweise zu realen Ereignissen bzw. Aktivitäten zu finden, ist die Analyse von Logdateien. Dabei handelt es sich ganz allgemein um „automatisch generierte[...] elektronische[...] Protokolle“ [Pri04] von Softwarekomponenten.
Wenn ein solches Protokoll einer Anwendung vorliegt und bekannt ist, wie zugehörige Einträge generiert werden, kann anhand derer darauf geschlossen werden, welche Aktivitäten der Nutzer wirklich ausgeführt hat.

Um Einträge, die Informationen über die Nutzung von WhatsApp enthalten, zu finden, wurden zwei verschiedene Logdateien betrachtet: das Android-Systemprotokoll „Logcat“ sowie die WhatsApp-interne Logdatei `whatsapp.log` (siehe auch Abschnitt 2.2.4.8).

Außer der Herkunft unterscheiden sich beide Logs auch in ihrem Aufbau und ihrer Speicherkapazität. Das Logcat ist als System aus mehreren Ringspeichern ausgelegt. Dabei existieren in diesem System fünf verschiedene Puffer: *main*, *system*, *radio*, *events* und *crash*, die jeweils eine begrenzte Größe haben.
Damit diese Größe nicht überschritten wird, werden beim Hinzukommen neuer Einträge die ältesten wieder entfernt. Die maximale Größe aller Ringpuffer ist fest in den Systemkernel integriert [The18] und kann mit dem folgenden ADB-Kommando ausgelesen werden [Gar11; Goo18c].

```
adb logcat -b all -g
```

Im Gegenteil dazu werden Zeilen in der WhatsApp-Logdatei nicht wieder gelöscht, sondern bei Bedarf die aktuelle Datei archiviert und eine neue angelegt. Allerdings werden auch diese Archive, und damit die enthaltenen Logeinträge, nach einer definierten Zeit wieder entfernt, um Speicherplatz zu schaffen.

3.2.1 Analyse der Logdateien

Um die Korrelation von Logeinträgen mit realen Ereignissen zu analysieren, wurden die auf einem Testgerät ausgeführten Aktionen nach der Extraktion der Logdateien auf einem Auswerterechner mit den generierten Einträgen verglichen.
Dabei kam ebenfalls die in Abschnitt 3.1 aufgeführte Testumgebung zum Einsatz.

Extraktion der Logdateien Zum Zugriff auf die WhatsApp-Logdatei wurde die in Abschnitt 3.1.3 erläuterte Methode zur Erstellung eines kompletten Backups aller WhatsApp-Daten über ADB angewandt und die Datei `whatsapp.log` aus dem tar-Archiv entpackt.
Die Inhalte des Android-Systemlogs können mit dem ADB-Kommando

```
adb logcat -b all -v time -d > logcat.txt
```

in die Textdatei `logcat.txt` ausgeleitet werden. Dabei bewirkt die Option `-b all`, dass alle Ringpuffer des Logcats ausgelesen werden, `-v time` stellt sicher, dass jede ausgegebene Zeile mit dem entsprechenden Zeitstempel beginnt und `-d` bedeutet, dass nur der aktuell vorhandene Inhalt gespeichert wird und keine permanente Aktualisierung gewünscht ist [Goo18c].

Analyse der Einträge Zur Generierung von Logeinträgen wurden auf dem Testgerät verschiedene definierte Aktionen ausgeführt, entweder direkt in der WhatsApp-Anwendung oder auf dem Smartphone allgemein. Alle einzelnen Aktionen sind in den Tabellen B.4 und B.5 im Anhang B aufgelistet. Dabei wurde ein weiteres Telefon mit installiertem WhatsApp als Kommunikationspartner genutzt, um Logeinträge zu den Aktionen zu generieren, bei denen das Testgerät der passive Part ist, z.B. bei eingehenden Nachrichten oder Anrufen.

Da bei beiden Protokolldateien innerhalb kürzester Zeit sehr viele Einträge erstellt werden, ist eine Separierung relevanter von irrelevanten Informationen notwendig.
Dazu wurde bei allen Ereignissen die Zeit der Ausführung auf Sekunden genau notiert.
Außerdem bietet die Android Debug Bridge die Möglichkeit, das Logcat komplett zu leeren, was zum Entfernen unwichtiger Loginhalte vor Ausführen der Aktion genutzt wurde. Der Befehl zum Aufräumen aller Puffer lautet:

```
adb logcat -b all -c
```

Qualitätssicherung Zur Eingrenzung der zu analysierenden Einträge wurden alle Aktionen mehrfach ausgeführt und die entstandenen Logdateien verglichen, um Zei-

len zu finden, die wahrscheinlich mit dem Ereignis in Verbindung stehen.
Dabei wurde besonders auf den Inhalt der Einträge geachtet, um falsch positive Ergebnisse so weit wie möglich auszuschließen (die Zeichenkette `app/startOutgoing Call` weist wahrscheinlicher auf einen ausgehenden Anruf hin als `device/memory/system/available`).
Des Weiteren wurden vermutliche Treffer stets gegen neutrale Logdateien geprüft, denen das betreffende Ereignis definitiv nicht zugrunde liegt. Ist der betreffende Eintrag in beiden Protokollen enthalten, wird er verworfen bzw. nach weiteren Zusammenhängen gesucht, die das Ereignis belegen.

3.2.2 Automatisierte Extraktion relevanter Logeinträge

Um die Zuordnung von Logeinträgen aus den beiden Dateien zu realen Ereignissen zu automatisieren, wurde im Rahmen dieser Arbeit ein Programm namens „WhatsApp Timeline Extractor“ (kurz „WATE“) entwickelt, welches mit Hilfe von regulären Ausdrücken (kurz Regex für englisch: **reg**ular **ex**pression) relevante Zeilen herausfiltert und die enthaltenen Informationen zur Auswertung aufbereitet.
Dieses Programm wurde in der Interpretersprache „Python 3.6“ implementiert und verwendet für die graphische Oberfläche (GUI) das Toolkit „wxPython“, welches auf der plattformübergreifenden C++-Bibliothek „wxWidgets“ aufbaut [wxP18]. Dadurch wird eine hohe Kompatibilität zu verschiedenen Betriebssystemen gewährleistet.

In diesem Abschnitt wird die Funktionsweise des Programms anhand einzelner Quellcode-Ausschnitte erläutert.

3.2.2.1 Funktionalität

Der WhatsApp Timeline Extractor extrahiert und präsentiert dem Nutzer reale Ereignisse aus Logdateien von Logcat oder WhatsApp.

Voraussetzungen Zum Ausführen des Programms werden einige Anforderungen an das ausführende System gestellt:

Python-Interpreter: mind. Version 3.6, über Systempfad erreichbar

zusätzliche Module: wxPython 3,
XlsxWriter 1 (zum Erstellen von Excel-Exports)

Betriebssystem: Windows, Linux oder Mac OS X (getestet auf Windows 8.1, Kernel 4.16.15-rt7-MANJARO und OS X 10.13.5)

Gestartet wird WATE durch das im Hauptordner des Programms ausgeführte Kommando:

```
python start.py
```

Input Mindestens eins der beiden Logs muss als Textdatei bereitgestellt werden. Im Fall von WhatsApp ist es möglich, statt der Datei `whatsapp.log` auch eine archivierte Version `whatsapp-yyyy-mm-dd.1.log.gz` zur Extraktion auszuwählen, welche vor der Verarbeitung automatisch entpackt wird.
Wird zusätzlich die SQLite-Datenbank `wa.db` bereitgestellt, werden die Ereignisse außerdem mit Kontaktinformationen angereichert.

Zeitleisten-Extraktion Die eingelesenen Dateien werden mit Hilfe von regulären Ausdrücken durchsucht und relevante Logeinträge herausgefiltert.
Die so generierte Zeitleiste wird im WATE-Fenster in übersichtlicher tabellarischer Form mit zusätzlichen Informationen, wie der Telefonnummer des betreffenden Kontaktes oder dem Namen der gestarteten Anwendung, dargestellt und kann bei Selektion durch den Nutzer mit den zugehörigen Zeilen in der entsprechenden Logdatei verglichen werden. Auf diese Weise wird eine größtmögliche Nachvollziehbarkeit gewährleistet.
Bei Bereitstellung der Kontaktdatenbank `wa.db` werden zu allen extrahierten Telefonnummern zusätzlich die Kontaktnamen eingeblendet.
Weiterhin ist es möglich, die extrahierte Zeitleiste nach dem Zeitstempel zu filtern und so auf relevante Ereignisse einzugrenzen.

Export Zur weiteren Verarbeitung oder permanenten Speicherung ist es möglich, eine Exportdatei mit allen aktuell angezeigten oder nur ausgewählten Ereignissen erstellen zu lassen. Dabei kann zwischen den Formaten *CSV* (Comma Separated Values) und *XLSX* (Excel 2007+ Arbeitsmappe) gewählt werden.
Auf Wunsch werden die zur Erzeugung der Zeitleiste genutzten Einstellungen, Hashwerte der eingelesenen Dateien, eine einfache textuelle Beschreibung jedes Ereignisses oder der komplette Logeintrag in die Exportdatei eingeschlossen.

Sprachwahl Die Nutzeroberfläche des Programms ist in den Sprachen Deutsch und Englisch verfügbar, wobei letztere der Standardeinstellung entspricht.
Über einen Menüpunkt kann die Sprache der GUI jederzeit gewechselt werden. Die

bereits extrahierte Zeitleiste wird dabei allerdings nicht übersetzt, sondern muss zu diesem Zweck neu erstellt werden.

Die Funktionalität des Programms mit Ein- und Ausgabedateien wird in Abbildung 3.2 schematisch dargestellt.
Je ein Bildschirmfoto des WATE-Hauptfensters nach erfolgter Extraktion für alle unterstützten Betriebssysteme ist den Abbildungen A.3 bis A.5 im Anhang A zu entnehmen.

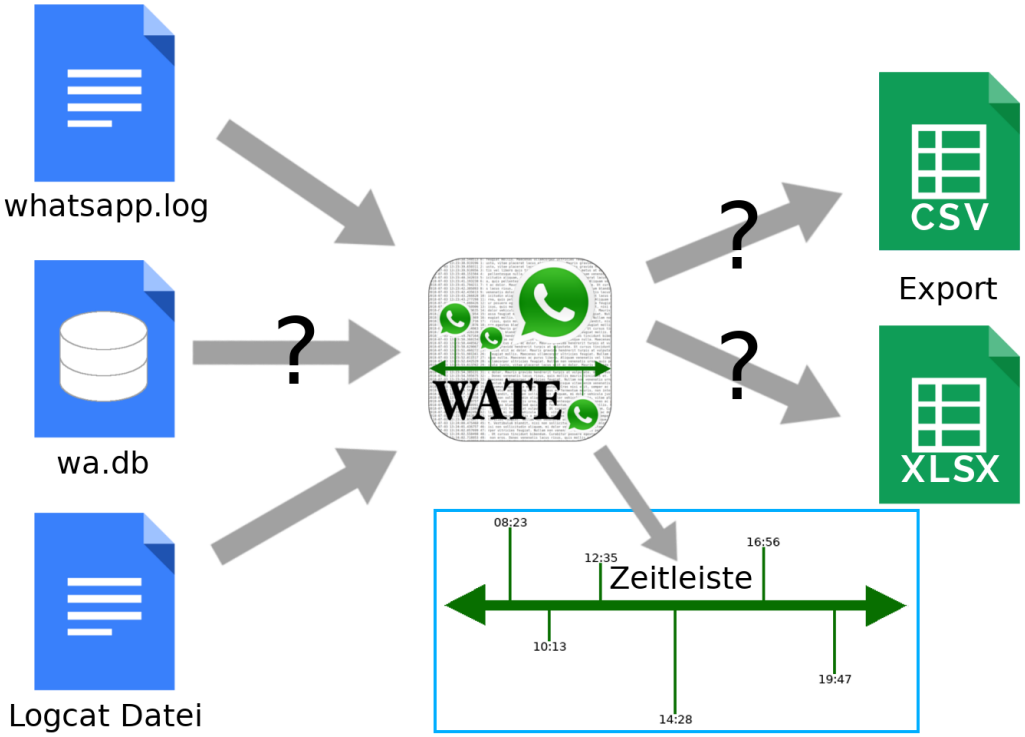


Abbildung 3.2: Schematische Darstellung der Ein- und Ausgaben bei WATE (? – In-/Output optional)
Quelle: eigene Arbeit, Icons von `pngimg.com` und `Leoncastro auf commons.wikimedia.org`

3.2.2.2 Struktureller Aufbau des Programms

Die Verzeichnisstruktur der zum WhatsApp Timeline Extractor gehörenden Dateien ist in der folgenden Übersicht dargestellt.

- `WhatsAppTimelineExtractor/`
  - `config/`
    - `logs.conf` (Konfigurationsdatei mit regulären Ausdrücken und Einstellungen)
  - `gui/`
    - `img` (Ordner für Bilder, die in der GUI dargestellt werden)
    - `locale` (Ordner für sprachenabhängige Zeichenketten)
      - `...`
    - `tmp` (Ordner für temporäre Dateien)
    - `__init__.py` (Python-Quelltext, enthält globale Programm-Konstanten)
    - `main.py` (Python-Quelltext für das Hauptfenster der GUI)
    - `customwidgets.py` (Python-Quelltext für alle weiteren GUI-Elemente)
  - `logparser/`
    - `__init__.py` (Python-Quelltext mit globalen Konstanten)
    - `log.py` (Python-Quelltext der Basisklasse für Logparser)
    - `logcat.py` (Python-Quelltext der Parser-Klasse für Logcat-Dateien)
    - `walogs.py` (Python-Quelltext der Parser-Klasse für WhatsApp-Logdateien)

Die Konfigurationsdatei `config/logs.conf` wird zur Zentralisierung aller regulären Ausdrücke für das Auffinden relevanter Logeinträge sowie zum persistenten Speichern gewählter Einstellungen verwendet. Dadurch können einzelne Ausdrücke sehr einfach verändert werden, falls Abweichungen oder Aktualisierungen der entsprechenden Logeinträge festgestellt werden (z.B. bei verschiedenen Gerätetypen oder nach einem Systemupdate).

Im `gui`-Ordner befinden sich alle Dateien, die für die Darstellung der Benutzeroberfläche notwendig sind.
Dabei enthält `main.py` die Python-Klasse `MainWindow(wx.Frame)`, die die gesamte Oberfläche des Hauptfensters sowie zu allen ausführbaren Aktionen Eventhandler-Methoden bereitstellt. Weitere notwendige GUI-Klassen (z.B. alle Dialoge, die geöffnet werden können) werden in der Datei `customwidgets.py` bereitgestellt.
Alle in der GUI dargestellten Zeichenketten sind in maschinenlesbaren Lokalisationsdateien `main.mo` im jeweiligen Unterordner von `gui/locale` für beide verfügbaren Sprachen abgelegt. Sie werden zur Laufzeit in der gewünschten Sprache geladen und

durch die Methode `_(key)` anhand eines für jede Zeichenkette definierten Schlüssels durch das Modul `gettext` in die GUI eingebaut.

3.2.2.3 Erstellen der Zeitleiste

Für die Extraktion von Ereignissen aus den gegebenen Logdateien sind die Python-Klassen im Ordner `logparser` zuständig, ebenso wie für die Erstellung einer geordneten Zeitleiste (englisch: Timeline) mit zu den Ereignissen gehörenden Metadaten.
Da Android-Logcat und WhatsApp Log unterschiedlich aufgebaut sind und unterschiedliche Einträge enthalten, sind verschiedene Methoden notwendig, um Ereignisse zu extrahieren, wobei es auch Schritte gibt, die bei beiden Logdateien ausgeführt werden müssen.
Um dem gerecht zu werden, wurden zwei separate Python-Klassen mit unterschiedlichen Methoden erstellt, welche beide von der Superklasse `LogFile` erben, die gemeinsame Funktionen enthält.
In diesem Abschnitt soll die Funktionsweise der drei Klassen kurz erläutert werden, die Klassendiagramme sind Abbildung A.2 (Anhang A) zu entnehmen.

Die Klasse LogFile Diese Klasse wird in der Datei `log.py` definiert und stellt die Basis für alle Ereignisse aus Logdateien extrahierenden Klassen dar, weshalb die enthaltenen Methoden besonders wichtig sind.

Definition von Variablen In der `__init__(self, filename, events)`-Methode, die beim Erzeugen eines `LogFile`-Objektes aufgerufen wird, erfolgt nach der Überprüfung, ob die angegebene Logdatei existiert, die Initialisierung einiger Klassenvariablen.
Je nach Logtyp (Logcat oder WhatsApp Log) enthalten diese jedoch später andere Werte.

```
16  def __init__(self, filename, events):
17      self.check_logfile_existance(filename)
18
19      self.filename = filename  # str
20      self.timeline = list()
21      self.EVENTS = events  # list
22      self.REGEX = dict()
23      self.META_REGEX = dict()
```

Listing 3.2: Definition von Klassenvariablen in `LogFile`

Wie in Listing 3.2 ersichtlich ist, werden fünf Variablen angelegt, die zu folgenden Zwecken verwendet werden:

`self.filename` speichert den Dateinamen der Eingabe-Logdatei.

`self.timeline` ist eine Liste, die später die extrahierten Ereignisse enthält. Diese werden als einzelne Wörterbücher (englisch: dictionaries) repräsentiert.

`self.EVENTS` wird hier mit dem entsprechenden Inhalt versehen, der durch die erbenden Klassen bereitgestellt wird. Die Liste enthält festgelegte Zeichenketten, die als Ereignis-IDs ($\mathcal{ID}_{Ereignis}$) zur Identifizierung der unterschiedlichen Ereignistypen dienen.

`self.REGEX` ist ein Dictionary, das alle in der Konfigurationsdatei festgelegten regulären Ausdrücke zum Auffinden relevanter Logzeilen enthält.

`self.META_REGEX` ist ebenfalls ein Wörterbuch mit zusätzlichen regulären Ausdrücken aus der `logs.conf`-Datei zum Extrahieren von Metadaten wie der Nachrichten-ID aus gefundenen Logeinträgen.

Vorbereiten der regulären Ausdrücke Das Einlesen der regulären Ausdrücke aus der Konfigurationsdatei `config_file` erfolgt in der Klassenmethode `read_config(self, config_file, section_prefix)` mit Hilfe der Klasse `ConfigParser` aus dem Modul `configparser`. Dabei werden alle Ausdrücke durch das Modul `re` kompiliert, damit die mehrfache Nutzung weniger Zeit beansprucht. Anschließend werden sie in den bereits angelegten Dictionaries `self.REGEX` und `self.META_REGEX` gespeichert. Da je nach Logtyp jedoch verschiedene Regex benötigt werden, müssen zuvor durch den Parameter `section_prefix` die korrekten Inhalte der Konfigurationsdatei gewählt werden (`"WA"` oder `"LOGCAT"`).

Extrahieren relevanter Logeinträge Die nächste wichtige Methode der Klasse `LogParser` heißt `find_events(self, log)` und durchsucht mit Hilfe der vorher geladenen regulären Ausdrücke die Logdatei nach relevanten Einträgen. Im nachfolgenden Listing ist der Quellcode der Methode dargestellt.

```
40  def find_events(self, log):
41    events = list()
42    # structure of events:
43    # [{'pos':   (<1st char position>, <line nr>),
44    #   'event': one out of self.EVENTS,
45    #   'entry': raw log entry (line)}, {...}, ...]
46
47    pos = 0
48    for nr, line in enumerate(log):  # for every log line
49      for event_name in self.EVENTS:  # check every event ↩
    ↪ regex
50        if self.REGEX[event_name].search(line):  # regex hit
51          events.append({'pos': (pos, nr),
52                         'event': event_name,
53                         'entry': line})
54          break
55      pos += len(line) + 1  # because \n is added later
56
57    return events
```

Listing 3.3: Die Methode `LogParser.find_events()`

Jede Zeile des Logs wird auf jeden Regex geprüft. Bei einem Treffer (in Zeile 50) wird der Eintrag in dem Dictionary `events` zusammen mit seiner Position sowie der korrekten $\mathcal{ID}_{Ereignis}$ aus `self.EVENTS` abgelegt.
Die Position setzt sich zusammen aus der Lage dieser Zeile im gesamten Log (vermerkt als die Stelle des ersten Zeichens der Zeile) und der Zeilennummer. Letztere dient der Nachvollziehbarkeit für den Nutzer, die Zeichenposition ist für das Hervorheben des Eintrags in der GUI notwendig.

weitere Methoden Weiterhin enthält die Klasse `LogParser` die gemeinsam genutzten Methoden `substring_log_entry(self, log_entry, key)` zum Extrahieren von zusätzlichen Informationen zu einem gegebenen Logeintrag, `get_sorted_timeline(self, timeline=None)` zum Sortieren einer fertigen Zeitleiste (notwendig nach Zusammenführen mehrerer Zeitleisten) sowie Methoden zur Überprüfung der Existenz bereitgestellter Dateien.

Die Klasse WALog Als Erbe von `LogFile` ist diese Klasse für das Verarbeiten von WhatsApp-Logdateien zuständig. Ihr Quelltext befindet sich in der Python-Datei `walogs.py`.

überschriebene Variablen Die Klassenvariable `self.EVENTS` ist wie folgt definiert:

```
6   EVENTS = ["OpenChat", "ChatSeen", "SendMessage", "↩
     ↪ ReceiveMessage", "DeleteMessage", "Unlock", "OutCall", "↩
     ↪ InCall", "AcceptOutCall", "AcceptInCall", "SendStatus", ↩
     ↪ "DeleteMessageAll", "DeleteStatus"]
7
8   EVENT_RETURN = "ReturnChat"  # the first "OpenChat" after ↩
     ↪ an "Unlock" Event is declared as "ReturnChat"
```

Listing 3.4: verfügbare Ereignisse in `WALog`

Wie in Listing 3.4 deutlich wird, gibt es zu den „normalen“ Ereignissen, die in `self.EVENTS` festgehalten werden (mehr Informationen dazu im Abschnitt 4.2), noch ein Weiteres namens `"ReturnChat"`. Dieses wird nicht direkt in der Logdatei repräsentiert, sondern im späteren Programmverlauf für das erste Öffnen eines Chats nach dem Entsperren des Telefons eingesetzt, was für den Nutzer besser nachvollziehbar ist.

Die Liste `self.timeline` enthält für jedes Ereignis ein Dictionary mit den Schlüsseln `time`, `pos`, `event`, `jid`, `msgid`, `type`, `raw` und `source`, wobei abhängig vom Ereignistyp einige Werte leer gelassen werden können (z.B. die `msgid` bei `"Unlock"`-Ereignissen).
Neben den offensichtlichen Bedeutungen steht `raw` für den kompletten Logeintrag und `type` für den Typ bei Ereignissen im Zusammenhang mit WhatsApp-Anrufen (Sprach- oder Videoanruf). Der Schlüssel `source` beschreibt, ob das Ereignis aus einer Logcat- oder WhatsApp-Logdatei stammt und ist demzufolge innerhalb dieser Klasse immer gleich.

Ablauf der Extraktion Da eine Instanz der `WALog`-Klasse immer nur eine Logdatei verarbeitet, wird die Extraktion schon während der Erzeugung (in der `__init__(self, filename, conf_file)`-Methode) durchgeführt.
Nachdem die Vorbereitungen durch die Methoden aus `LogParser` abgeschlossen sind, wird die angegebene Logdatei zeilenweise eingelesen und in die Klassenliste `self.log` eingetragen.
Die eigentlichen Ereignisse werden ebenfalls durch eine Elternmethode extrahiert (siehe Abschnitt 3.2.2.3), zur Erhöhung des Informationsgehaltes werden die Ergebnisse jedoch noch mit zusätzlichen, in den Logeinträgen enthaltenen Informationen (wie z.B. die angerufene Telefonnummer) angereichert. Dies geschieht in der durch `__init__()` aufgerufenen Methode `refine_event_lines(self, events)`.

Anreicherung der Ereignisse Die Extraktion von Metadaten zu den Ereignissen erfolgt in der `refine_event_lines()`-Methode. Dabei wird über jeden gefundenen Eintrag iteriert und mit Hilfe einer `if...elif...`-Anweisung je nach $\mathcal{ID}_{Ereignis}$ eine Extraktion der passenden Teile aus der Logzeile vorgenommen. Zu diesem Zweck wird die Methode `LogParser.substring_log_entry()` und die regulären Ausdrücke aus `self.META_REGEX` genutzt.
Für das Verfassen einer neuen Statusmeldung soll dieses Verfahren in Listing 3.5 beispielhaft verdeutlicht werden. Dabei wird zuerst der Zeitstempel und anschließend die JID des Empfängers[9], die Nachrichten-ID sowie der Typ des Status extrahiert.
Die benötigten regulären Ausdrücke werden durch ihren Schlüssel in der Konfigurationsdatei (als zweiter Parameter der `substring_log_entry()`-Methode) ausgewählt, wobei in diesem Fall teilweise die gleichen wie bei `"SendMessage"`-Ereignissen (z.B. `jid_send`) genutzt werden können.

```
152         elif line['event'] == self.EVENTS[10]:  # 'SendStatus'
153             self.insert_timeline_event(
154                 self.REGEX['time'].search(line['entry']).group(1)↩
    ↪ ,  # timestamp
155                 line['pos'],  # (pos, line nr)
156                 line['event'],  # Event ID
157                 self.substring_log_entry(line['entry'], 'jid_send↩
    ↪ '),  # same regex as for SendMessage
158                 self.substring_log_entry(line['entry'], '↩
    ↪ msgid_send'),
159                 self.substring_log_entry(line['entry'], '↩
    ↪ mediatype_status'),
160                 line['entry'])  # raw log content
```

Listing 3.5: Anreicherung von Statusereignissen in `WALog`

Zur Speicherung wird jedes Ereignis mit den gefundenen Metadaten der Klassenliste `self.timeline` hinzugefügt.
Ist zum Erschließen der Meta-Informationen mehr als nur der aktuelle Logeintrag notwendig (z.B. bei der JID von `"DeleteMessage"`-Ereignissen, siehe auch Abschnitt 4.2), werden die Informationen aus allen betroffenen Zeilen zusammengeführt. Im Beispiel wird die JID des zuletzt geöffneten Chats in der Variable `current_chat` zwischengespeichert und beim nächsten Auftreten von `"DeleteMessage"` genutzt, wobei für das resultierende Ereignis der Zeitstempel des letzten zugehörigen Logeintrags

[9]Im Normalfall werden Statusmeldungen immer an die JID `status@broadcast` verschickt. Sie soll dennoch extrahiert werden, um mit anderen Ereignissen konform zu sein.

genutzt wird.

Umgang mit Statusmeldungen Eine weitere Besonderheit gibt es bei eingehenden Nachrichten, da Statusmeldungen, die von jedem eingespeichertem Kontakt empfangen werden können, in der Logdatei mit dem gleichen Eintrag wie `"ReceiveMessage"`-Ereignisse repräsentiert werden. Da eingehende Statusmeldungen jedoch weder auf einer aktiven Handlung des Nutzers beruhen noch länger als 24 Stunden gespeichert werden, werden in dieser Methode durch eine `if`-Verzweigung diejenigen Ereignisse verworfen, deren extrahierte JID `status@broadcast` entspricht.
Die ID der Statusmeldung wird allerdings in der Klassenliste `self.status_ids` gespeichert, da beim Löschen von Status auch ein `"DeleteMessage"`-Eintrag generiert wird, dieses Ereignis allerdings nicht in die Zeitleiste übernommen werden soll. Die Methode sortiert also alle gelöschten Nachrichten aus, bei denen es sich eigentlich um Statusmeldungen handelt, indem solche, deren Nachrichten-IDs auch in `self.status_ids` enthalten sind, verworfen werden.
Einen weiteren Hinweis gibt das `"DeleteStatus"`-Ereignis (siehe Abschnitt 4.2), welches die Anzahl der zu löschenden Statusnachrichten enthält. Diese Anzahl an Löscheinträgen kann also ignoriert werden, bevor wieder relevante Einträge folgen.
Die beschriebene Methode stellt sicher, dass das Löschen von Statusmeldungen nur auftritt, wenn das Empfangen dieser nicht in der Logdatei enthalten ist und sie zusätzlich nicht von einem `"DeleteStatus"`-Eintrag abgedeckt werden. Es ist also möglich (wenn auch unwahrscheinlich bedenkend der Tatsache, dass die Zeit zwischen Empfangen und Löschen eines Status maximal 24 Stunden betragen kann und die Logdatei oft einen längeren Zeitraum enthält), dass auf diese Weise `"DeleteMessage"`-Ereignisse falsch zugeordnet werden.

Die Klasse Logcat Zur Verarbeitung von Logcat-Inhalten wird diese Klasse aus der Datei `logcat.py` genutzt. Sie ist ähnlich aufgebaut wie `WALog` und erbt ebenfalls von `LogFile`.

überschriebene Variablen Für Logcat-Analysen stehen folgende $\mathcal{ID}_{Ereignis}$ aus der Klassenvariable `self.EVENTS` sowie ein weiteres Ereignis zur Verfügung:

```
11  EVENTS = ["ScreenOn", "ScreenOff", "WakeLock", "↩
     ↪ LockscreenAction", "Unlocked", "StartApp", "CloseApp", "↩
     ↪ PauseApp", "ResumeApp"]
12
```

```
13  EVENT_SWITCH = "SwitchApp"  # Start / ResumeApp directly ↩
     ↪ after PauseApp
```

Listing 3.6: verfügbare Ereignisse in `LogCat`

Das `"SwitchApp"`-Ereignis stammt nicht direkt aus der Logdatei, sondern wird erst im Laufe des Programms generiert. Es bedeutet, dass der Nutzer zwischen zwei Anwendungen gewechselt hat, was durch ein Minimieren (`"PauseApp"`) der alten Applikation und das anschließende Wiederherstellen (`"ResumeApp"`) der neuen Anwendung beschrieben wird. Damit dieses Ereignis erzeugt wird, müssen beide Aktionen direkt nacheinander auftreten und zwei verschiedene Apps betroffen sein.
Für diesen zusätzlichen Aufbereitungsschritt kommt in der sonst analog zur der in `WALog` aufgebauten Methode `refine_event_lines(self, events)` zuerst die Liste `self.tmp_timeline` zur temporären Speicherung der Zeitleiste zum Einsatz. Danach werden in einem zusätzlichen Schritt passende Einträge durch `"SwitchApp"`-Ereignisse ersetzt und die finale Zeitleiste in `self.timeline` erzeugt.

Diese enthält wieder für jedes Ereignis ein Dictionary. Dabei kommen die Schlüssel `time`, `pos`, `event`, `meta1`, `meta2`, `type`, `raw` und `source` zum Einsatz, wobei `type` immer leer und nur zur Kompatibilität mit der Zeitleiste von WhatsApp-Logdateien vorhanden ist.
Mögliche Metadaten werden zu den Schlüsseln `meta1` bzw. `meta2` gespeichert. Je nach Ereignis handelt es sich dabei aber um verschiedene Informationen, wie z.B. den Namen der App und der zugehörigen Activity[10] oder die zwei Anwendungen, zwischen denen gewechselt wurde.

Ablauf der Extraktion Die Extraktion und Anreicherung der Ereignisse erfolgt ähnlich wie in der Klasse `WALog`, allerdings gibt es auch einige wichtige Unterschiede. So werden bei Logcat-Extraktionen eine Whitelist für sinnvolle `"WakeLock"`-Tags (mehr dazu im Abschnitt 4.2) sowie eine Blacklist für Anwendungen, die in den `"*App"`-Ereignissen nicht vorkommen sollen[11], berücksichtigt. Beide Listen werden wie die regulären Ausdrücke in der Konfigurationsdatei gespeichert.

[10] Bei den analysierten Ereignissen unterscheidet Android nicht zwischen einzelnen Apps, sondern zwischen Aktivitäten (englisch: Activities).
Eine App besteht aus mindestens einer Activity, oft sind aber mehrere vorhanden. Jede Activity stellt eine „Seite“ oder „Ansicht“ der Anwendung dar, zwischen denen der Nutzer wechseln kann [Gar11].
WhatsApp hat z.B. eine `HomeActivity` für die Übersicht der aktuellen Chats und eine `Conversation`-Activity zur Darstellung eines einzelnen Nachrichtenverlaufs.

[11] Eine irrelevante App, die möglicherweise ignoriert werden soll, ist z.B. die Anwendungsübersicht („Launcher“).

Apps und Activities Die in Fußnote 10 erläuterte Tatsache, dass Android einzelne Activities, und nicht ganze Apps, als Einheit behandelt, steht im Gegensatz zur Einschätzung der meisten Menschen, die nur Anwendungen unterscheiden. Ein `"SwitchApp"`-Ereignis zwischen `WhatsApp/HomeActivity` und `WhatsApp/Conversation` ist für den Nutzer deshalb schlecht nachvollziehbar.
Um den Nutzen der extrahierten Informationen zu maximieren und diese Diskrepanz zu überwinden, enthält die Klasse `LogCat` noch fünf weitere Methoden und zwei, in Listing 3.7 dargestellte, Dictionaries als Klassenvariablen.

```
43      self.active_apps = dict()    # all currently active apps (↩
        ↪ from start / resume to pause)
44      self.running_apps = dict()  # all started apps (from ↩
        ↪ start to stop, active or inactive)
```

Listing 3.7: Klassenvariablen zur Nachverfolgung von Aktivitäten in `LogCat`

Mit deren Hilfe wird nachverfolgt, welche Anwendungen laut der Logdatei gerade geöffnet (also gestartet und noch nicht beendet) bzw. aktiv (also gestartet oder wiederhergestellt und noch nicht minimiert) sind. Dazu gibt es in den beiden Wörterbüchern für jede App einen Eintrag. Der Paketname (z.B. `com.whatsapp`) dient hierbei als Schlüssel und der zugehörige Wert ist eine Liste mit allen Aktivitäten der App, die sich in dem jeweiligen Zustand befinden.
Die Dictionaries werden verwaltet von den vier Methoden `start_act(self, app, activity)`, `resume_act(self, app, activity)`, `pause_act(self, app, activity)` und `stop_act(self, app, activity)`, die für die entsprechenden Ereignisse in `refine_event_lines()` aufgerufen werden.
Mit den Rückgabewerten signalisieren sie, ob tatsächlich die gesamte Anwendung gestartet (vorher noch kein Eintrag für diese App in `self.running_apps`), beendet usw. wurde oder ob es sich nur um eine Veränderung innerhalb der App handelt. Nur im ersten Fall wird das Ereignis als relevant erachtet und in die Zeitleiste eingetragen.
Eine Besonderheit stellt die Methode `start_act()` dar, denn wenn für die gegebene Anwendung mindestens eine Aktivität im Hintergrund, aber keine aktiv ist und ein `"StartApp"`-Ereignis für eine neue Activity dieser Anwendung auftaucht, handelt es sich nicht wirklich um den Start einer neuen App, sondern um das Wiederherstellen mit einer anderen Aktivität.
Dies wird über einen zweiten Rückgabewert verdeutlicht und gegebenenfalls in `refine_event_lines()` mit einer Änderung der $\mathcal{ID}_{Ereignis}$ berücksichtigt.

Des Weiteren enthält `LogCat` die Methode `app_from_activity(self, activity)`, um den Namen der Anwendung von dem der Activity zu separieren.

4 Ergebnisse

4.1 Extraktion von WhatsApp-Daten

Nach jeder im Abschnitt 3.1 erklärten Methode wurde eine Extraktion der betreffenden WhatsApp-Daten durchgeführt. In den nachfolgenden Ergebnissen wird deutlich, welche Daten dadurch gewonnen wurden.

4.1.1 Export von Chatverläufen als Textdatei

Als Grundlage wurde der Chat zwischen „Ich“ (eigener WhatsApp-Name) und „Kommunikationspartner“ (Name des Kontaktes) mit insgesamt 226 Nachrichten genutzt. Der Export wurde über Bluetooth, WhatsApp sowie, wie in Abschnitt 3.1.1 beschrieben, über E-Mail durchgeführt. Dabei wurde sowohl die Option „Einschließlich Medien“ als auch „Ohne Medien“ gewählt.

Von den 226 Nachrichten sind in dem Export mit Mediendateien nur fünf vorhanden (siehe Abbildung 4.1), was leicht nachzählbar ist.
Da die Exportdatei ohne Medien allerdings einen viel größeren Umfang besitzt (siehe Abbildung 4.2), kam zum Zählen ein regulärer Ausdruck für den einmal pro Nachricht enthaltenen Zeitstempel zum Einsatz. Wie in Abbildung 4.3 ersichtlich ist, wurde dieser 226 mal gefunden, es sind also alle Nachrichten enthalten.

Wird die Option „Ohne Medien“ gewählt, erhält man eine einzelne Textdatei namens `WhatsApp Chat mit Kommunikationspartner.txt`, wobei anstelle von nicht exportierten Dateien die Information `<Medien ausgeschlossen>` steht, ein Hinweis auf den Dateinamen fehlt. Beim Export von Mediendateien werden diese zusammen mit der Textdatei in einem *ZIP*-Archiv gespeichert und im Nachrichtenverlauf auf die korrekte Datei hingewiesen. Dieses hat ebenfalls den Namen `WhatsApp Chat mit Kommunikationspartner.zip`.

Die erhaltene Textdatei beim Export ohne Medien wurde mit Hilfe des in Abschnitt 3.1.1 erwähnten Python-Skriptes in eine HTML-Datei umgewandelt, von der ein Ausschnitt in Abbildung 4.4 zu sehen ist. Dazu wurde folgendes Kommando genutzt:

```
python wa_txt2html.py -i WhatsApp\ Chat\ mit\
Kommunikationspartner.txt -o Kommunikationspartner.html -m "Ich"
```

```
10.07.18, 09:38 - Kommunikationspartner: IMG-20180710-WA0005.jpg (Datei angehängt)
10.07.18, 09:38 - Kommunikationspartner: IMG-20180710-WA0004.jpg (Datei angehängt)
10.07.18, 09:38 - Kommunikationspartner: IMG-20180710-WA0003.jpg (Datei angehängt)
10.07.18, 09:38 - Kommunikationspartner: IMG-20180710-WA0002.jpg (Datei angehängt)
10.07.18, 14:18 - Ich: DOC-20180710-WA0007.zip (Datei angehängt)
WhatsApp Chat mit Änderkontakt
```

Abbildung 4.1: Chat-Exportdatei bei Export mit Mediendateien (komplett)
Quelle: eigene Arbeit

```
[...]
10.07.18, 07:26 - Kommunikationspartner: Hallo, Kommunikationspartner! 😀

Wie geht es dir denn heute so?
10.07.18, 07:29 - Ich: Hallo, danke der Nachfrage.
Mir geht es gut ☺.
10.07.18, 07:29 - Ich: Im Moment bin ich gerade dabei, Inhalt für einen Test zu generieren.
10.07.18, 07:30 - Kommunikationspartner: Oh, was denn für ein Test?🤔
10.07.18, 07:32 - Ich: Ich will einen WhatsApp-Chat über die eingebaute E-Mail-Funktion
exportieren, damit ich ihn am Computer besser lesen und weitergeben kann...
10.07.18, 07:32 - Ich: Bin mal gespannt, wie das funktionieren wird
10.07.18, 07:44 - Kommunikationspartner: Na dann viel Erfolg! 🤞🤞
10.07.18, 07:54 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 08:03 - Die Sicherheitsnummer von Kommunikationspartner hat sich geändert. Tippe für
mehr Infos.
10.07.18, 08:17 - Kommunikationspartner: Damit du weißt, wo ich bin:
10.07.18, 08:17 - Die Sicherheitsnummer von Kommunikationspartner hat sich geändert. Tippe für
mehr Infos.
10.07.18, 08:18 - Ich: Wenn du das lesen kannst, hast du die *.crypt12-Datei erfolgreich
entschlüsselt!!! 👍
10.07.18, 08:19 - Kommunikationspartner: 😀
10.07.18, 09:33 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:34 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:36 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:38 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:38 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:38 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:38 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 09:38 - Kommunikationspartner: <Medien ausgeschlossen>
10.07.18, 14:18 - Ich: <Medien ausgeschlossen>
```

Abbildung 4.2: Chat-Exportdatei bei Export ohne Mediendateien (Ausschnitt)
Quelle: eigene Arbeit

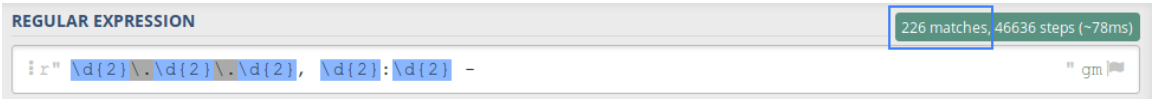


Abbildung 4.3: mit regulärem Ausdruck gezählte Nachrichteneinträge im Chat-Export ohne Medien
Quelle: eigene Arbeit unter Verwendung von [fir18]

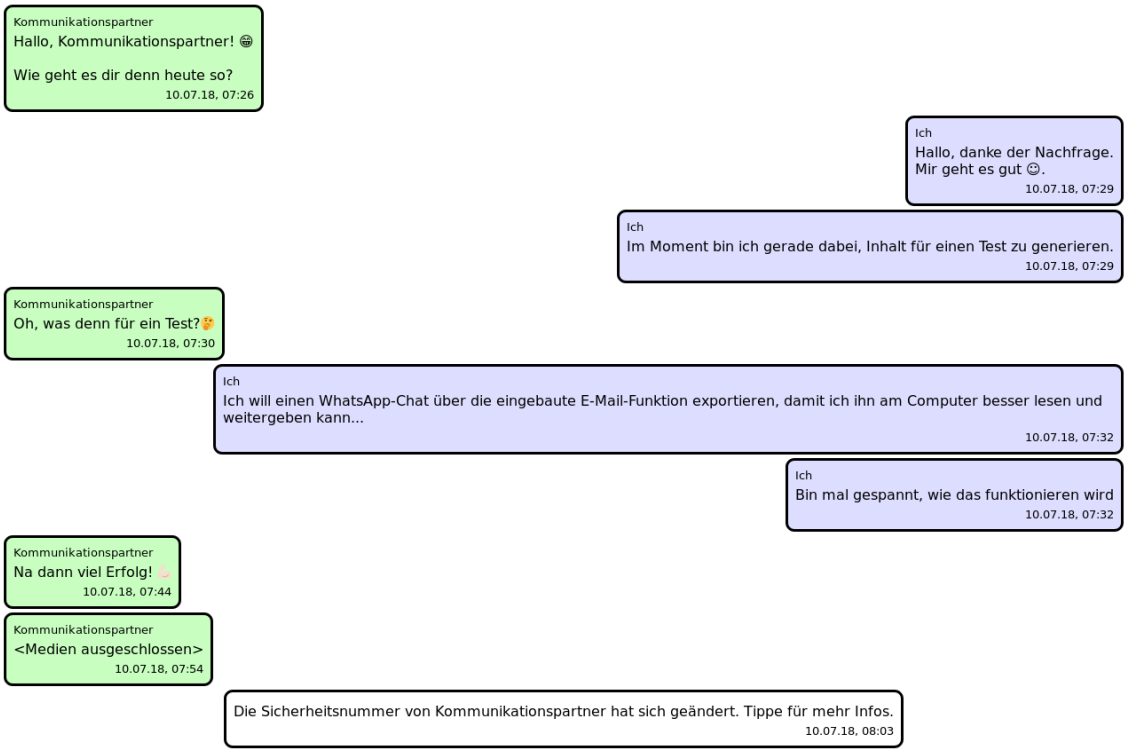


Abbildung 4.4: Chatexport als HTML-Datei (Ausschnitt)
Quelle: eigene Arbeit

4.1.2 Nachrichtenwiederherstellung durch Neuregistrierung

Nach der Installation von WhatsApp auf $\mathcal{T}_{Emu}$ und der erfolgreichen Registrierung mit der gleichen Telefonnummer wie vorher, wird auf dem englischsprachigen Emulator die Meldung aus Abbildung 3.1 auf Englisch angezeigt.
Nachdem die Wiederherstellung bestätigt wurde, führt WhatsApp diesen Vorgang aus und auf $\mathcal{T}_{Emu}$ sind alle Nachrichten verfügbar, allerdings ohne Kontaktnamen. Die so gewonnene Chat-Übersicht wird in Abbildung 4.5 dargestellt (Profilbilder und Telefonnummern geschwärzt).

Anschließend kann über ADB auf alle Daten zugegriffen werden. Die unverschlüsselte Nachrichtendatenbank auf dem aktuellsten Stand konnte so gewonnen werden, wie in Abbildung 4.6 ersichtlich ist.

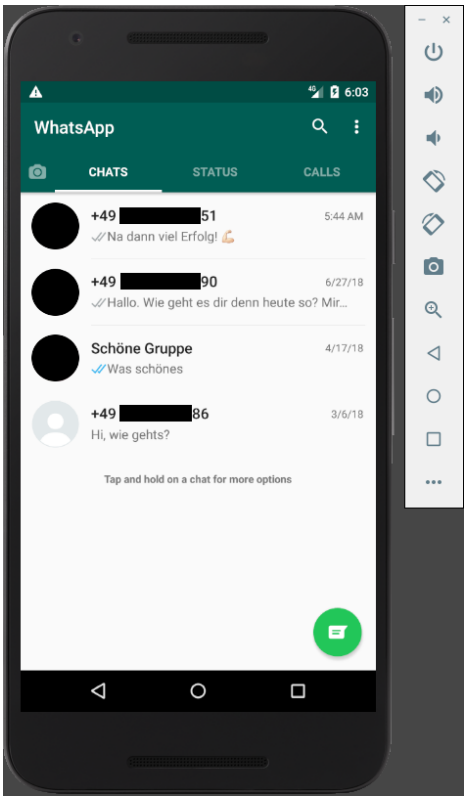


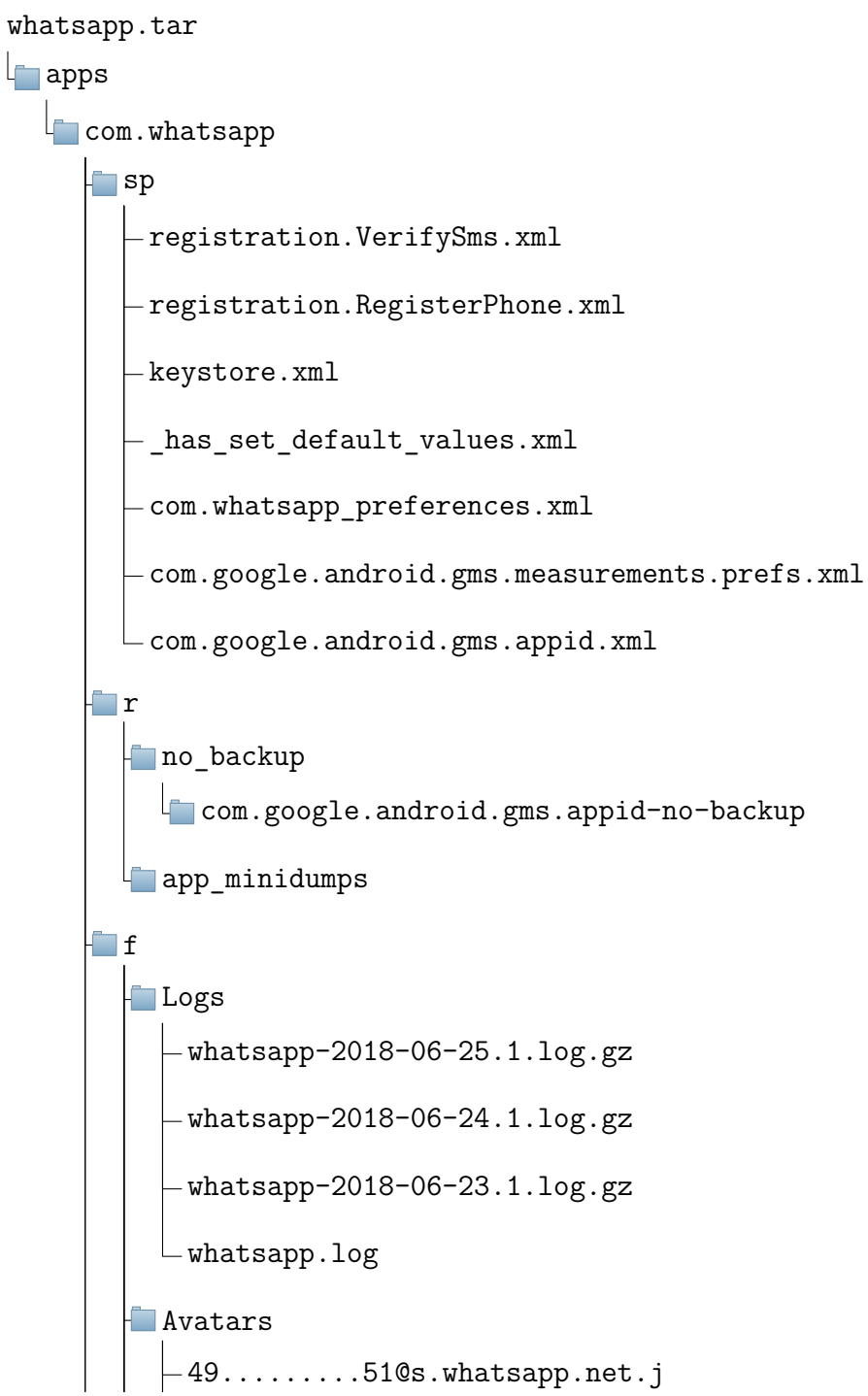
Abbildung 4.5: wiederhergestellte Nachrichtenverläufe auf $\mathcal{T}_{Emu}$
Quelle: eigene Arbeit

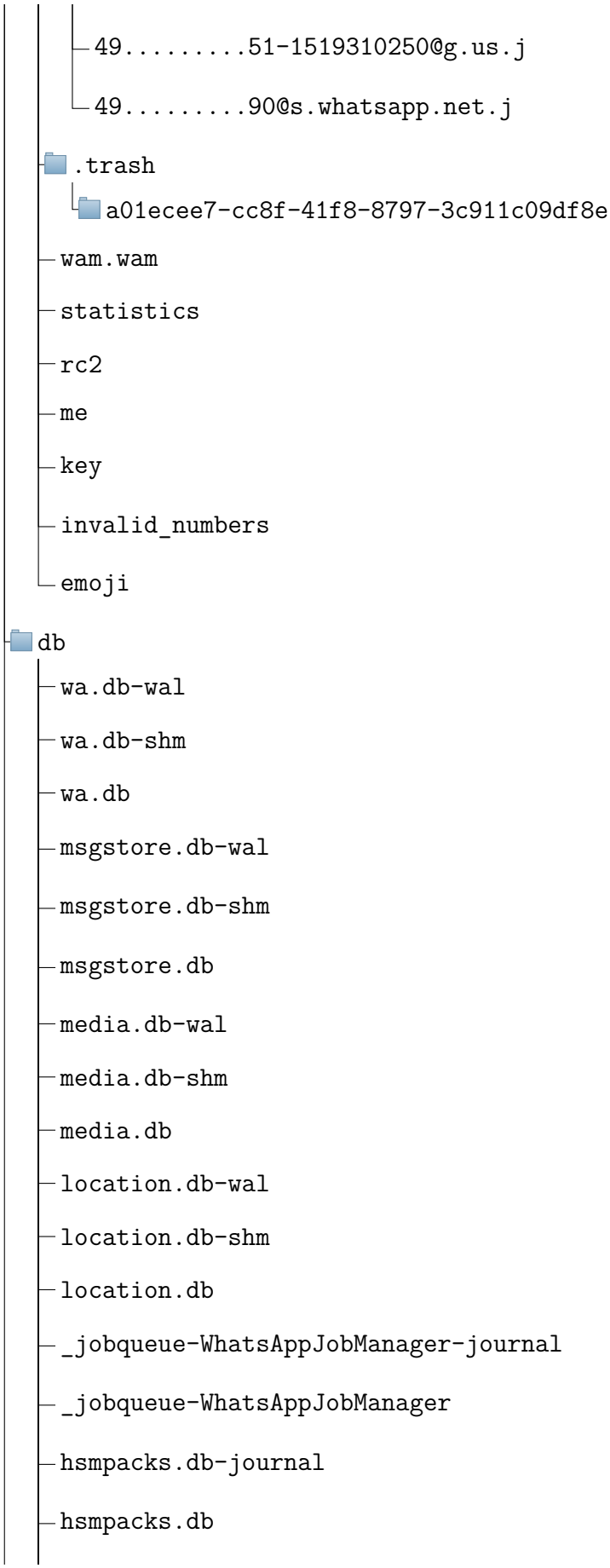
	_id	key_remote_jid	key_from_me	key_id	status	needs_push	data	timestamp
		Filtern	Filtern	Filtern	Filtern	Filtern	Filtern	Filtern
158	203	49[redacted]51@s.whatsapp.net	1	214FA9C75...	5	0	Hallo, Kommunikationspartn...	153120040...
159	204	49[redacted]51@s.whatsapp.net	0	3EB0EE6D5...	0	0	Hallo, danke der Nachfrage....	153120057...
160	205	49[redacted]51@s.whatsapp.net	0	3EB0EA287...	0	0	Im Moment bin ich gerade d...	153120060...
161	206	49[redacted]51@s.whatsapp.net	1	4EE93DEEF...	5	0	Oh, was denn für ein Test?□	153120064...
162	207	49[redacted]51@s.whatsapp.net	0	3EB04A80F...	0	0	Ich will einen WhatsApp-Cha...	153120072...
163	208	49[redacted]51@s.whatsapp.net	0	3EB09EA36...	0	0	Bin mal gespannt, wie das f...	153120074...
164	209	49[redacted]51@s.whatsapp.net	1	E71D54A92...	5	0	Na dann viel Erfolg! □□	153120145...

Abbildung 4.6: `messages`-Tabelle der `msgstore.db`-Datenbank nach WhatsApp-Neuregistrierung
Quelle: eigene Arbeit

4.1.3 Komplettes App-Backup über ADB

Nachdem das in Abschnitt 3.1.3 erwähnte Shellskript ausgeführt und den Anweisungen gefolgt wurde, befindet sich im Unterordner `tmp` das Archiv `whatsapp.tar`. Darin sind folgende Verzeichnisse und Dateien enthalten, die bei Bedarf einzeln oder gemeinsam extrahiert werden können:




49.....51-1519310250@g.us.j
49.....90@s.whatsapp.net.j
.trash
a01ecee7-cc8f-41f8-8797-3c911c09df8e
wam.wam
statistics
rc2
me
key
invalid_numbers
emoji
db
wa.db-wal
wa.db-shm
wa.db
msgstore.db-wal
msgstore.db-shm
msgstore.db
media.db-wal
media.db-shm
media.db
location.db-wal
location.db-shm
location.db
_jobqueue-WhatsAppJobManager-journal
_jobqueue-WhatsAppJobManager
hsmpacks.db-journal
hsmpacks.db

```
├── google_app_measurements_local.db-journal
├── google_app_measurements_local.db
├── google_app_measurements.db-journal
├── google_app_measurements.db
├── emojidictionary.db-journal
├── emojidictionary.db
├── chatsettingsbackup.db-journal
├── chatsettings.db-journal
├── chatsettings.db
├── axolotl.db-wal
├── axolotl.db-shm
└── axolotl.db
└── _manifest
```

4.1.4 Manuelle Entschlüsselung des Nachrichtenbackups

Die `key`- und die `msgstore.db.crypt12`-Datei wurden vom Testgerät auf den Auswerterechner kopiert und die entsprechenden Dateipfade in die Java-Klasse `WhatsAppBackupDecryptor` eingetragen (siehe Listing C.1 im Anhang C).
Der für die Entschlüsselung notwendige AES-Schlüssel ist in Abbildung 4.7, der Initialisierungsvektor in Abbildung 4.8 farblich hervorgehoben.

Offset	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	ASCII
00000000	AC ED 00 05 75 72 00 02 5B 42 AC F3 17 F8 06 08	....ur..[B......
00000010	54 E0 02 00 00 78 70 00 00 00 83 00 01 02 24 21	T....xp.......$!
00000020	BB 16 C1 CE 2B 11 15 52 6A B1 C1 14 D5 EA 1A 30	....+..Rj......0
00000030	AD C6 3A F5 68 9D 42 36 C9 62 68 2F 18 DE CD 3B	..:.h.B6.bh/...;
00000040	14 63 B0 A3 05 46 AF 4A B7 14 28 8C 27 EE D3 A5	.c...F.J..(.'...
00000050	F1 ED C4 E9 DB FE 76 3E CE E9 0B DE 73 28 60 4E	......v>....s(`N
00000060	1A 51 15 DC 4A 9E E6 B6 90 1F 4D 0F AA 24 00 00	.Q..J.....M..$..
00000070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 B5 F6	....
00000080	00 B9 A0 8D 1E 19 F6 A8 A3 A7 EF 71 CA 77 38 A3	...........q.w8.
00000090	82 B4 A1 A3 E3 8E 45 80 8A 41 EE B0 8F 3C	......E..A...<

Abbildung 4.7: AES-Schlüssel in der Datei `key`
Quelle: eigene Arbeit

Offset	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	ASCII
00000000	00	01	02	24	21	BB	16	C1	CE	2B	11	15	52	6A	B1	C1	...$!....+..Rj..
00000010	14	D5	EA	1A	30	AD	C6	3A	F5	68	9D	42	36	C9	62	68	....0..:.h.B6.bh
00000020	2F	18	DE	CD	3B	14	63	B0	A3	05	46	AF	4A	B7	14	28	/...;.c...F.J..(
00000030	8C	27	EE	88	52	D0	86	9B	7F	DB	85	91	40	B9	4B	49	.'..R.......@.KI
00000040	3B	A7	F4	77	AB	97	37	9E	E5	1B	52	9B	FD	2D	36	26	;..w..7...R..-6&
00000050	E1	99	A1	8D	16	93	5A	B5	2F	A6	D2	A0	CF	3C	C6	D1	......Z./....<..
00000060	BE	4C	D1	70	A3	44	BA	80	1C	CD	6A	3B	47	6E	F0	4F	.L.p.D....j;Gn.O
00000070	74	79	A4	D7	C5	F1	A4	C1	C7	2F	8E	33	60	BA	9F	C9	ty......./.3`...
00000080	FC	45	8F	9F	A2	EF	85	31	A9	5A	97	41	D3	90	89	A5	.E.....1.Z.A....
00000090	5A	CA	E1	45	25	A7	EE	16	83	D6	3A	B3	B6	D1	4D	EE	Z..E%.....:...M.

Abbildung 4.8: IV in der Datei `msgstore.db.crypt12` (Ausschnitt)
Quelle: eigene Arbeit

Nachdem das Java-Programm mit den Befehlen

```
javac WhatsAppBackupDecryptor.java
java WhatsAppBackupDecryptor
```

kompiliert und ausgeführt wurde, liegt im zuvor eingetragenen Verzeichnis die komplette entschlüsselte Nachrichtendatenbank `msgstore.db` vor. Ein Ausschnitt aus dem Inhalt ist (mit geschwärzten Telefonnummern) in Abbildung 4.9 zu sehen.

	_id	key_remote_jid	key_from_me	key_id	status	needs_push	data	timestamp
		Filtern	Filtern	Filtern	...	Filtern	Filtern	Filtern
157	202	49[redacted]...	0	call:3DEB1E...	6	0	NULL	153113026...
158	203	49[redacted]...	1	214FA9C75...	5	0	Hallo, Kommunikationspartner! 😀	153120040...
159	204	49[redacted]...	0	3EB0EE6D5...	0	0	Hallo, danke der Nachfrage. Mir geht es gut ☺.	153120057...
160	205	49[redacted]...	0	3EB0EA287...	0	0	Im Moment bin ich gerade dabei, Inhalt für einen Test zu generieren.	153120060...
161	206	49[redacted]...	1	4EE93DEEF...	5	0	Oh, was denn für ein Test?□	153120064...
162	207	49[redacted]...	0	3EB04A80F...	0	0	Ich will einen WhatsApp-Chat über die eingebaute E-Mail-Funktion exportieren, damit ...	153120072...
163	208	49[redacted]...	0	3EB09EA36...	0	0	Bin mal gespannt, wie das funktionieren wird	153120074...
164	209	49[redacted]...	1	E71D54A92...	5	0	Na dann viel Erfolg! □□	153120145...
165	210	49[redacted]...	1	A2D74173C...	5	0	NULL	153120206...
166	211	49[redacted]...	1	C8C75DBB...	5	0	Damit du weißt, wo ich bin:	153120336...
167	212	49[redacted]...	1	B491E3C9B...	1	0	NULL	153120340...
168	213	49[redacted]...	0	3EB0DF1AA...	0	0	Wenn du das lesen kannst, hast du die *.crypt12-Datei erfolgreich entschlüsselt!!! 👍	153120353...
169	214	49[redacted]...	1	2C2A21567...	5	0	😀	153120354...

Abbildung 4.9: `messages`-Tabelle der entschlüsselten `msgstore.db`
Quelle: eigene Arbeit

4.1.5 Vergleich

Wie bereits zu erkennen war, hat jede der erläuterten und getesteten Methoden sowohl Vor- als auch Nachteile für die forensische Extraktion von WhatsApp-bezogenen Daten von dem auszulesenden Android-Gerät $\mathcal{T}_{WA}$. In der nachfolgenden Tabelle werden diese aufgezeigt.

Vorteile	Nachteile
Export von Chatverläufen als Textdatei	
• kompletter Chatverlauf kann extrahiert werden (mit Option „Ohne Medien“, gleicher Stand wie in der App)	• Methode muss für jeden relevanten Chat einzeln angewandt werden • Export von Mediendateien schwierig (entweder mit wenigen Dateien und wenigen Nachrichten, oder ohne Dateien, aber mit allen Nachrichten) • kein Zugriff auf weitere WhatsApp-Daten (nur Nachrichtenverlauf) • je nach gewählter Exportvariante mehr oder weniger signifikante Veränderung von $\mathcal{T}_{WA}$ (z.B. bei Einrichtung eines neuen E-Mail-Kontos oder Versenden als WhatsApp-Nachricht)
Nachrichtenwiederherstellung durch Neuregistrierung	
• kompletter Nachrichtenbestand (aus allen Chats) inkl. Anrufliste zugänglich (`msgstore.db`) • wiederherzustellende Version wählbar (für neusten Stand vorher manuelles Backup notwendig)	• nur Nachrichten extrahierbar, keine weiteren Daten wie Logdateien oder Statistiken, da auf anderem Gerät neu angelegt • nur Telefonnummern (keine Namen) angezeigt, da Kontaktinformationen fehlen • durch Onlinezugriff Veränderung des Nachrichtenbestandes (Zustellen neuer bzw. Löschen alter Nachrichten) möglich • signifikante Veränderung von $\mathcal{T}_{WA}$ (WhatsApp ist darauf nicht mehr registriert)
Komplettes App-Backup über ADB	
• voller Zugriff auf alle Anwendungsdaten in der aktuellen Version	• nur bei Geräten mit einer Android-Version unter 7.0 möglich* • temporäre Veränderung von $\mathcal{T}_{WA}$ (Installation einer anderen WhatsApp-Version, wird aber wieder rückgängig gemacht)

Vorteile	Nachteile
Manuelle Entschlüsselung des Nachrichtenbackups	
• kompletter Nachrichtenbestand (aus allen Chats) inkl. Anrufliste zugänglich • Wiederherstellung von älteren Versionen der Datenbank möglich • keine Veränderung von $\mathcal{T}_{WA}$ (durch die Entschlüsselung als solche)	• nur Zugriff auf Nachrichtenverlauf, keine weiteren Daten • Zugriff auf `key`-Datei notwendig (liegt im geschützten Bereich)

[*] Bei späteren Versionen wurde ein Sicherheitsmechanismus eingebaut, um Downgrades von normalen Anwendungen ohne vorheriges Löschen der Anwendungsdaten zu verhindern, siehe [Kly16].

Tabelle 4.1: Vor- und Nachteile der verschiedenen Extraktionsmethoden
Quelle: eigene Arbeit

Für einen einfachen Vergleich werden die Eigenschaften der einzelnen Methoden (nummeriert von 1 bis 4 nach der Reihenfolge aus Tabelle 4.1) in der Übersicht aus Tabelle 4.2 zusammengefasst und mit dem Ideal einer physischen Datensicherung (phys.) verglichen.

Methode			1	2	3	4	phys.
extrahierbare Daten	Nachrichten	einzelner Chat	ja[a]	ja	ja	ja	ja
		komplett	nein	ja	ja	ja	ja
	Kontakte		nein	nein	ja	nein	ja
	weitere Daten		nein	nein	alle	nein	alle
ohne Veränderung			nein[b]	nein	nein[c]	ja[d]	nein[e]
Android-Version			alle	alle	bis 6.0	alle	alle

[a] Je nachdem, ob ein Export mit oder ohne Medien stattfindet, wird der komplette Nachrichtenverlauf ohne oder ein Teil desselbigen mit Mediendateien extrahiert.
[b] Je nach gewählter Exportvariante.
[c] Die Veränderung ist nur temporär.
[d] Die `key`-Datei wird benötigt. Evtl. ist zum Beschaffen dieser eine Veränderung notwendig.
[e] Je nachdem, wie die physische Sicherung durchgeführt wird (idealerweise ohne Veränderung).

Tabelle 4.2: Vergleich der Eigenschaften aller vier Extraktionsmethoden
Quelle: eigene Arbeit

4.2 Analyse von Logdateien in Bezug auf WhatsApp

4.2.1 Inhalte von Logcat und WhatsApp Log

Nach der Analyse der beiden Logdateien mit der in Abschnitt 3.2.1 erläuterten Methode wurden zu den durchgeführten Aktivitäten folgende Logeinträge in Verbindung gebracht.
Dabei wird für jedes Ereignis die entsprechende, im WhatApp Timeline Extractor genutzte $\mathcal{ID}_{Ereignis}$ sowie der relevante Teil (ohne Zeitstempel) des Logeintrags, der Logtyp und eine Beispiel-Logzeile[1] aufgeführt. Außerdem werden die enthaltenen Metadaten genannt.

4.2.1.1 WhatsApp-Aktionen

- **einen Chat aus dem Hauptbildschirm heraus öffnen:**

$\mathcal{ID}_{Ereignis}$	`"OpenChat"`
Logeintrag	`[...] conversations/click/jid <JID> [...]`
Metadaten	• JID des geöffneten Chats
Logtyp	Whatapp Log
Beispiel	`2018-06-27 13:06:06.262 LL_I W [1:main] conversations/click/jid 491234567890@s.whatsapp.net pos=3`

Tabelle 4.3: Logeintrag beim Öffnen eines Chats
Quelle: eigene Arbeit

[1] Bei den Beispielen wurden jeweils die JID sowie die Nachrichten-ID durch Beispielzahlen ersetzt.

- **einen Chat mit neuen, ungelesenen Nachrichten öffnen:**

$\mathcal{ID}_{Ereignis}$	`"ChatSeen"`
Logeintrag	`[...]:main] msgstore/setchatseen/<JID>/[...]`
Metadaten	• JID des geöffneten Chats
Logtyp	Whatapp Log
Beispiel	`2018-06-27 13:09:21.333 LL_I W [1:main] msgstore/setchatseen/491234567890@s.whatsapp.net/1/1/0/null/0`

Tabelle 4.4: Logeintrag beim Lesen neuer Nachrichten
Quelle: eigene Arbeit

Hinweis: Während bei allen anderen Ereignissen die Threadbezeichnung keine Rolle spielt (bzw. immer gleich ist), muss es sich beim Lesen neuer Nachrichten immer um den `main`-Thread handeln, da sonst mehr als ein Eintrag dem Ereignis zugeordnet wird.

- **eine neue Textnachricht versenden:**

$\mathcal{ID}_{Ereignis}$	`"SendMessage"`
Logeintrag	`[...] msgstore/add/send; key=Key[id=<MsgID>[...]remote_jid=<JID>][...]`
Metadaten	• JID des Kommunikationspartners • ID der neuen Nachricht (`MsgID`)
Logtyp	Whatapp Log
Beispiel	`2018-06-27 13:06:18.545 LL_I W [1353:Messages Async Commit Thread] msgstore/add/send; key=Key[id=1234567890ABCDEF1234567890ABCDEF, from_me=true, remote_jid=491234567890@s.whatsapp.net]; media_wa_type=0; status=0`

Tabelle 4.5: Logeintrag beim Senden einer neuen Nachricht
Quelle: eigene Arbeit

- **eine neue Mediennachricht versenden:**

Es wird der gleiche Logeintrag wie in Tabelle 4.5 erstellt, eine Unterscheidung ist durch `media_wa_type` möglich, z.B. 1 für Bilder und 3 für Videos (siehe auch Abschnitt 2.2.4.12).

- **eine Textnachricht empfangen:**

$\mathcal{ID}_{Ereignis}$	`"ReceiveMessage"`
Logeintrag	`[...] xmpp/reader/read/message <JID> <MsgID> [...]`
Metadaten	• JID des Kommunikationspartners • ID der neuen Nachricht
Logtyp	Whatapp Log
Beispiel	`2018-06-27 13:09:08.180 LL_I W [1420:ReaderThread] xmpp/reader/read/message 491234567890@s.whatsapp.net 1234567890ABCDEF1234567890ABCDEF none 180 null 0`

Tabelle 4.6: Logeintrag beim Empfangen einer neuen Nachricht
Quelle: eigene Arbeit

- **eine Mediennachricht empfangen:**

Es wird der gleiche Logeintrag wie in Tabelle 4.6 erstellt, eine Unterscheidung ist mit Hilfe der Spalte `media_wa_type` der `messages`-Tabelle in `msgstore.db` möglich (siehe Abschnitt 2.2.4.12).

- **eine Nachricht „für mich löschen“:**

$\mathcal{ID}_{Ereignis}$	`"DeleteMessage"`
Logeintrag	`[...] thumbnailmsgstore/deleteMessageThumbnail/<MsgID> /[...]`
Metadaten	• ID der gelöschten Nachricht
Logtyp	Whatapp Log
Beispiel	`2018-06-08 11:02:37.321 LL_I W [1180:Messages Async Commit Thread] thumbnailmsgstore/ deleteMessageThumbnail/1234567890ABCDEF1234567890ABCDEF/ 1`

Tabelle 4.7: Logeintrag beim Löschen einer Nachricht „für mich“
Quelle: eigene Arbeit

Hinweis: Die Zuordnung zu der entsprechen JID ist mit Hilfe von `"OpenChat"`-Ereignissen möglich. Die Nachricht wurde immer in dem zuletzt geöffneten Chat gelöscht.
Berücksichtigt werden muss, dass auch beim automatischen Löschen von Statusmeldungen durch WhatsApp ein solcher Eintrag generiert wird. Beide Varianten können

nicht direkt unterschieden werden. Da die ID der gelöschten Nachricht allerdings enthalten ist, kann möglicherweise eine Verbindung zu empfangenen Status hergestellt werden (siehe auch Klasse `WALog` im Abschnitt 3.2.2.3, Seite 44).
Da WhatsApp periodisch abgelaufene Statusmeldungen aufräumt, findet man außerdem vor den eigentlichen Löscheinträgen den in Tabelle 4.8 dargestellten Eintrag in der Logdatei. Dieser weist auf die Anzahl n_{Status} der Statusmeldungen hin, die in Kürze gelöscht werden, das heißt, diese n_{Status} `"DeleteMessage"`-Einträge gehören zu keiner normalen Nachricht.

$\mathcal{ID}_{Ereignis}$	`"DeleteStatus"`
Logeintrag	`[...] statusmsgstore/deleteoldstatuses/delete[...] deleted:<`n_{Status}`>[...]`
Metadaten	• Anzahl der zu löschenden Status n_{Status}
Logtyp	Whatapp Log
Beispiel	`2018-07-18 12:40:00.870 LL_I D [814:WhatsApp Worker #2] statusmsgstore/deleteoldstatuses/delete total:13 deleted:4 last batch:4`

Tabelle 4.8: Logeintrag mit dem Hinweis auf zu löschende Statusmeldungen
Quelle: eigene Arbeit

- **eine Nachricht „für alle löschen“:**

$\mathcal{ID}_{Ereignis}$	`"DeleteMessageAll"`
Logeintrag	`[...] msgstore/edit/revoke send [...] key=Key[id=<MsgID> [...] remote_jid=<JID>]`
Metadaten	• JID des Kommunikationspartners • ID der gelöschten Nachricht
Logtyp	Whatapp Log
Beispiel	`2018-07-23 07:19:57.235 LL_I W [1:main] msgstore/edit/revoke send deleteMedia=false key=Key[id=1234567890ABCDEF1234567890ABCDEF, from_me=true, remote_jid=491234567890@s.whatsapp.net]`

Tabelle 4.9: Logeintrag beim Löschen einer Nachricht „für alle“
Quelle: eigene Arbeit

Hinweis: Dieser Eintrag besagt, dass eine Nachricht wieder aus der Nachrichtendatenbank entfernt werden soll. Er tritt auch auf, wenn nicht tatsächlich eine Nachricht

gelöscht wurde, sondern ein veröffentlichter Status abgelaufen ist. Die Differenzierung ist anhand der enthaltenen JID möglich (bei normalen Nachrichten ist diese ungleich `status@broadcast`).

- **einen Sprachanruf starten:**

$\mathcal{ID}_{Ereignis}$	`"OutCall"`
Logeintrag	`[...] app/startOutgoingCall/[...] video call:<type>[...]`
Metadaten	• Art des Anrufs (`type`)
Logtyp	Whatapp Log
Beispiel	`2018-06-08 11:03:44.284 LL_W W [1:main] app/ startOutgoingCall/from 8 video call:false, smallerBtn: false`

Tabelle 4.10: Logeintrag bei ausgehenden Anrufen
Quelle: eigene Arbeit

Hinweis: In diesem Logeintrag findet sich nur die Information, *dass* ein Anruf gestartet wurde, nicht mit wem.
Da dieser jedoch analog zu normalen abgehenden Nachrichten in die Datenbank eingetragen wird, findet sich später in der Logdatei noch ein zu dem Anruf gehörender `"SendMessage"`-Eintrag. Die JID des Kommunikationspartners sowie die Nachrichten-ID des Anrufes (beginnend mit `call:`) können daraus extrahiert werden.

- **einen Videoanruf starten:**

Es wird der gleiche Logeintrag wie in Tabelle 4.10 erstellt, eine Unterscheidung ist durch den Wahrheitswert `type` möglich (bei Videoanrufen `true`).

- **mittels Sprachanruf angerufen werden:**

$\mathcal{ID}_{Ereignis}$	`"InCall"`
Logeintrag	`[...] xmpp/reader/on-call-offer stanzaKey=[StanzaKey from=<JID> [...]] callId=<MsgID> [...]`
Metadaten	• JID des Anrufers • Nachrichten-ID Anrufes
Logtyp	Whatapp Log
Beispiel	`2018-06-11 08:10:35.957 LL_I W [560:ReaderThread] xmpp/reader/on-call-offer stanzaKey=[StanzaKey from=491234567890@s.whatsapp.net cls=call id=0ECB8F22C4092D90B592B843810D48D4 type=offer] callId=1234567890ABCDEF1234567890ABCDEF epochTimeMillis=1528697435000 elapsedTime=0 audioEncodings=[opus, opus] rates=[16000, 8000] videoEncoding=null videoOrientation=0 endpoints=[] endpointPriorities=[] endpointEnablePortPredicting=[] netMedium=3 relayToken=[...] relayEndpoints=[...] rte=[...] voipOptions=VoipOptions userRate[...]`

Tabelle 4.11: Logeintrag bei ausgehenden Anrufen
Quelle: eigene Arbeit

- **mittels Videoanruf angerufen werden:**

Es wird der gleiche Logeintrag wie in Tabelle 4.11 erstellt, eine Unterscheidung ist anhand der `videoEncoding` möglich. Bei einem Sprachanruf ist (wie in Tabelle 4.11) dazu kein Eintrag vorhanden, bei Videoanrufen wird in eckigen Klammern mindestens eine Codierung gegeben, z.B. `[h.264, vp8, vp8/h.264]`.

- **einen eingehenden Sprachanruf annehmen:**

$\mathcal{ID}_{Ereignis}$	`"AcceptInCall"`
Logeintrag	`[...] wa_call_utils. <time> EVENT: Call accept sent`
Metadaten	keine
Logtyp	Whatapp Log
Beispiel	`2018-06-27 13:10:09.050 LL_I W [1455:VoIP Signaling Thread] wa_call_utils. 13:10:09 EVENT: Call accept sent`

Tabelle 4.12: Logeintrag bei der Annahme eines eingehenden Anrufs
Quelle: eigene Arbeit

- **einen eingehenden Videoanruf annehmen:**

Es wird der gleiche Logeintrag wie in Tabelle 4.12 erstellt, eine Unterscheidung ist nur anhand des vorangehenden `"InCall"`-Ereignisses möglich.

- **ein ausgehender Sprachanruf wird angenommen:**

$\mathcal{ID}_{Ereignis}$	`"AcceptOutCall"`
Logeintrag	`[...] wa_call_utils. <time> EVENT: Call accept received`
Metadaten	keine
Logtyp	Whatapp Log
Beispiel	`2018-06-08 11:03:56.710 LL_I W [1335:VoIP Signaling Thread] wa_call_utils. 11:03:56 EVENT: Call accept received`

Tabelle 4.13: Logeintrag bei der Annahme eines ausgehenden Anrufs
Quelle: eigene Arbeit

- **ein ausgehender Videoanruf wird angenommen:**

Es wird der gleiche Logeintrag wie in Tabelle 4.13 erstellt, eine Unterscheidung ist nur anhand des vorangehenden `"OutCall"`-Ereignisses möglich.

- **einen neuen Textstatus veröffentlichen:**

$\mathcal{ID}_{Ereignis}$	`"SendStatus"`
Logeintrag	`[...] xmpp/writer/write/message-encrypted; key=Key[id=<MsgID>[...]remote_jid=status@broadcast][...]`
Metadaten	• Nachrichten-ID des Status
Logtyp	Whatapp Log
Beispiel	`2018-06-27 09:39:19.648 LL_I W [969:WriterThread] xmpp/writer/write/message-encrypted; key=Key[id= 1234567890ABCDEF1234567890ABCDEF, from_me=true, remote_jid=status@broadcast]; originalTimestamp=0; participant=null; groupParticipantHash=1:JIq92gRp; mediaType=null; webAttribute=NONE; encryptedMessage=EncryptedMessage{ciphertextVersion=2, ciphertextType=2};[...]`

Tabelle 4.14: Logeintrag beim Veröffentlichen eines neuen Status
Quelle: eigene Arbeit

- **einen neuen Foto-, Video- oder GIF-Status veröffentlichen:**

Es wird der gleiche Logeintrag wie in Tabelle 4.14 erstellt, eine Unterscheidung ist anhand des `mediaType` möglich (`null` für Text, sonst `image`, `video` oder `gif`).

4.2.1.2 Android-Aktionen

- **den Bildschirm durch Betätigen des Einschaltknopfes einschalten:**

Das Ein- bzw. Ausschalten sowie das Entsperren des Bildschirmes wird sowohl in der WhatsApp-Logdatei als auch im Android Logcat vermerkt, jedoch auf unterschiedliche Weise.
Während bei Letzterem jeweils verschiedene Logeinträge erstellt werden, ist die Zeile im WhatsApp Log immer die gleiche, unterschieden wird anhand eines `Status`, der die Werte `on`, `off` oder `unlock` besitzen kann.

$\mathcal{ID}_{Ereignis}$	`"Unlock"`
Logeintrag	`[...] ScreenLockReceiver; tag=<Status>[...]`
Metadaten	• Bildschirmstatus (`Status`)
Logtyp	WhatsApp Log
Beispiel	`2018-07-13 11:31:20.434 LL_I W [1:main] ScreenLockReceiver; tag=on; locked=true; oldLocked=true`

Tabelle 4.15: Logeintrag beim Einschalten des Telefons im WhatsApp Log
Quelle: eigene Arbeit

$\mathcal{ID}_{Ereignis}$	`"ScreenOn"`
Logeintrag	`[...]DisplayPowerController[...] Unblocked screen on after [...]`
Metadaten	keine
Logtyp	Android Logcat
Beispiel	`06-08 11:01:44.172 I/DisplayPowerController( 525): Unblocked screen on after 382 ms`

Tabelle 4.16: Logeintrag beim Einschalten des Telefons im Logcat
Quelle: eigene Arbeit

• **Bildschirmaktivierung durch andere Ursache:**

Im Android-Log wird der gleiche Eintrag wie in Tabelle 4.16 erstellt, die Unterscheidung ist durch einen vorangegangenen Eintrag möglich:

$\mathcal{ID}_{Ereignis}$	`"WakeLock"`
Logeintrag	`[...]PowerManagerService[...] acquireWakeLockInternal: [...] tag="<WakeLockTag>"[...]`
Metadaten	• Ursache des Einschaltens (`WakeLockTag`)
Logtyp	Android Logcat
Beispiel	`06-08 11:01:38.036 D/PowerManagerService( 525): acquireWakeLockInternal: lock=636501148, flags=0x1, tag="PhoneWindowManager.mPowerKeyWakeLock", ws=null, uid=1000, pid=525`

Tabelle 4.17: Logeintrag mit Ursache für eine Bildschirmaktivierung
Quelle: eigene Arbeit

Jedem Eintrag beim Einschalten geht ein solcher „WakeLock“-Eintrag voran. Der in Tabelle 4.17 gekennzeichnete `WakeLockTag` gibt einen Hinweis darauf, wodurch das Display aktiviert wurde. Folgende Tags wurden bestimmt:

Betätigen des Einschaltknopfes:	`PhoneWindowManager.mPowerKeyWakeLock`
eingehender Anruf:	`InCallWakeLockController`
eingehender WhatsApp-Sprachanruf:	`WhatsApp VoiceService`
eingehender WhatsApp-Videoanruf:	wie bei WhatsApp-Sprachanrufen
eingehende SMS:	`GsmInboundSmsHandler`
abgelaufener Timer:	`AlarmClock`
klingelnder Wecker:	wie bei abgelaufenen Timern

Hinweis: Nicht nur beim Einschalten wird der in Tabelle 4.17 aufgeführte Eintrag generiert, sondern auch in vielen anderen Fällen (mit anderem `WakeLockTag`). Die relevanten Einträge können jedoch mit Hilfe einer Whitelist herausgefiltert werden. Im WhatsApp Timeline Extractor kommt eine Whitelist mit allen o.g. Tags zum Einsatz.
Andere Hinweise auf die Ursache des Einschaltens wurden nicht gefunden.

- **den Bilschirm ausschalten:**

Im WhatsApp-Log wird der gleiche Eintrag wie in Tabelle 4.15 erstellt, eine Unterscheidung ist durch den enthaltenen `Status` möglich.
Im Logcat wird ein anderer Eintrag erstellt:

$\mathcal{ID}_{Ereignis}$	`"ScreenOff"`
Logeintrag	`[...]WindowManager[...] Screen turned off...`
Metadaten	keine
Logtyp	Android Logcat
Beispiel	`06-08 11:01:38.197 I/WindowManager( 525): Screen turned off...`

Tabelle 4.18: Logeintrag beim Ausschalten des Bildschirms
Quelle: eigene Arbeit

- **das Gerät durch Codeeingabe entsperren:**

Das Entsperren des Gerätes erfolgt in zwei Schritten: erst wird der notwendige Code eingegeben, dann wird das Telefon entsperrt. Beide Schritte spiegeln sich im Logcat wider (siehe Tabellen 4.19 und 4.20).
Im WhatsApp-Log kann nur der zweite Schritt durch den in Tabelle 4.15 aufgeführten Eintrag nachvollzogen werden.

$\mathcal{ID}_{Ereignis}$	`"LockscreenAction"`
Logeintrag	`[...]sysui_lockscreen_gesture[...]`
Metadaten	keine
Logtyp	Android Logcat
Beispiel	`06-27 13:05:43.777 I/sysui_lockscreen_gesture( 707): [1,188,2996]`

Tabelle 4.19: Logeintrag bei der Eingabe des Entsperrcodes
Quelle: eigene Arbeit

$\mathcal{ID}_{Ereignis}$	`"Unlocked"`
Logeintrag	`[...]screen_toggled[...]2`
Metadaten	keine
Logtyp	Android Logcat
Beispiel	`07-14 17:45:15.734 I/screen_toggled(  703): 2`

Tabelle 4.20: Logeintrag beim Entsperren des Telefons
Quelle: eigene Arbeit

Hinweis: Ist kein Sperrcode festgelegt, entfallen beide Schritte und das Gerät ist nach Einschalten des Displays sofort entsperrt.

- **den Entsperrcode falsch eingeben:**

In diesem Fall wird nur für das Eingeben des Codes, nicht aber für das erfolgreiche Entsperren ein Eintrag erzeugt. Man kann also das Ereignis aus Tabelle 4.19, allerdings ohne nachfolgendes `"Unlocked"`-Ereignis extrahieren.

- **eine App starten:**

$\mathcal{ID}_{Ereignis}$	`"StartApp"`
Logeintrag	`[...]am_create_activity[...] [[...]<Activityname>[...]`
Metadaten	• Name der Activity (und damit auch der App)
Logtyp	Android Logcat
Beispiel	`06-27 13:05:50.634 I/am_create_activity(  525): [0,1025744689,383,com.whatsapp/.Main, android.intent.action.MAIN,NULL,NULL,270532608]`

Tabelle 4.21: Logeintrag beim Starten einer Anwendung
Quelle: eigene Arbeit

- **eine App minimieren:**

$\mathcal{ID}_{Ereignis}$	`"PauseApp"`
Logeintrag	`[...]am_pause_activity[...] [[...]<Activityname>[...]`
Metadaten	• Name der Activity (und damit auch der App)
Logtyp	Android Logcat
Beispiel	`06-27 13:05:52.065 I/am_pause_activity( 525): [0,1025744689,com.whatsapp/.Main]`

Tabelle 4.22: Logeintrag beim Minimieren einer Anwendung
Quelle: eigene Arbeit

- **eine App wieder in den Vordergrund holen:**

$\mathcal{ID}_{Ereignis}$	`"ResumeApp"`
Logeintrag	`[...]am_resume_activity[...] [[...]<Activityname>[...]`
Metadaten	• Name der Activity (und damit auch der App)
Logtyp	Android Logcat
Beispiel	`06-27 13:06:42.524 I/am_resume_activity( 525): [0,321500833,383,com.whatsapp/.HomeActivity]`

Tabelle 4.23: Logeintrag beim Wiederherstellen einer Anwendung
Quelle: eigene Arbeit

- **zwischen zwei Apps wechseln:**

Dieses Ereignis wird, unterteilt in das Minimieren der ersten und anschließendes Starten bzw. Wiederherstellen der zweiten App, in das Android-Systemlog eingetragen. Der WhatsApp Timeline Extractor führt dazu zwei passende Ereignisse zusammen (siehe Abschnitt 3.2.2.3).

- **eine App schließen:**

$\mathcal{ID}_{Ereignis}$	`"StopApp"`
Logeintrag	`[...]am_destroy_activity[...] [[...]<Activityname>[...]`
Metadaten	• Name der Activity (und damit auch der App)
Logtyp	Android Logcat
Beispiel	`06-27 13:06:42.965 I/am_destroy_activity( 525): [0,125992624,383,com.whatsapp/.Conversation,finish-imm]`

Tabelle 4.24: Logeintrag beim Schließen einer Anwendung
Quelle: eigene Arbeit

4.2.2 Ereigniserkennung des WhatsApp Timeline Extractors

Um die Zuverlässigkeit der regulären Ausdrücke und des WhatsApp Timeline Extractors zu testen, wurden verschiedene, definierte Aktivitäten auf dem Testgerät nacheinander ausgeführt und mit den extrahierten Ereignissen verglichen.

Vorgehen Der Logcat-Inhalt wurde über ADB ausgelesen und die WhatsApp-Logdatei sowie die Kontaktdatenbank `wa.db` mit Hilfe der in Abschnitt 3.1.3 erläuterten Methode extrahiert.
Der Start der Aktivitätenserie war am 14.07.2018 um 17:45 Uhr und zu Beginn war der Bildschirm ausgeschaltet. Außerdem lief die WhatsApp-Anwendung bereits im Hintergrund.
Nacheinander wurden folgende Aktionen durchgeführt:

1. Bildschirm einschalten
2. Entsperrcode richtig eingeben und dadurch das Telefon entsperren
3. Kamera-App starten und ein Foto schießen
4. Kamera-App wieder minimieren
5. WhatsApp starten
6. Chat des Kontaktes „Kommunikationspartner“ öffnen, in dem es neue, ungelesene Nachrichten gibt
7. eine Nachricht schreiben und absenden

8. zurück in den Hauptbildschirm und einen weiteren Chat („Noch Jemand 😎“) öffnen
9. eine Nachricht schreiben und absenden
10. die letzte geschriebene Nachricht „Für mich löschen“
11. eine weitere Nachricht versenden
12. WhatsApp minimieren
13. die Android-Einstellungen öffnen
14. eine Nachricht von „Kommunikationspartner“ empfangen
15. über die Übersicht der aktiven Anwendungen zurück auf WhatsApp wechseln
16. neuen Fotostatus erstellen und veröffentlichen
17. Chat der Gruppe „Schöne Gruppe“ öffnen
18. Nachricht schreiben und absenden
19. WhatsApp minimieren
20. Telefon sperren
21. eingehender WhatsApp-Sprachanruf
22. Anruf annehmen und einige Sekunden telefonieren
23. nachdem der Bildschirm wieder ausgegangen ist, erneut einschalten
24. Entsperrcode falsch eingeben und das Telefon nicht entsperren
25. Bildschirm ausschalten

Nach dem Einlesen der entsprechenden Dateien in WATE wurden alle Ereignisse aus dem betroffenen Zeitraum als CSV-Datei in englischer Sprache exportiert. Die enthaltenen Einträge sind (mit geschwärzten Telefonnummern) in Abbildung 4.10 auf Seite 76 dargestellt.
Beim Erstellen der Zeitleiste kam die folgende Blacklist mit Apps, deren Logeinträge ignoriert wurden, zum Einsatz:

- com.zte.mifavor.launcher
- com.android.systemui

Auswertung Beim Vergleich der Exportdatei mit den durchgeführten Aktivitäten wird deutlich, dass alle tatsächlich stattgefundenen Ereignisse durch WATE extrahiert werden, sodass die Aktivitätenserie gut nachvollzogen werden kann.
Auch die korrekten Telefonnummern bzw. Kontaktnamen sind enthalten, wobei allerdings deutlich wird, dass der Smiley „😎“ zwar im Programm ähnlich angezeigt wird (siehe z.B. Abbildung A.4 im Anhang A), im Export (Abbildung 4.10) allerdings nicht dekodiert werden kann.

Das Ein- bzw. Ausschalten des Telefons kommt sowohl im WhatsApp-Log als auch im Android Logcat vor, weshalb diese Ereignisse auch jeweils zweimal extrahiert werden (z.B. die beiden letzten Ereignisse in Abbildung 4.10).
Allerdings wird das Einschalten ganz zu Beginn nur im WhatsApp Log gefunden und der entsprechende Eintrag ist auch in der Logdatei nicht enthalten. Dies lässt sich durch die begrenzte Kapazität des Logcat und die Überschreibung alter Einträge durch neu hinzukommende erklären.
Außerdem fehlt die Information, dass mit Hilfe der Kamera-App ein Foto geschossen wurde. Das liegt daran, dass im Android-Log kein Eintrag dazu angelegt wird und dieses Ereignis außerdem durch das Programm nicht unterstützt wird (für Informationen zu unterstützten Ereignissen siehe Abschnitt 3.2.2.3).

Das Minimieren einer App und Wiederherstellen einer anderen (wie z.B. in den Schritten 4 und 5) wird korrekterweise als `"SwitchApp"`-Ereignis erkannt (bei 17:45:33:131 in Abbildung 4.10).
Weiterhin fällt auf, dass der eingehende WhatsApp-Sprachanruf aus Schritt 21, durch den der Bildschirm aktiviert wird, mehrere Ereignisse erzeugt, sowohl im WhatsApp- also auch im Android-Log. Beginnend bei 17:48:43:319 wurden folgende Ereignisse extrahiert:

1. eingehender Sprachanruf (WhatsApp Log)
2. Wiederherstellen der WhatsApp-Anwendung (Logcat)
3. Aktivieren des Bildschirms durch eingehenden WhatsApp-Anruf (Logcat)
4. Aktivieren des Bildschirms (WhatsApp Log)

Dadurch wird deutlich, dass Anwendungen auch ohne Zutun des Nutzers gestartet bzw. wiederhergestellt werden können.

Timestamp	Event	Contact JID	Contact name	Message ID	App	(Sub) Activity	New App	Power on cause	Source	Affected line(s)
2018-07-14 17:45:11.086	Screen on								WhatsApp Log File	2751
2018-07-14 17:45:12.320	Tried to unlock the phone								Android Logcat	7
2018-07-14 17:45:15.734	Unlocked the phone								Android Logcat	9
2018-07-14 17:45:19.057	Started an app				com.android.camera2	com.android.camera.CameraLauncher			Android Logcat	19
2018-07-14 17:45:33.131	Switched between two apps				com.android.camera2		com.whatsapp		Android Logcat	51, 74
2018-07-14 17:45:37.322	Returned to a Chat	4917[redacted]	Kommunikationspartner						WhatsApp Log File	2763
2018-07-14 17:45:38.474	Read new message(s)	4917[redacted]	Kommunikationspartner						WhatsApp Log File	2802
2018-07-14 17:45:43.410	Sent a message	4917[redacted]	Kommunikationspartner	F4C9ACCD2					WhatsApp Log File	2820
2018-07-14 17:45:51.949	Opened a Chat	4915[redacted]	Noch Jemand 🔲						WhatsApp Log File	2878
2018-07-14 17:46:20.467	Sent a message	4915[redacted]	Noch Jemand 🔲	651615B57					WhatsApp Log File	2926
2018-07-14 17:46:31.272	Deleted a message	4915[redacted]	Noch Jemand 🔲	651615B57					WhatsApp Log File	2878, 2964
2018-07-14 17:46:44.372	Sent a message	4915[redacted]	Noch Jemand 🔲	E8244F1FD0					WhatsApp Log File	2971
2018-07-14 17:46:50.166	Sent an app to background				com.whatsapp	.Conversation			Android Logcat	326
2018-07-14 17:46:54.520	Started an app				com.android.settings	.Settings			Android Logcat	431
2018-07-14 17:47:08.044	Received new message	4917[redacted]	Kommunikationspartner	43226F8590					WhatsApp Log File	3013
2018-07-14 17:47:20.005	Switched between two apps				com.android.settings		com.whatsapp		Android Logcat	825, 879
2018-07-14 17:47:43.936	Published an image status	status@bro		CE77C8FC50					WhatsApp Log File	3201
2018-07-14 17:47:46.367	Opened a Chat	4917[redacted]	Schöne Gruppe						WhatsApp Log File	3212
2018-07-14 17:48:20.625	Sent a message	4917[redacted]	Schöne Gruppe	5B9DC6025					WhatsApp Log File	3274
2018-07-14 17:48:28.792	Sent an app to background				com.whatsapp	.HomeActivity			Android Logcat	2467
2018-07-14 17:48:31.715	Screen off								Android Logcat	2697
2018-07-14 17:48:32.175	Screen off								WhatsApp Log File	3381
2018-07-14 17:48:43.319	Incoming voice call	4917[redacted]	Kommunikationspartner	call:6A5513E					WhatsApp Log File	3384
2018-07-14 17:48:44.117	Resumed an app				com.whatsapp	.voipcalling.VoipActivityV2			Android Logcat	2951
2018-07-14 17:48:45.669	Screen on							Incoming WhatsApp Voice or Video Call	Android Logcat	2967, 3236
2018-07-14 17:48:45.858	Screen on								WhatsApp Log File	3896
2018-07-14 17:48:56.192	Accepted incoming call								WhatsApp Log File	4107
2018-07-14 17:49:03.556	Sent an app to background				com.whatsapp	.voipcalling.VoipActivityV2			Android Logcat	4191
2018-07-14 17:49:06.239	Screen off								Android Logcat	5241
2018-07-14 17:49:06.394	Screen off								WhatsApp Log File	4693
2018-07-14 17:49:21.784	Screen on							Power Button Press	Android Logcat	5862, 6013
2018-07-14 17:49:21.871	Screen on								WhatsApp Log File	4707
2018-07-14 17:49:23.375	Tried to unlock the phone								Android Logcat	6488
2018-07-14 17:49:29.471	Screen off								Android Logcat	7405
2018-07-14 17:49:29.733	Screen off								WhatsApp Log File	4709

Abbildung 4.10: Export der von WATE erstellten Zeitleiste für definierte Ereignisse
Quelle: eigene Arbeit

5 Diskussion

5.1 Zusammenfassung

5.1.1 Grundlagen

Im ersten Kapitel dieser Arbeit wurden die aus forensischer Sicht relevanten Grundlagen von WhatsApp erläutert. Dabei wurde neben den verfügbaren Funktionen vor allem auch auf die technischen Einzelheiten des verschlüsselten Versands von Nachrichten und die auf dem Gerät abgelegten Daten eingegangen.

Es wurde deutlich, dass die Ende-zu-Ende-Verschlüsselung der Nachrichten das *Signal Protocol* verwendet, das mit *Curve25519*-Schlüsseln arbeitet. Durch eine stetige Veränderung dieser Schlüssel, sowohl beim Senden einer Nachricht als auch beim Empfangen einer Antwort, wird selbst bei einer Kompromittierung der Schaden so gering wie möglich gehalten, da alte Schlüssel nicht aus neueren berechnet werden können („Forward Secrecy“ [Wha17]).

Neben den Eins-zu-eins-Nachrichten gibt es in WhatsApp jedoch auch Gruppenchats, die laut einer aktuellen Studie der Ruhr-Universität Bochum [RMS18] nicht so gut geschützt werden.
Demnach werden zwar die eigentlichen Inhalte von Gruppennachrichten genauso sicher verschlüsselt, allerdings sind Gruppenmodifikationsoperationen (wie z.B. das Hinzufügen oder Entfernen von Mitgliedern oder Administratoren) zwar auf der Transportschicht (*TLS*), nicht aber Ende-zu-Ende-verschlüsselt.
Da bei solchen Nachrichten die Identität des Absenders nicht überprüft wird, ist ein Fälschen (englisch: *spoofing*) möglich. Ein Angreifer kann also Modifikationen in fremdem Namen durchführen, ohne selbst überhaupt Gruppenmitglied sein zu müssen. Auf diese Weise ist es unter Vortäuschung der Identität eines Administrators z.B. möglich, neue Mitglieder zu Gruppen hinzuzufügen. Diese können dann unbemerkt neue Nachrichten mitlesen (wegen der Forward Secrecy jedoch keine zurückliegenden).
Des Weiteren kann ein solcher Angreifer Nachrichten auf Serverseite verwerfen oder die Reihenfolge verändern und die entsprechenden Empfangsbestätigungen vortäu-

schen, wodurch einzelne Mitglieder diese Manipulationen nicht bemerken.
Voraussetzung für alle genannten Angriffe ist jedoch die Kontrolle über den WhatsApp-Server bzw. das Brechen der Verschlüsselung auf der Transportschicht, was ein selbstständiges Senden von Nachrichten an die Clients ermöglicht [RMS18].

Beim Betrachten der durch WhatsApp auf Android-Geräten gespeicherten Daten fällt auf, dass viele Informationen über den Nutzer (z.B. in den Dateien `registration.RegisterPhone.xml` oder `com.whatsapp_preferences.xml`) bzw. über die Nutzung der Anwendung (z.B. in `com.google.android.gms.measurements.prefs.xml`, `statistics` oder der Logdatei `whatsapp.log`) abgelegt werden.
Alle diese Dateien befinden sich jedoch in dem normalerweise geschützten Speicherbereich des Telefons, auf das nur WhatsApp Zugriff hat. Lediglich verschlüsselte Backups der Nachrichtendatenbank und die verschickten oder empfangenen Mediendateien sind für den Nutzer zugänglich. Letztere können allerdings ohne die Klartext-Datenbank oder den Chatverlauf auf dem Telefon nicht dem entsprechenden Kontakt oder Nachricht zugeordnet werden.

5.1.2 Methoden

Verschiedene Methoden, wie WhatsApp-Daten aus Android-Geräten ohne Rootzugriff und ohne physikalisches Backup extrahiert werden können, wurden in Kapitel 3 mit ihren jeweiligen Voraussetzungen, dem Vorgehen und dem Ergebnis vorgestellt. Dabei wurde deutlich, dass jede dieser Methoden gewisse Vor-, aber auch Nachteile aufweist, wobei jedoch immer der Zugriff auf das Gerät (z.B. bekannter Sperrcode) notwendig ist.
Die umfangreichste Extraktion kann durch ein komplettes ADB-Backup erfolgen, allerdings nur bei Geräten mit einer Android-Version vor 7.0. Bei allen anderen Methoden kann jeweils nur eine Auswahl der Daten gewonnen werden. Die versandten Nachrichten können auf alle genannten Weisen ausgelesen werden, entweder die aktuellste Version oder auch historische Bestände. Am einfachsten, aber auch am eingeschränktesten, ist dies durch die WhatsApp-interne Exportfunktion möglich.

Des Weiteren wurde erläutert, wie Logdateien in Bezug auf WhatsApp analysiert wurden und wie das entwickelte Python-Programm „WhatsApp Timeline Extractor“ zur Automatisierung von Zeitleistenextraktionen funktioniert. Dabei wurde deutlich, dass neben dem einfachen Filtern relevanter Logeinträge und Extrahieren zusätzlicher Informationen innerhalb des Programms bei beiden Logdateien ein auftretendes Problem behandelt werden muss.
In der WhatsApp-Logdatei handelt es sich dabei um die Einträge von `"DeleteMes-`

sage" -Ereignissen, wobei Statusmeldungen, welche automatisch (ohne Zutun des Nutzers) gelöscht werden, die gleichen Inhalte generieren wie manuell gelöschte Nachrichten. Da aus forensischer Sicht nur das aktive Verhalten des Nutzers relevant ist, soll das Löschen von Status aber möglichst nicht in der extrahierten Zeitleiste enthalten sein. Wie in Abschnitt 3.2.2.3, Seite 47 erläutert, wird das so gut wie möglich realisiert. Die beiden eingesetzten Methoden (programmatisch durch Zwischenspeichern empfangener Status-IDs sowie mit Hilfe eines weiteren Logeintrags) ergänzen sich dabei sehr gut, eine falsche Einordnung kann allerdings nicht ausgeschlossen werden.
In Bezug auf das Logcat wurde deutlich, dass die meisten Menschen die einzelnen Anwendungen (Apps) auf einem Gerät unterscheiden, im Android-Log allerdings einzelne Aktivitäten differenziert betrachtet werden. Um diese Diskrepanz zu überwinden, wurde eine Möglichkeit gefunden, durch Buchführung über laufende und aktive Anwendungen aus den Logeinträgen verständliche App-Ereignisse zu generieren.

Außer Acht gelassen wurde die Möglichkeit, das Android-Gerät mit Hilfe einer Sprachsteuerung (z.B. dem Android-internen „Okay, Google“) zu bedienen. In diesem Fall kann zumindest nicht mehr von einer aktiven Bedienung ausgegangen werden, wie sie z.B. während des Autofahrens verboten ist. Allerdings ist nicht klar, inwieweit die Sprachsteuerung neben den Funktionen auf Systemebene (z.B. Starten einer Anwendung) auch Drittanbieter-Apps wie WhatsApp steuern kann, ob das Versenden einer Nachricht mit diesem Messenger also ohne das Berühren des Displays möglich ist.
Eventuell könnte in solch einem Fall allerdings zusätzlich die Logdatei des Sprachassistenten (so sie existiert) untersucht werden.

5.1.3 Ergebnisse

In Kapitel 4 wurden zuerst beispielhaft Ergebnisse angeführt, die durch die zuvor beschriebenen Extraktionsmethoden gewonnen werden können und die verschiedenen Vor- und Nachteile dieser verglichen.

Anschließend wurden die Ergebnisse der Logfile-Analyse von `whatsapp.log` und Android Logcat aufgeführt. Es ist zu beobachten, dass bei vielen Logeinträgen zusätzliche Informationen zu dem zugrunde liegenden Ereignis vorhanden sind. So wird z.B. beim Öffnen eines Chats die JID des Kontaktes mitgeloggt oder beim Nachricht senden deren ID.
Mitunter sind solche Informationen allerdings auch nicht direkt in dem entsprechenden Eintrag enthalten, sondern können nur durch Verknüpfung von mehreren Zeilen extrahiert werden. Beispiele dafür sind der Kontakt, in dessen Chat eine Nachricht

gelöscht wurde, oder die JID eines angerufenen Kommunikationspartners.
Grundsätzlich sind die Ereignisse, die in der WhatsApp- bzw. der Android-Logdatei mitgeschrieben werden, verschieden. Eine Ausnahme dazu stellt das Ein- und Ausschalten des Telefons dar, wozu sich in beiden Logs Einträge befinden.
Beachtet werden muss die Möglichkeit, dass verschiedene Hersteller eigene Funktionen in ihre Mobiltelefone einbauen, die zu einzelnen veränderten Einträgen im Android-Logcat führen können. Dadurch, oder durch Systemaktualisierungen, könnten die gefundenen Logeinträge zu einzelnen Aktivitäten aus Tabelle B.5, Anhang B ihre Gültigkeit verlieren und eine erneute Analyse müsste durchgeführt werden. Da WhatsApp nur von einem Unternehmen angeboten wird, ist das Risiko der Veränderung für `whatsapp.log`-Dateien geringer, aufgrund von Softwareupdates ist es jedoch auch in diesem Fall nicht auszuschließen.

Die Ergebnisse eines Tests über die Genauigkeit des WhatsApp Timeline Extractors belegen, dass mit Hilfe der gefundenen Einträge die Interaktion des Nutzers mit dem Gerät allgemein und in der WhatsApp-Anwendung im Speziellen sehr gut nachvollzogen werden kann.

Zusammenfassend kann also gesagt werden, dass das Ziel dieser Bachelorarbeit erreicht wurde. Die forensisch relevanten Grundlagen der Messengeranwendung WhatsApp auf dem mobilen Betriebssystem Android wurden erläutert und vier verschiedene Methoden, mit dem IM-Dienst in Verbindung stehende Daten auch ohne physikalisches Image zu extrahieren, wurden aufgeführt.
Auch die Analyse von zwei Logdateien in Bezug auf WhatsApp konnte für 35 unterschiedliche Ereignisse (Tabellen B.4 und B.5 im Anhang B) erfolgreich durchgeführt werden. Die gewonnenen Erkenntnisse wurden zudem in einem entwickelten Programm zur Automation der Ereignisextraktion und Aufbereitung der Ergebnisse zur späteren Auswertung umgesetzt.
Sowohl die Informationen und Methoden aus dieser Arbeit als auch der „WhatsApp Timeline Extractor“ können demnach in zukünftigen digitalforensischen Untersuchungen zum Einsatz kommen und zu neuen Erkenntnissen führen.

5.2 Vergleich mit anderen Studien

Die folgenden Artikel stehen in Bezug zum Thema dieser Arbeit und sollen in diesem Abschnitt vergleichend herangezogen werden:

1. Mansur Zakariyya Shuaibu und Alhassan Bala: „WhatsApp Forensics and its Challenges for Android Smartphones“. 2016. [SB16]
2. John K. Alhassan et al.: „Forensic Acquisition of Data from a Crypt 12 Encrypted Database of Whatsapp“. 2017. [Alh+17]
3. Neha S. Thakur: „Forensic Analysis of WhatsApp on Android Smartphones“. 2013. [Tha13]
4. Gandeva Bayu Satrya et al.: „Android Forensics Analysis: Private Chat on Social Messenger“. 2016. [SDS16]

5.2.1 Shuaibu und Bala

Shuaibu und Bala stellen in ihrer Arbeit WhatsApp aus Forensik- bzw. Sicherheitsperspektive vor und erläutern eine Methode, um die verschlüsselten Nachrichtenbackups öffnen zu können. Die Auswertung oder Extraktion von weiteren Daten des Messengers wird dagegen nicht thematisiert.
Nachdem WhatsApp allgemein und die (zum Zeitpunkt der Veröffentlichung) aktuelle Marktsituation erläutert wird, werden technische Details angeführt. Zum Beispiel werden die beim Einschalten des Telefons automatisch gestarteten WhatsApp-Dienste („ExternalMediaManage“ und „MessageService“) genannt.
Es erfolgt eine Unterscheidung zwischen der aktuellen und früheren Versionen der Anwendung („Now and Before“). Deutlich wird das anhand der Nachrichtendatenbank `msgstore.db`, die ursprünglich unverschlüsselt im zugänglichen Bereich des Telefons gespeichert und nach und nach mit immer stärkerer Verschlüsselung geschützt wurde. Sichtbar wird das an der Dateiendung des Nachrichtenbackups, die von `.crypt` über `.crypt5` und `.crypt7` zu `.crypt8` verändert wurde.
Wie im Abschnitt 2.2.4.14 erläutert, wurde die Verschlüsselung seitdem nochmals verändert, was sich in der aktuellen Endung `.crypt12` widerspiegelt.
Nachdem die Anfänge der Verschlüsselung erläutert wurden, erwähnen Shuaibu und Bala auch die neue Ende-zu-Ende-Verschlüsselung, die WhatsApp seit 2016 einsetzt (siehe auch Abschnitt 2.2.3).

Zuletzt wird eine Methode zum Entschlüsseln des Nachrichtenbackups in der `.crypt8`-Version aufgezeigt. Zusammengefasst sind die folgenden Schritte notwendig:

1. Installation der App „WhatsApp-Crypt-DB Converter“ (im Google Play Store nicht mehr verfügbar) auf dem auszulesenden Telefon
2. Umwandeln der `.crypt8`-Dateien in `.crypt`[1]-Dateien mit Hilfe dieser Anwendung
3. Kopieren der umgewandelten Datenbanken auf einen Computer
4. Entschlüsselung der Dateien mit Hilfe des Python-Programms „WhatsApp Xtract Tool“ (verfügbar unter [Pic14])

Anhand der Aktualisierung der Verschlüsselung seit der Veröffentlichung dieses Artikels wird deutlich, dass Softwareanwendungen allgemein, insbesondere aber mobile Applikationen sehr häufigen Änderungen unterliegen. Die `.crypt8`-Variante, auf die sich Shuaibu und Bala bezogen haben, wird in der aktuellen Version nicht mehr unterstützt, weshalb so gut wie keine Geräte mehr mit solchen Dateien existieren und diese Methode nicht mehr angewandt werden kann[2].
Genauso ist nicht sicher, wie lange die derzeit (zum Zeitpunkt der Abgabe dieser Bachelorarbeit) genutzte Verschlüsselung `.crypt12` noch im Einsatz sein wird. Vor allem die Informationen über den Aufbau der `key`- und der `msgstore.db.crypt12`-Datei aus den Abschnitten 2.2.4.6 und 2.2.4.14 könnten dann ihre Gültigkeit verlieren.

5.2.2 Alhassan et al.

Auch Alhassan et al. konzentrieren sich in ihrer Arbeit für die „2nd International Engineering Conference“ in Nigeria auf die von WhatsApp verwendete verschlüsselte Nachrichtendatenbank.
Nach einer ähnlich wie in dem zuvor erwähnten Artikel aufgebauten Einleitung über die App und technische Details des Messengers folgt eine Methode zur Entschlüsselung von solchen Nachrichtenbackups. Im Gegensatz zu Shuaibu und Bala bezieht sich diese Arbeit allerdings auf die aktuell eingesetzte Verschlüsselungsvariante `.crypt12` und weist demnach auch zum Zeitpunkt der Abgabe dieser Bachelorarbeit noch Re-

[1] In der ersten Variante des `crypt`-Formats war die Datenbank immer mit dem gleichen 192 Bit großen Schlüssel *346a23652a46392b4d73257c67317e352e3372482177652c* AES-verschlüsselt [CSW12].

[2] Sollte es allerdings gelingen, die aktuell vorhandene verschlüsselte Datenbank ebenfalls in das alte `.crypt`-Format zu überführen, könnten die letzten beiden Schritte von Shuaibu und Bala angewandt werden, um Zugriff auf die Nachrichten zu erhalten.

levanz auf.

Zum Umgang mit dieser neusten Version der Datenbankverschlüsselung wird die Android-App „Omni-Crypt“ genutzt, die im Google Play Store nicht mehr verfügbar ist, aber von anderen Quellen als APK-Datei heruntergeladen werden kann, z.B. [Eli16a]. Diese Anwendung wurde (wie auch das Shellskript zum App-Backup über ADB, das in Abschnitt 3.1.3 genutzt wird) von dem GitHub-Nutzer „Elite-AndroidApps“ geschrieben und hat eine Funktion namens „Decrypt WhatsApp Database“, die die jeweils aktuelle `msgstore.db.crypt12`-Datenbank in das unsichere `.crypt`-Format (siehe Fußnote 1) umwandelt und in das zugängliche Verzeichnis `/sdcard/WhatsApp/Databases` kopiert. Allerdings ist nicht dokumentiert, wie diese Umwandlung erfolgt.
Die so konvertierte Datei kann dann schließlich am Computer entschlüsselt werden, wobei für diesen Zweck auch bei Alhassan et al. das „WhatsApp Xtract Tool“ [Pic14] zum Einsatz kommt.

Die beschriebene Methode kann wie folgt zusammengefasst werden:

1. Installieren der Omni-Crypt-App auf dem auszulesenden Gerät
2. Konvertieren der aktuellen `.crypt12`-Datenbank nach `.crypt` mit Hilfe von Omni-Crypt
3. Kopieren der resultierenden Datei auf einen Auswerterechner
4. Entschlüsseln der `.crypt`-Datei mit Hilfe des WhatsApp Xtract Tools
5. Visualisieren des Datenbankinhalts mit dem Programm „DB Browser for SQLite“ [Pia+18]

Zum Schluss der Arbeit fassen die Autoren zusammen, dass die Entschlüsselung mit der beschriebenen Methode erfolgreich durchgeführt wurde und weisen darauf hin, dass es sehr wichtig ist, immer auf dem aktuellen Stand zu bleiben, was durch die schnelle Entwicklung der WhatsApp-Datenbankverschlüsselung veranschaulicht wird.

Diese Methode unter Nutzung der Omni-Crypt-App ist sehr einfach anzuwenden und kann als Alternative zu den in Abschnitt 3.1 beschriebenen gesehen werden, wobei beachtet werden muss, dass sie nicht immer (z.B. bei einigen Testgeräten mit Android 7.1 und 8.0) zum Erfolg führt. Funktioniert sie, ist es durch Umbenennen eines beliebigen Datenbank-Backups mit dem Namen `msgstore-yyyy-mm-dd.1.db.crypt12` zu `msgstore.db.crypt12` sogar möglich, auch ältere Versionen des Nachrichtenbestandes (mit eventuell später gelöschten Inhalten) zu extrahieren.

Analog zu den Extraktionsmethoden aus dieser Bachelorarbeit werden in Tabelle 5.1 die Vor- und Nachteile der Methode von Alhassan et al. aufgeführt.

Vorteile	Nachteile
• kompletter Nachrichtenbestand (aus allen Chats) inkl. Anrufliste zugänglich • Wiederherstellung von älteren Versionen der Datenbank möglich	• nur Zugriff auf Nachrichtenverlauf, keine weiteren Daten • keine Kontaktinformationen enthalten (nur entschlüsselte `msgstore.db` vorhanden) • Veränderung von $\mathcal{T}_{WA}$ (Installation einer neuen Anwendung) • nicht bei allen Android-Versionen anwendbar*

* Erfolgreich getestet wurde die Methode z.B. auf Android 5.1 und 6.0, ohne Erfolg auf Android 7.1 und 8.0 (siehe auch vorheriger Abschnitt).

Tabelle 5.1: Vor- und Nachteile der Extraktionsmethode von [Alh+17] unter Nutzung von Omni-Crypt

Quelle: eigene Arbeit

5.2.3 Thakur

In ihrer Arbeit für den Master of Science-Abschluss führt Frau Thakur mehrere Möglichkeiten an, WhatsApp-Daten aus Android-Geräten zu extrahieren. Dabei nimmt sie hauptsächlich auf Nachrichten und Kontakte Bezug. Eine Besonderheit dabei ist das Einbeziehen der flüchtigen Daten im Arbeitsspeicher (*RAM*).
Nachdem in einigen Details erläutert wird, wie WhatsApp funktioniert (Aufbau der JIDs, Verwendung von XMPP zum Versand von Nachrichten), führt sie mit Hilfe von „Related Work“ den (zum Zeitpunkt der Veröffentlichung) aktuellen Stand der WhatsApp-Forensik an. Dabei wird deutlich, dass noch die erste Version der Nachrichtenbackup-Verschlüsselung zum Einsatz kommt (siehe auch Fußnote 1 auf Seite 82).

Es folgt eine kurze Einführung über Anwendungen und die Speicherverwaltung auf Android-Geräten. Wichtig ist dabei, dass jede Applikation (die in der Programmiersprache Java geschrieben wurde) in einer separaten „security sandbox“ [Tha13] läuft. Das bedeutet, dass jede App unter ihrem eigenen Benutzer gestartet wird und demzufolge eigene Rechte und einen eigenen Datenordner im Verzeichnis `/data/data` besitzt. Auf diese Weise ist es nur der entsprechenden Anwendung (und dem Systembenutzer *root*) möglich, auf ihre dort gespeicherten Daten zuzugreifen.

Des Weiteren wird erwähnt, dass ausführbare sowie temporäre Daten laufender Programme im RAM des Gerätes gespeichert werden und dort so lange erhalten bleiben, bis entweder die Stromzufuhr unterbrochen wird (z.B. durch Ausschalten des Telefons) oder das Android-System den Speicherplatz für eine andere Anwendung mit höherer Priorität benötigt (wobei IM-Anwendungen wie WhatsApp oft hohe Prioritäten haben). Dementsprechend können temporäre Daten also auch nach Beenden der Anwendung noch im Arbeitsspeicher vorhanden sein, wenn der Platz noch nicht neu vergeben wurde.
Diese Eigenschaft nutzt Frau Thakur im weiteren Verlauf ihrer Arbeit aus, um gelöschte WhatsApp-Nachrichten aus dem RAM wiederherzustellen.

Zuerst werden Methoden erläutert, die sich mit der Gewinnung von WhatsApp-Daten aus nicht-flüchtigem Speicher beschäftigen, wobei zum Zeitpunkt der Veröffentlichung noch die alte `.crypt`-Version der Verschlüsselung aktuell war. Da diese sehr einfach zu knacken ist, geht die Arbeit ähnlich vor wie die beiden zuvor genannten und verwendet zur Entschlüsselung das „WhatsApp Xtract Tool“ von Francesco Picasso [Pic14]. Außerdem wird erwähnt, dass bei einem gerooteten Gerät (bei dem der Nutzer alle Rechte des Systemverwalters *root* hat), einfach auf die unverschlüsselte Nachrichtendatenbank sowie weitere Dateien, z.B. `wa.db` unter `/data/data/com.whatsapp/databases` zugegriffen werden kann. Auch auf einige weitere Dateien im WhatsApp-Datenverzeichnis wird eingegangen, allerdings weniger umfangreich als in Abschnitt 2.2.4.

Das Hauptaugenmerk dieser Arbeit liegt jedoch auf der Datengewinnung aus dem Arbeitsspeicher (wobei ein Root-Zugang vorausgesetzt wird). Zu diesem Zweck wird die Anwendung „memfetch“ [lca07] eingesetzt, die den gesamten RAM des Gerätes lesen kann und den entsprechenden Bereich einer einzelnen Anwendung in Binärdateien abspeichert.
Nachdem das ausführbare Programm auf das Android-System kopiert wurde, muss zuerst die Prozess-ID von WhatsApp ermittelt werden, damit memfetch anhand dieser den korrekten Speicherbereich auslesen kann. Die fertiggestellte Datei kann dann z.B. über ADB für weitere Untersuchungen auf einen Rechner kopiert werden.
Die wichtigsten der dazu notwendigen ADB-Kommandos werden im Folgenden aufgeführt (nach [Tha13]).

```
1 adb push memfetch /sdcard/memfetch # Anwendung auf das ↩
   ↪ Telefon kopieren
2 adb shell
3 cd /sdcard/memfetch
```

```
4  su # zum Erlangen von Rootrechten
5  ps # zum Ermitteln der Prozess-ID (PID) von WhatsApp
6  ./memfetch <pid> # Auslesen des korrekten ↩
    ↪ Speicherabschnittes
7  exit # Verlassen der ADB-Shell
8  adb pull /sdcard/mem-002.bin
```

Listing 5.1: ADB-Kommandos zum Einsatz von memfetch

Um die so extrahierten Binärdaten auszuwerten, wurde im Rahmen der Arbeit das Shellskript „whatsappRamXtract“ (für die Linux-Bash) entwickelt, welches mit Hilfe von Carving Telefonnummern, Nachrichten und SQL-Anfragen extrahiert. Außerdem werden unter Einsatz von „Scalpel“ Bilddateien aus dem Binärdump gecarvt.

Zum Schluss ihrer Arbeit weist Frau Thakur darauf hin, dass WhatsApp-Nachrichten beim Löschen zwar aus der Datenbank entfernt werden, nicht aber sofort aus dem Arbeitsspeicher. Das bedeutet, dass mit Hilfe des beschriebenen RAM-Abbildes und des entwickelten Programms auch gelöschte Nachrichten noch wiederhergestellt werden können.
Was Thakur nicht erwähnt ist, dass durch den Einsatz von sog. *Write-Ahead-Logs* (kurz WAL) bei den SQLite-Datenbanken `msgstore.db`, `wa.db` etc. Änderungen nicht sofort in die Datenbankdatei übernommen, sondern zunächst nur in die entsprechende WAL-Datei (z.B. `msgstore.db-wal`) geschrieben werden. Erst nach einem kompletten Commit wird die Datei aktualisiert.
So sind die neusten Nachrichten möglicherweise in der `msgstore.db` noch gar nicht enthalten, vor Kurzem gelöschte Mitteilungen können dafür aber auch noch existieren.

Da sie sich vor Allem auf die Extraktion von Daten aus dem Arbeitsspeicher konzentriert, sind außerdem keine weiteren Methoden enthalten, mit denen WhatsApp-Dateien gewonnen werden können, wie sie in dieser Bachelorarbeit thematisiert werden. Allerdings kann die Einbeziehung des Arbeitsspeichers in forensische Analysen wichtige zusätzliche Informationen liefern, die wiederum in dieser Arbeit außer Acht gelassen werden.

5.2.4 Satrya et al.

Im Gegensatz zu den drei zuvor genannten Arbeiten beschäftigen sich die Autoren dieser Veröffentlichung nicht mit WhatsApp, sondern mit anderen IM-Anwendungen auf Android-Geräten. Dabei werden die Apps „Telegram“, „Line“ und „KakaoTalk“

untersucht, wobei der Fokus vorrangig auf den Versand von Textnachrichten gelegt wurde. Bei diesen drei Messengern gibt es jeweils neben normalen Chats sog. „Hidden“ oder „Secret Chats“, die einen sichereren Nachrichtenaustausch ermöglichen sollen.
In der Arbeit wurden die Auswirkungen von IM-Kommunikation sowohl mittels einer „Post-mortem“-Analyse [SDS16] der veränderten Dateien auf dem Telefon als auch durch Einbeziehen von Einträgen in Logdateien analysiert.

Nachdem zuerst Testdaten generiert wurden, indem, ähnlich zu den Methoden in dieser Bachelorarbeit, zwischen zwei Android-Telefonen[3] Nachrichten ausgetauscht wurden, erfolgte eine Identifizierung der dadurch veränderten Daten. Dabei wurden vor und nach den ausgeführten Aktionen jeweils die Hashwerte in Betracht kommender Dateien berechnet und verglichen.
Für weitere Hinweise auf den Versand von Nachrichten mit den entsprechenden Anwendungen wurde das Android-Systemlog „Logcat“ ausgewertet, welches in dieser Arbeit im Abschnitt 3.2 in Bezug auf WhatsApp ebenfalls zum Einsatz kommt. Dabei wurden jeweils zu den Ereignissen gehörende Logeinträge gesucht.

Satrya et al. kommen zu dem Ergebnis, dass alle drei untersuchten Messenger-Apps die Nachrichten in ihrem eigenen Datenverzeichnis in einer SQLite-Datenbank auf dem Telefon speichern, z.B. `/data/data/org.telegram.messenger/files/cache4.db` für Telegram. Bei diesem Messenger gibt es zwar in der Datenbank einen Hinweis auf die verwendete Chatfunktion, die Nachrichteninhalte werden allerdings immer im Klartext gespeichert.
Ähnlich verhält es sich bei Line, wobei in diesem Fall die Schwierigkeit hinzukommt, dass im „Hidden Chat“ versandte Mitteilungen nach einer eingestellten Zeit (maximal eine Woche) automatisch gelöscht werden und danach nicht mehr zugänglich sind.
Grundsätzlich werden die Nachrichteninhalte in der Datenbank von KakaoTalk verschlüsselt, die jeweils letzte Mitteilung jedes Chats findet man allerdings im Klartext in einer anderen Tabelle der Datenbank vor. Außerdem bleibt unklar, welches Verfahren zur Verschlüsselung zum Einsatz kommt.
Des Weiteren wurden für jede IM-Anwendung mit dem Nachrichtenversand bzw. -empfang korrelierende Einträge im Logcat gefunden.

[3] Bei Satrya et al. kommen zur Untersuchung zwei gerootete Android-Geräte zum Einsatz. Es besteht also Zugriff auf den geschützten Datenbereich unter `/data/data`.

5.2.5 Zusammenfassung

Beim Betrachten der vorgestellten Arbeiten wird deutlich, dass sich die meiste (veröffentlichte) Forschung in Bezug auf WhatsApp mit dem Entschlüsseln der Nachrichtenbackups beschäftigt. Oft kommt dabei nicht die aktuelle Version zum Einsatz, was allerdings den häufigen Aktualisierungen der Anwendung geschuldet ist.
Die Extraktion von weiteren Dateien der Anwendung wird dagegen nicht berücksichtigt. Auch ist keine Arbeit bekannt, die eine Analyse von Logdateien allgemein im Hinblick auf den Messenger WhatsApp anstellt und die `whatsapp.log`-Datei im Speziellen wurde ebenfalls noch nicht genauer untersucht. Lediglich in Bezug auf andere IM-Dienste wurde das Android-Systemlog untersucht [SDS16], wenn auch nur für wenige Ereignisse.

5.3 Ausblick

Wie schon im Vergleich mit den Studien von Shaibu und Bala bzw. Thakur deutlich wurde, gibt es im Bereich von Mobilgeräten, wie z.B. Smartphones, sowie den passenden Anwendungen (wie WhatsApp) eine sehr schnelle Entwicklung. Nicht nur technische Einzelheiten, z.B. die Verschlüsselung des Nachrichtenbackups, werden durch den ständigen Wettlauf zwischen Sicherheitsangriffen und -verbesserungen verändert. Auch werden immer wieder zusätzliche Funktionen eingeführt, die sowohl zu neuen Möglichkeiten als auch zu Einschränkungen in der Forensik führen können, wie z.B. die kommende WhatsApp-Funktion zum Versenden von Finanzmitteln, die zur Zeit in Indien getestet wird (mehr dazu im Abschnitt 2.1).
Es muss also berücksichtigt werden, dass die vorliegende Arbeit nur eine Momentaufnahme der WhatsApp-Forensik zum Abgabezeitpunkt darstellt und einzelne Informationen mit der Zeit an Gültigkeit verlieren können. Es ist, gerade in der digitalen Forensik, wichtig, immer auf dem neusten Stand zu bleiben und sich fortlaufend über Neuerungen zu informieren.

In dieser Arbeit wurden verschiedene Möglichkeiten der Forensik in Bezug auf die Messengeranwendung WhatsApp auf dem Betriebssystem Android diskutiert, da dieses laut [IDC17] mit ca. 85% Marktanteil (Stand: 2017) am weitesten verbreitet ist. Allerdings könnte (und sollte) die forensische Analyse von WhatsApp in Zukunft auch auf weitere existierende Systeme ausgedehnt werden, wie z.B. das Apple iOS, das mit 14,7% ([IDC17], Stand: 2017) an zweiter Stelle kommt.
Außerdem könnte die Analyse von Logeinträgen noch auf weitere, nicht in den Tabellen B.4 und B.5 (Anhang B) enthaltene Aktivitäten ausgeweitet werden, um eine noch detailliertere Zeitleiste erstellen zu können. Denkbar wäre z.B. das Archivieren von WhatsApp-Chats oder das Starten eines Sprachsteuerungs-Assistenten (siehe Abschnitt 5.1.2).

Anhang A: Abbildungen

	_id	browser_id	secret	token	os
		Filtern	Filtern	Filtern	Filtern
1	6	0GmEQpuVYezj+/Op3/0twg==	hByiQbJFr...	MOzQAnQ...	Windows 8.1
2	8	SevTLUiGZ0qVVEqn2ydAzw==	rX15WXI...	r8w8DY1ni...	Mac OS 10.12.6
3	9	+axqz9J4LfZZQzM/kG5gTg==	lqbMwXz...	tJ5YuflJsW...	Linux x86_64

[...]

browser_type	lat	lon	accuracy	place_name	last_active	timeout	expiration	fservice
Filtern			Filt...	Filtern	Filtern	Fi...	Filt...	F...
Firefox	NULL	NULL	NULL	NULL	1526547194000	0	NULL	0
Safari	NULL	NULL	NULL	NULL	1526634406000	0	NULL	0
Chrome	NULL	NULL	NULL	NULL	1526634447000	0	NULL	0

[...]

Abbildung A.1: Beispiel der Tabelle `sessions` in `web_sessions.db`
Quelle: eigene Arbeit

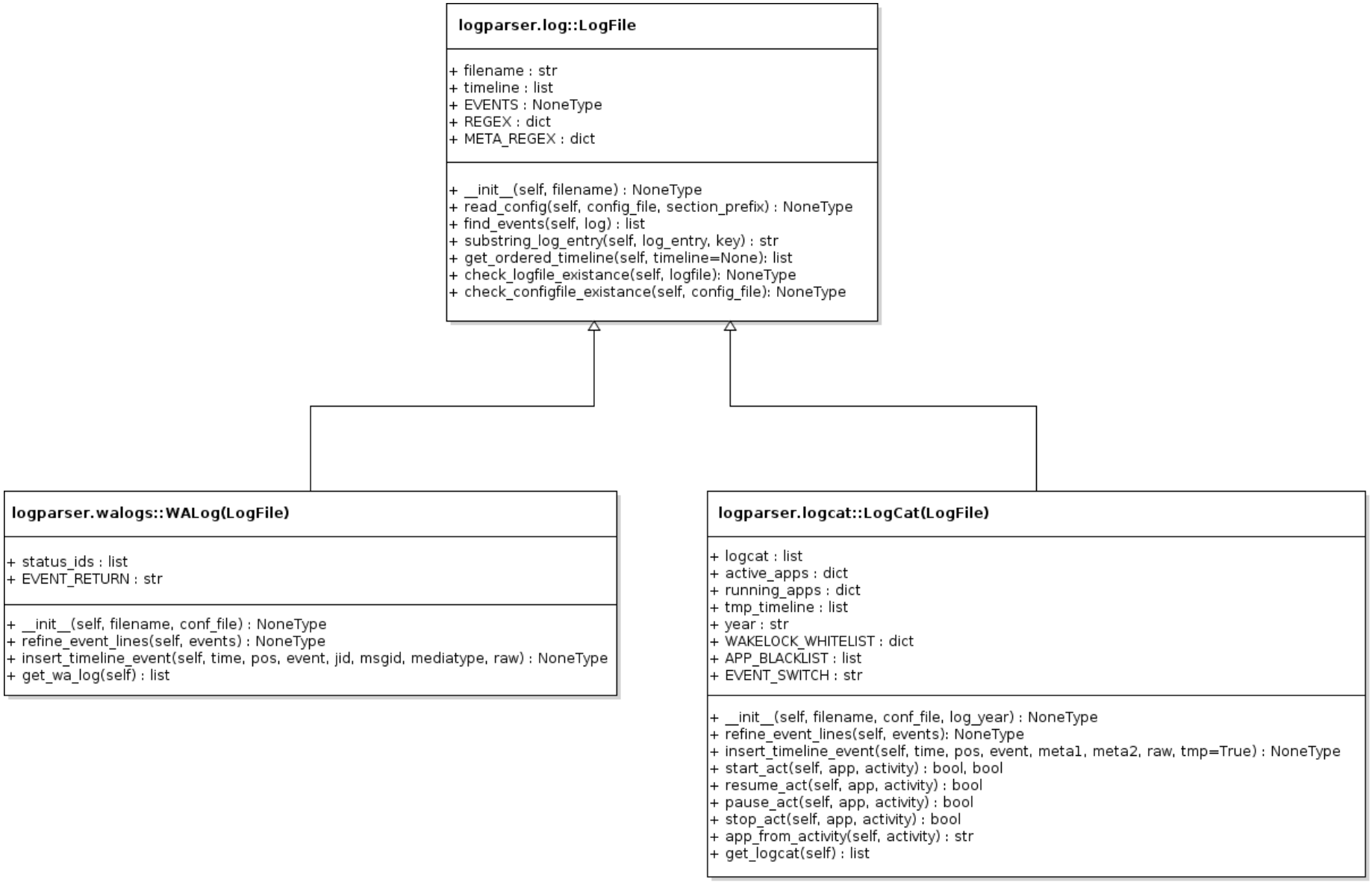


Abbildung A.2: UML-Diagramm der `LogFile`-, `WALog`- und `LogCat`-Klasse
Quelle: eigene Arbeit

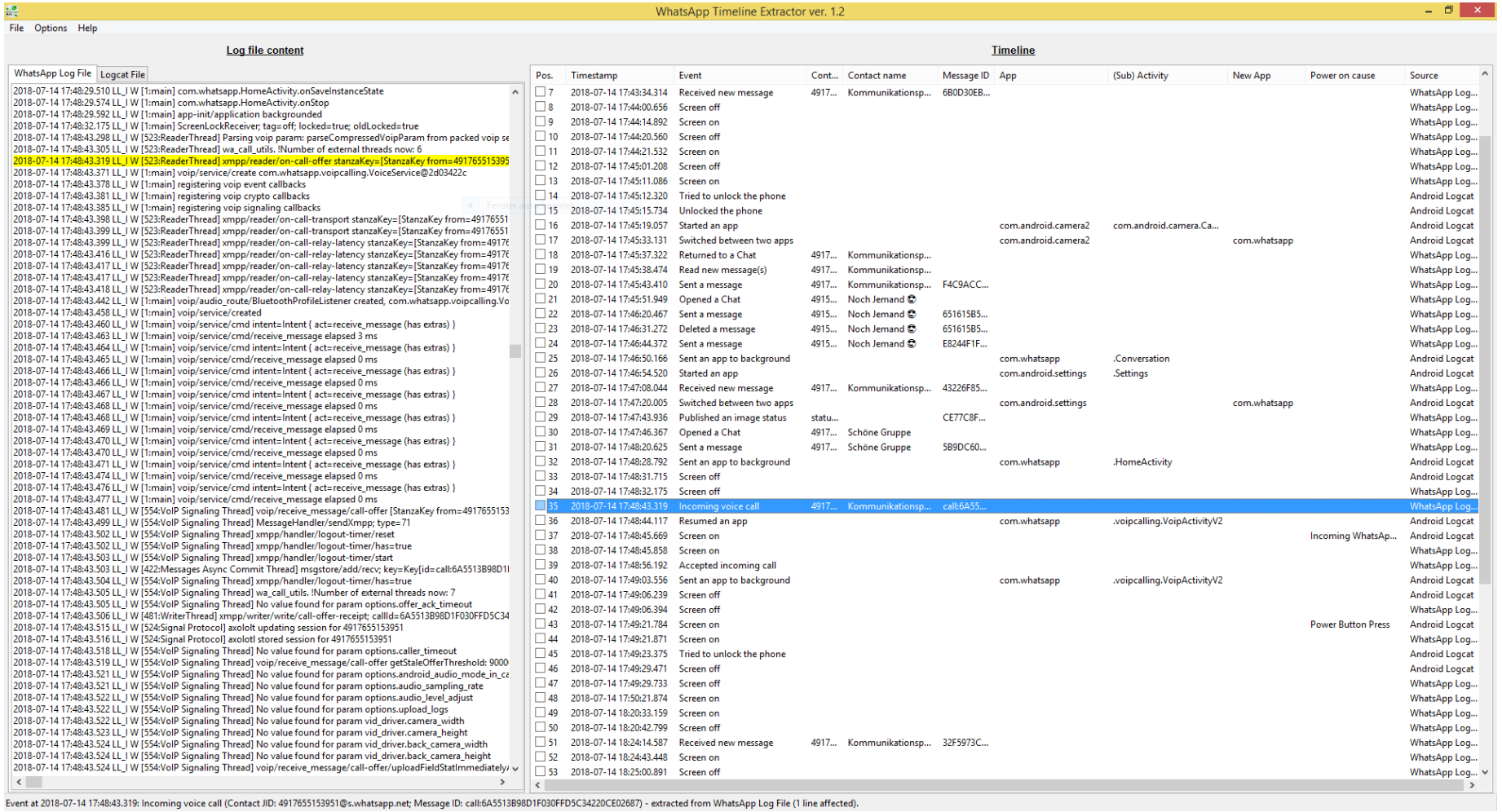


Abbildung A.3: Bildschirmfoto nach erfolgter Extraktion (WATE auf Windows)
Quelle: eigene Arbeit

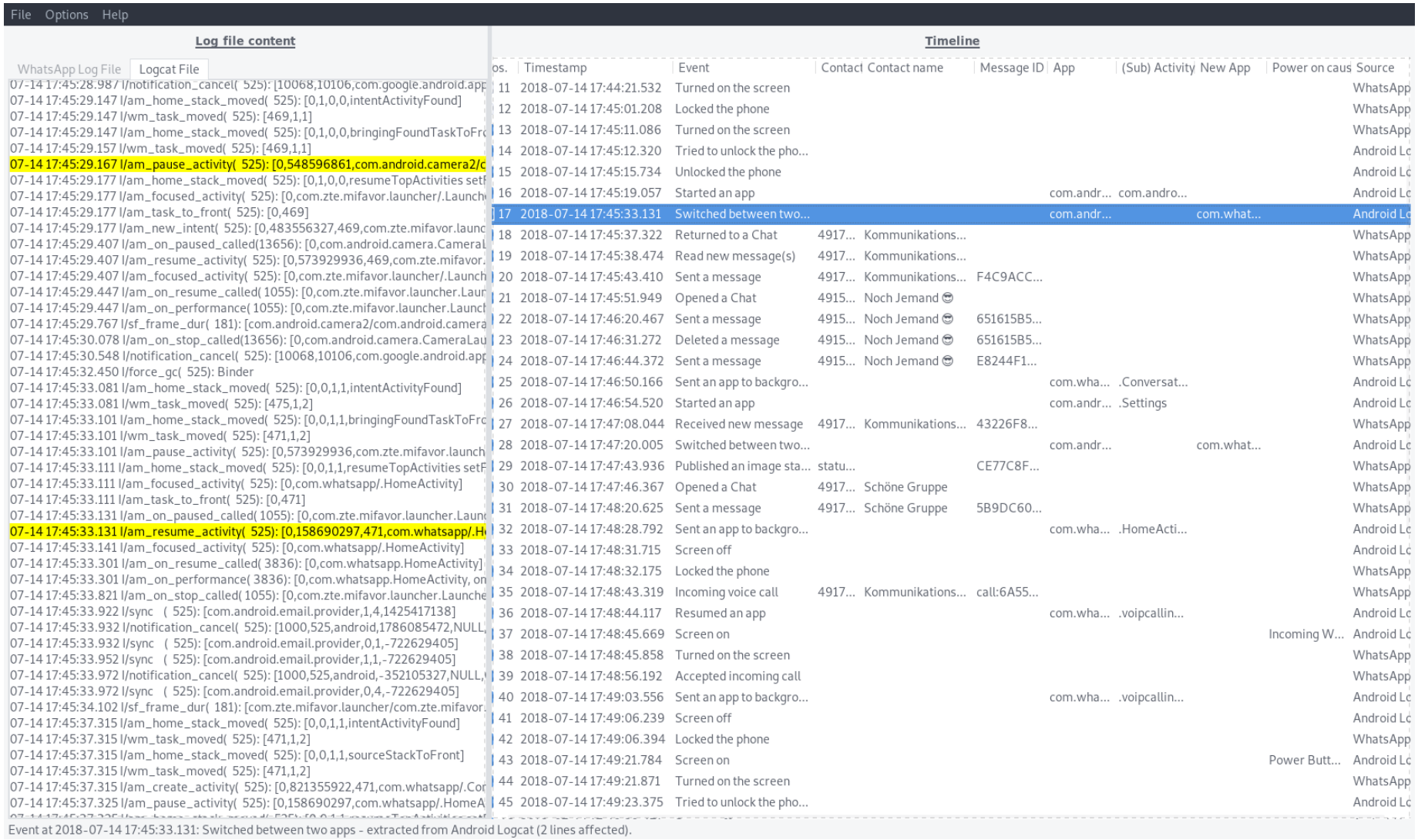


Abbildung A.4: Bildschirmfoto nach erfolgter Extraktion (WATE auf Linux)
Quelle: eigene Arbeit

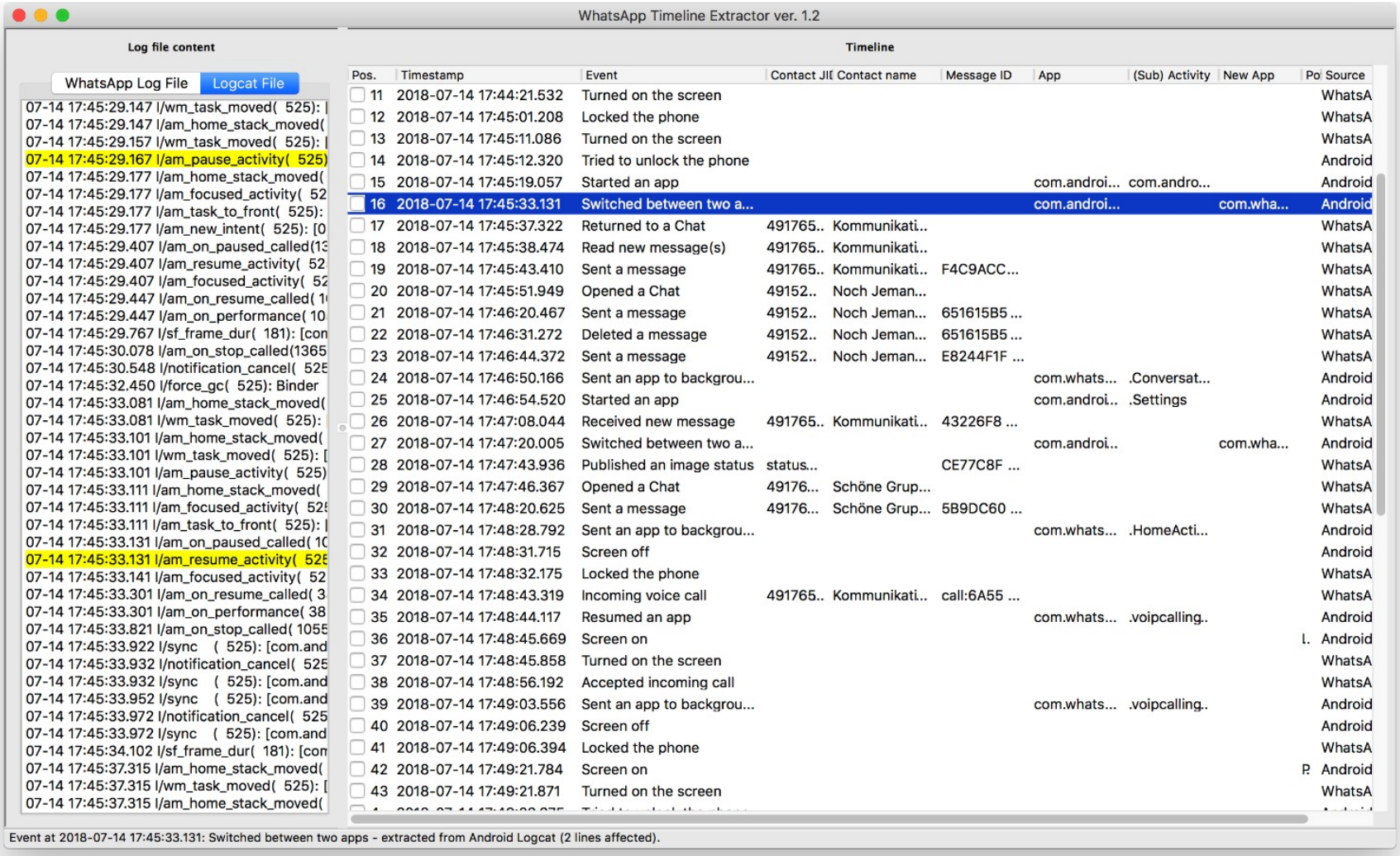


Abbildung A.5: Bildschirmfoto nach erfolgter Extraktion (WATE auf Mac OS X)
Quelle: eigene Arbeit

Anhang B: Tabellen

Offset	Länge	Inhalt
0x00	0x1B	Java-Serialisierungsdaten-Header
0x1B	0x2	Schlüssel-Header Präambel (`0x00 0x01`)
0x1D	0x1	Schlüsselversion (aktuell `0x02`)
0x1E	0x20	Schlüssel-Salt
0x3E	0x10	Salt für Google-Nutzernamen
0x4E	0x20	Hash von Google-Nutzernamen und Salt
0x6E	0x10	Initialisierungsvektor für AES-GCM-Verschlüsselung (ausgenullt)
0x7E	0x20	256 Bit-AES-Schlüssel

Tabelle B.1: Struktur der Datei `key`
Quelle: [Dai+17]

Offset	Länge	Inhalt
0x00	0x2	Schlüssel-Header Präambel (`0x00 0x01`)
0x02	0x1	Schlüsselversion (aktuell `0x02`)
0x03	0x20	Schlüssel-Salt
0x23	0x10	Salt für Google-Nutzernamen
0x33	0x10	Initialisierungsvektor für AES-GCM-Verschlüsselung
0x43	l_{enc}	verschlüsselte Daten
l_{enc} + 0x43	0x10	MD5-Hash der verschlüsselten Datenbank
l_{enc} + 0x53	0x2	feste Zeichenkette „`--`“ (`0x2D 0x2D`)
l_{enc} + 0x55	0x2	letzte 2 Ziffern der Telefonnummer

Tabelle B.2: Struktur der Datei `msgstore.db.crypt12`
Quelle: [Dai+17]

Offset	Länge	Inhalt
0x0000	0x2F	Datei-Header (enthält die Zeichenfolge „`com.whatsapp.Statistics`“)
0x002F	0x1C4	Auflistung aller enthaltenen Informationen
0x01F3	0x8	letzter Reset[a]
0x01FB	0x8	Größe der empfangenen Google Drive-Backups (in Bytes)
0x0203	0x8	Größe der empfangenen Mediendaten (in Bytes)
0x020B	0x8	Anzahl der empfangenen Mediennachrichten
0x0213	0x8	Größe der empfangenen Service-Nachrichten (in Bytes)
0x021B	0x8	Offline-Delay[a]
0x0223	0x8	Anzahl der empfangenen Offline-Nachrichten[a]
0x022B	0x8	Anzahl der empfangenen Zahlungs-Nachrichten
0x0233	0x8	Download-Volumen der Daten, die mittels Roaming[b] übertragen wurden (in Bytes)
0x023B	0x8	Größe der empfangenen Statusmeldungen (in Bytes)
0x0243	0x8	Anzahl der empfangenen Statusmeldungen
0x024B	0x8	Anzahl der empfangenen Textnachrichten
0x0253	0x8	Größe der eingehenden Anrufe (in Bytes)
0x025B	0x8	Anzahl der eingehenden Anrufe
0x0263	0x8	Größe der gesendeten Google Drive-Backups (in Bytes)
0x026B	0x8	Größe der gesendeten Mediendaten (in Bytes)
0x0273	0x8	Anzahl der gesendeten Mediennachrichten
0x027B	0x8	Größe der gesendeten Service-Nachrichten (in Bytes)
0x0283	0x8	Anzahl der gesendeten Zahlungs-Nachrichten
0x028B	0x8	Upload-Volumen der Daten, die mittels Roaming[b] übertragen wurden (in Bytes)
0x0293	0x8	Größe der gesendeten Statusmeldungen (in Bytes)
0x029B	0x8	Anzahl der gesendeten Statusmeldungen
0x02A3	0x8	Anzahl der gesendeten Textnachrichten
0x02AB	0x8	Größe der ausgehenden Anrufe (in Bytes)
0x02B3	0x8	Anzahl der ausgehenden Anrufe

[a] Bedeutung dieser Eigenschaft unbekannt.
[b] Roaming bezeichnet die Nutzung eines Mobilfunknetzes außerhalb des eigenen Netzes, z.B. im Ausland.

Tabelle B.3: Struktur der Datei `statistics`
Quelle: eigene Arbeit

WhatsApp-Aktionen	• einen Chat aus dem Hauptbildschirm heraus öffnen
	• einen Chat mit neuen, ungelesenen Nachrichten öffnen
	• eine neue Textnachricht versenden
	• eine neue Mediennachricht (mit Bild oder Video) versenden
	• eine Textnachricht empfangen
	• eine Mediennachricht empfangen
	• eine Nachricht „für mich löschen“
	• eine Nachricht „für alle löschen“
	• einen Sprachanruf starten
	• einen Videoanruf starten
	• mittels Sprachanruf angerufen werden
	• mittels Videoanruf angerufen werden
	• einen eingehenden Sprachanruf annehmen
	• einen eingehenden Videoanruf annehmen
	• ein ausgehender Sprachanruf wird angenommen
	• ein ausgehender Videoanruf wird angenommen
	• einen neuen Textstatus veröffentlichen
	• einen neuen Fotostatus veröffentlichen
	• einen neuen Videostatus veröffentlichen
	• einen neuen GIF-Status veröffentlichen

Tabelle B.4: ausgeführte WhatsApp-Aktionen auf dem Testgerät
Quelle: eigene Arbeit

Android-Aktionen	• den Bildschirm durch Betätigen des Einschaltknopfes einschalten
	• Bildschirmaktivierung durch eingehenden Anruf
	• Bildschirmaktivierung durch eingehenden WhatsApp-Sprachanruf
	• Bildschirmaktivierung durch eingehenden WhatsApp-Videoanruf
	• Bildschirmaktivierung durch eingehende SMS
	• Bildschirmaktivierung durch abgelaufenen Android-Timer
	• Bildschirmaktivierung durch Wecker
	• den Bildschirm ausschalten
	• das Gerät durch Codeeingabe entsperren
	• den Entsperrcode falsch eingeben
	• eine App starten
	• eine App minimieren
	• eine App wieder in den Vordergrund holen
	• zwischen zwei Apps wechseln
	• eine App schließen

Tabelle B.5: ausgeführte Android-Aktionen auf dem Testgerät
Quelle: eigene Arbeit

Anhang C: Quellcode

```
1  import java.io.FileOutputStream;
2  import java.io.IOException;
3  import java.io.RandomAccessFile;
4  import java.security.GeneralSecurityException;
5  import java.util.zip.DataFormatException;
6  import java.util.zip.Inflater;
7
8  import javax.crypto.Cipher;
9  import javax.crypto.spec.GCMParameterSpec;
10 import javax.crypto.spec.SecretKeySpec;
11
12 public class WhatsAppBackupDecryptor
13 {
14   // Hier entsprechende Pfade eintragen:
15   public static String keyFile = "";
16   public static String cryptFile = "";
17   public static String plainFile = "";
18
19   public static void main(String[] args)
20   {
21     RandomAccessFile keyStream = null, cryptStream = null;
22     FileOutputStream plainStream = null;
23     try
24     {
25       // In- und OutputStreams initialisieren
26       keyStream = new RandomAccessFile(keyFile, "r");
27       cryptStream = new RandomAccessFile(cryptFile, "r");
28       plainStream = new FileOutputStream(plainFile);
29
30       // Größe der verschlüsselten Daten bestimmen
31       // (ohne Header und Trailer)
32       int encDataSize = (int) (cryptStream.length() - 0x43 - ↩
   ↪ 0x14);
33       byte[] key = new byte[0x20];
```

```
34          byte[] iv = new byte[0x10];
35          byte[] encData = new byte[encDataSize];
36
37          // Einlesen der (benötigten Inhalte der) Dateien
38          keyStream.seek(0x7E); // Schlüssel (in key-Datei, ↩
    ↪ Offset 0x7E)
39          keyStream.read(key);
40          cryptStream.seek(0x33); // IV (in crpyt12-Datei, Offset↩
    ↪  0x33)
41          cryptStream.read(iv);
42          cryptStream.read(encData); // verschlüsselte Daten
43
44          // AES-GCM-Entschlüsselung
45          Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding")↩
    ↪ ;
46          SecretKeySpec keySpec = new SecretKeySpec(key, "AES");
47          GCMParameterSpec ivSpec = new GCMParameterSpec(128, iv)↩
    ↪ ;
48          cipher.init(Cipher.DECRYPT_MODE, keySpec, ivSpec);
49          // eigentliche Entschlüsselung:
50          byte[] compressedData = cipher.doFinal(encData);
51
52          // ZLIB-archivierte Daten entpacken & in Ausgabedatei ↩
    ↪ schreiben
53          Inflater decompressor = new Inflater(false);
54          decompressor.setInput(compressedData);
55          // 1KB Puffer, für größere Dateien sinnvoll
56          byte[] buffer = new byte[1024];
57          while(!decompressor.finished())
58          {
59            int count = decompressor.inflate(buffer);
60            plainStream.write(buffer, 0, count);
61          }
62
63       } catch (IOException | GeneralSecurityException | ↩
    ↪ DataFormatException e)
64       {
65          // Exceptions einfach ausgeben (z.B. bei ungültigem ↩
    ↪ Dateipfad)
66          e.printStackTrace();
67       } finally
```

```
68      {
69        // auf jeden Fall alle Streams schließen
70        try
71        {
72          if(keyStream != null)
73            keyStream.close();
74          if(cryptStream != null)
75            cryptStream.close();
76          if(plainStream != null)
77            plainStream.close();
78        } catch(IOException e)
79        {
80          e.printStackTrace();
81        }
82      }
83    }
84  }
```

Listing C.1: Quelltext der Datei `WhatsAppBackupDecryptor.java`

```
1  import re
2  import sys
3  from getopt import getopt, GetoptError
4
5  import os
6
7
8  def usage(nohelp=False):
9    # printing a usage hint
10   this_script = os.path.basename(__file__)
11
12   if not nohelp:
13     print("WhatsApp␣chat␣export␣parser␣{script}\n".format(↩
     ↪ script=this_script))
14
15   print("Usage:\n"
16         "-------------------------(PLEASE␣USE␣PYTHON␣3!!!)↩
     ↪ -------------------------\n"
17         "\n"
18         "python␣{script}␣[-h]␣-i␣<input␣txt␣file>␣[-o␣<output↩
     ↪ ␣html␣file>]␣[-m␣<my␣own␣WhatsApp␣name>]\n\n".format(↩
```

```
    ↪ script=this_script))
19    if not nohelp:
20      print("Try␣python␣{script}␣-h␣for␣more␣information.".↩
    ↪ format(script=this_script))
21
22
23  def help_msg():
24    # printing a help message
25    print("Welcome␣to␣wa_txt2html.py!\n"
26          "This␣is␣a␣simple␣script␣to␣parse␣WhatsApp␣chat␣↩
    ↪ exports␣from␣txt␣to␣html␣format␣for␣better␣examination.\↩
    ↪ n"
27          "\n")
28    usage(True)
29    print("Options:\n"
30          "-h␣Display␣this␣help␣message␣and␣exit."
31          "-i␣Specifies␣the␣input␣file␣name␣for␣WhatsApp␣chat␣↩
    ↪ export␣(.txt␣file).\n"
32          "-o␣Specifies␣the␣output␣file␣name␣(html␣file).\n"
33          "The␣default␣output␣name␣is␣WhatsApp_Chat.html␣(in␣↩
    ↪ the␣current␣directory).\n"
34          "-m␣Specifies␣from␣whose␣point␣of␣view␣the␣messages␣↩
    ↪ shall␣be␣displayed␣(Which␣chat␣partner␣is␣on␣the␣right␣↩
    ↪ side).\n"
35          "If␣not␣supplied,␣the␣author␣of␣the␣first␣message␣↩
    ↪ will␣be␣chosen␣for␣the␣point␣of␣view.")
36
37
38  def prepare_html(file, me, they, filename):
39    # writing the html preamble to the file
40    title = "WhatsApp␣Chat<br>between<br>{me}␣and␣{they}".↩
    ↪ format(me=me, they=they)
41
42    content = ["<!DOCTYPE␣HTML>",
43               "<meta␣charset=\"UTF-8\">",
44               "<head>",
45               "<title>WhatsApp␣Chat␣between␣{me}␣and␣{they}</↩
    ↪ title>".format(me=me, they=they),
46               "<style>",
47               "h1␣{",
48                 "text-align:␣center;",
```

```
49                  "font-family: sans-serif;}",
50              "h2 {"
51                      "text-align: center;"
52                      "font-family: sans-serif;}"
53              ".msg {",
54            "border: medium solid black;",
55                  "border-radius: 10px;",
56                  "font-family: sans-serif;",
57                  "display: inline-block;",
58                  "padding: 5px;",
59                  "word-break: break-all;",
60                  "max-width: 80vw;}",
61              ".center {",
62                  "text-align: center;}",
63              ".left {",
64                    "float: left;",
65                    "background-color: #c8ffc0;}",
66          ".right {",
67            "float: right;",
68            "background-color: #ddddff;}",
69          ".author {",
70            "float: left;",
71            "font-size: small;}",
72          ".right .content {",
73                "float: right;}",
74            ".left .content {",
75              "float: left;}",
76          ".time {",
77            "font-size: small;",
78            "float: right;}",
79          "</style>",
80              "</head>",
81              "<body>",
82              "<h1>{tit}</h1>".format(tit=title),
83              "<h2>({fil})</h2>".format(fil=filename),
84              "<table width=100%>"]
85
86    file.write("\n".join(content))
87
88
89  def write_html_msg(file, center, own, author, msg, time):
```

```
90    # writes all html code for one message
91    side = "right" if own else "left"
92    center_class = ""
93    if center:  # for system messages
94      side = ""
95      center_class = "center"
96
97    content = ["<tr>",
98               "<td␣class=\"{cls_center}\"><div␣class=\"msg␣{↩
      ↪ cls_side}\">".format(cls_center=center_class, cls_side=↩
      ↪ side),
99                "<table␣class=\"{cls}\">".format(cls=side),
100                    "<tr><td␣class=\"author\">",
101                    author,
102                    "</td></tr>",
103                  "<tr><td␣class=\"content\">",
104                  msg.replace("<", "&lt;").replace(">", "&gt;↩
      ↪ ").replace("\n", "<br>"),  # HTML char replacement
105                  "</td></tr>",
106                "<tr><td␣class=\"time\">",
107                time,
108                  "</td></tr>",
109                "</table>"
110                "</div></td>",
111                "</tr>"]
112
113   file.write("\n".join(content))
114
115
116 def finish_html(file):
117   # writes the html code to finish this file
118   content = ["</table>",
119          "</body>"]
120
121   file.write("\n".join(content))
122
123
124 if __name__ == '__main__':
125   if len(sys.argv) < 2 or len(sys.argv) > 7:
126     usage()
127     sys.exit(1)
```

```
128
129   try:  # parsing command line options
130     opts, args = getopt(sys.argv[1:], "hi:o:m:")
131   except GetoptError as e:
132     usage()
133     print("\n", e, "")
134     sys.exit(1)
135
136   infile = ""
137   outfile = ""
138   me = ""
139
140   for opt, arg in opts:  # processing command line options
141     if opt == "-h":
142       help_msg()
143       sys.exit(0)
144     elif opt == "-i":
145       if len(arg) == 0 or not os.path.exists(arg):
146         usage()
147         print("Invalid input file name:", arg)
148         sys.exit(1)
149       else:
150         infile = arg
151     elif opt == "-o":
152       if len(arg) == 0:
153         usage()
154         print("Invalid output file name:", arg)
155         sys.exit(1)
156       else:
157         outfile = arg
158     elif opt == "-m":
159       me = arg
160
161   if not len(outfile):
162     outfile = "WhatsApp_Chat.html"
163
164   # regex for all parts of a message (timestamp, author and ↩
      ↪ content):
165   regex = re.compile("(\d{2}\.\d{2}\.\d{2}, \d{1,2}:\d{2}) - ↩
      ↪ ((.*?): (.*)|.*)")
166
```

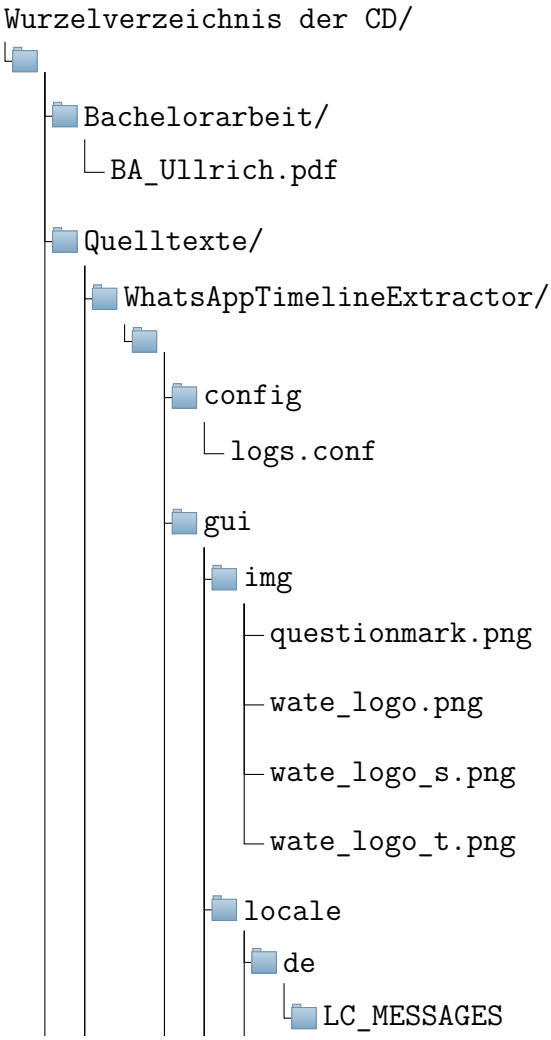
```
167   messages = list()
168
169   with open(infile, "r") as txt:
170     curr_time = ""
171     curr_author = ""
172     curr_msg = ""
173
174     for line in txt.readlines():
175       found = regex.search(line)
176       if found:
177         # save the last message
178         if len(curr_time):
179           messages.append({'time': curr_time,
180                            'author': curr_author,
181                            'msg': curr_msg.strip()})
182
183         curr_time = found.group(1)
184         if found.group(3):  # author given (normal message)
185           curr_author = found.group(3)
186           curr_msg = found.group(4) + "\n"
187           if not len(me):  # if no "me" supplied:
188             me = curr_author  # the first name will be chosen
189         else:  # system message -> no author
190           curr_author = ""
191           curr_msg = found.group(2) + "\n"
192
193       else:  # continuing message lines
194         curr_msg += line
195
196     # save the very last message, too
197     if len(curr_time):
198       messages.append({'time': curr_time,
199                        'author': curr_author,
200                        'msg': curr_msg.strip()})
201
202   # get the contact's name
203   they = ""
204   for msg in messages:
205     if len(msg['author']) and msg['author'] != me:
206       they = msg['author']
207       break
```

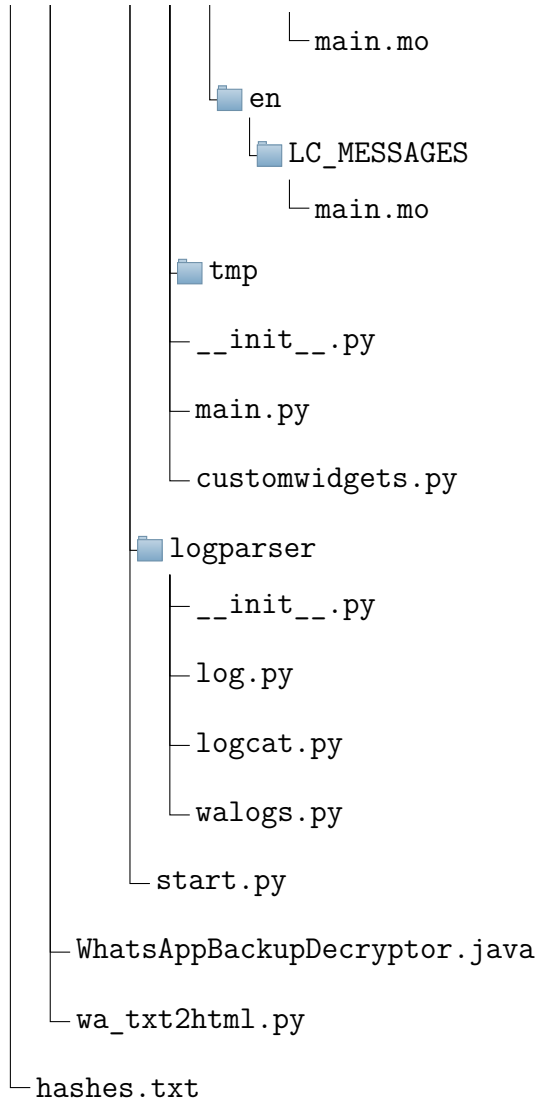
```
208
209   # write to html file
210   with open(outfile, "w") as html:
211     prepare_html(html, me, they, infile)  # filename as title
212
213     for msg in messages:  # write content
214       write_html_msg(html, len(msg['author']) == 0, msg['↩
      ↪ author'] == me, msg['author'], msg['msg'], msg['time'])
215
216     finish_html(html)
217
218   print("Successfully␣parsed␣{cnt}␣messages␣to␣HTML␣file␣{out↩
      ↪ }".format(cnt=len(messages), out=outfile))
219   sys.exit(0)
```

Listing C.2: Quelltext der Datei `wa_txt2html.py`

Anhang D: Inhalt des beiliegenden Datenträgers

Auf der CD, die dieser Bachelorarbeit beiliegt, befindet sich die digitale Fassung der Arbeit sowie alle Quelltexte, die zum WhatsApp Timeline Extractor gehören oder im Anhang C aufgeführt werden.
Zusätzlich ist die Datei `hashes.txt` enthalten, die die SHA1-Prüfsummen aller anderen auf der CD enthaltenen Dateien beinhaltet. So kann deren Integrität jederzeit überprüft werden.
Im Folgenden ist der Inhalt des Datenträgers vollständig aufgelistet.




main.mo
en
LC_MESSAGES
main.mo
tmp
__init__.py
main.py
customwidgets.py
logparser
__init__.py
log.py
logcat.py
walogs.py
start.py
WhatsAppBackupDecryptor.java
wa_txt2html.py
hashes.txt

Literaturverzeichnis

[Alh+17] John K. Alhassan, Bilikisu Abubakar, Morufu Olalere, Shafi'i Muhammad Abdulhamid und Suleiman Ahmad. "Forensic Acquisition of Data from a Crypt 12 Encrypted Database of Whatsapp". In: *2nd International Engineering Conference (IEC 2017)*. Federal University of Technology. Minna, 2017. URL: https://s3.amazonaws.com/academia.edu.documents/55215694/112.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1532685946&Signature=PlbORn9duBgL8wDW%2FYix0yD2v1g%3D&response-content-disposition=inline%3B%20filename%3DForensic_Acquisition_of_Data_from_a_Cryp.pdf (besucht am 27. 07. 2018).

[Bra+06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau und John Cowan. *Extensible Markup Language (XML): 1.1 (Second Edition)*. Spezifikation. W3C, 16. Aug. 2006. URL: https://www.w3.org/TR/xml11/ (besucht am 19. 06. 2018).

[BSI11] BSI, Hrsg. *Leitfaden „IT-Forensik"*. Version 1.0.1. Bundesamt für Sicherheit in der Informationstechnik. März 2011. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Cyber-Sicherheit/Themen/Leitfaden_IT-Forensik.pdf?__blob=publicationFile&v=2 (besucht am 21. 06. 2018).

[CSW12] D. Cortjens, A. Spruyt und W. F. C. Wieringa. *WhatsApp database encryption project report*. Technical Report. zitiert nach [Tha13]. Universiteit van Amsterdam, 2012.

[Dai+17] Zhongmin Dai, Sufatrio, Tong-Wei Chua, Dinesh Kumar Balakrishnan und Vrizlynn L. L. Thing. "Chat-App Decryption Key Extraction Through Information Flow Analysis". In: *A Systems Approach to Cyber Security. Proceedings of the 2nd Singapore Cyber-Security R&D Conference (SG-CRC 2017)*. Hrsg. von Abhik Roychoudhury und Yang Liu. Amsterdam: IOS Press BV, 2017, S. 3–18. URL: https://books.google.de/books?id=RUXiDgAAQBAJ (besucht am 16. 05. 2018).

[Ele18] Nikolay Elenkov. *Android backup extractor*. GitHub. 22. Mai 2018. URL: https://github.com/nelenkov/android-backup-extractor (besucht am 22. 06. 2018).

[Eli16a] EliteAndroidApps. *Omni-Crypt apk*. APKMonk. 22. Okt. 2016. URL: https://www.apkmonk.com/app/com.omnicrypt/ (besucht am 21. 07. 2018).

[Eli16b] EliteAndroidApps. *WhatsApp Key/DB Extractor*. GitHub. 21. Okt. 2016. URL: https://github.com/EliteAndroidApps/WhatsApp-Key-DB-Extractor (besucht am 21. 06. 2018).

[eMa15] eMarketer, Hrsg. *Top 3 Social Messaging Apps Among Smartphone Users in Selected Countries, Jan 2015*. zitiert nach: Baker, Dillon: The State of Messaging Apps, in 5 Charts. 2015. 2015. URL: https://contently.com/strategist/2015/06/30/the-state-of-messaging-apps-in-5-charts/ (besucht am 07. 05. 2018).

[eMa17] eMarketer, Hrsg. *Mobile Phone Messaging App Users Worldwide, 2016-2021 (billions and % change)*. 28. Juni 2017. URL: https://www.emarketer.com/Chart/Mobile-Phone-Messaging-App-Users-Worldwide-2016-2021-billions-change/209369 (besucht am 07. 05. 2018).

[fir18] firasdib. *regular expressions 101*. 2018. URL: https://regex101.com/ (besucht am 12. 07. 2018).

[Gar11] Marko Gargenta. *Learning Android*. Hrsg. von Andy Oram und Brian Jepson. Erste Auflage. Sebastopol: O'Reilly Media Inc., März 2011, S. 62–64. URL: https://books.google.de/books?id=oMYQz4_BW48C (besucht am 02. 07. 2018).

[Goo18a] Google LLC, Hrsg. *Android Debug Bridge (adb)*. 5. Juni 2018. URL: https://developer.android.com/studio/command-line/adb (besucht am 21. 06. 2018).

[Goo18b] Google LLC, Hrsg. *Android Studio*. 2018. URL: https://developer.android.com/studio/ (besucht am 19. 07. 2018).

[Goo18c] Google LLC, Hrsg. *Logcat command-line tool*. 5. Juni 2018. URL: https://developer.android.com/studio/command-line/logcat (besucht am 02. 07. 2018).

[Gup16] Umesh Gupta. "An Overview on the Architecture of WhatsApp". In: *International Journal of Computer Science & Engineering Technology* 7.7 (7. Juli 2016), S. 335–337. ISSN: 2229-3345. URL: http://www.ijcset.com/docs/IJCSET16-07-07-015.pdf (besucht am 09. 05. 2018).

[Hof14] Todd Hoff. *The WhatsApp Architecture Facebook Bought For $19 Billion*. 26. Feb. 2014. URL: http://highscalability.com/blog/2014/2/26/the-whatsapp-architecture-facebook-bought-for-19-billion.html (besucht am 27. 05. 2018).

[IDC17] IDC, Hrsg. *Smartphone OS Market Share*. 2017. URL: https://www.idc.com/promo/smartphone-market-share/os (besucht am 24. 07. 2018).

[Jos06] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. Internet Engineering Task Force (IETF). Okt. 2006. URL: https://www.rfc-editor.org/rfc/rfc4648.txt (besucht am 01. 06. 2018).

[Kly16] Alex Klyubin. *Disallow downgrading of non-debuggable packages*. Kommentar zu Commit 921dd754ab49df0cd580ff96503f7616c4c85f4a des Android-Quelltextes. 24. Feb. 2016. URL: https://android.googlesource.com/platform/frameworks/base/+/921dd75 (besucht am 22. 06. 2018).

[Kou09] Jan Koum. *some notes on the 2.3 release*. WhatsApp Inc. 12. Dez. 2009. URL: https://blog.whatsapp.com/106/some-notes-on-the-2.3-release (besucht am 07. 05. 2018).

[Kou11] Jan Koum. *Group chat*. WhatsApp Inc. 22. Feb. 2011. URL: https://blog.whatsapp.com/157/Group-chat (besucht am 07. 05. 2018).

[Kou13] Jan Koum. *Introducing Voice Messages*. 7. Aug. 2013. URL: https://blog.whatsapp.com/378/Introducing-Voice-Messages (besucht am 08. 05. 2018).

[Kou15] Jan Koum. *WhatsApp Web*. 21. Jan. 2015. URL: https://blog.whatsapp.com/614/WhatsApp-Web (besucht am 25. 05. 2018).

[Kou16a] Jan Koum. *Introducing WhatsApp's desktop app*. 10. Mai 2016. URL: https://blog.whatsapp.com/10000621/Introducing-WhatsApps-desktop-app (besucht am 08. 05. 2018).

[Kou16b] Jan Koum. *WhatsApp Video Calling*. 14. Nov. 2016. URL: https://blog.whatsapp.com/10000629/WhatsApp-Video-Calling (besucht am 08. 05. 2018).

[Kou17a] Jan Koum. *Share your live location*. 17. Okt. 2017. URL: https://blog.whatsapp.com/10000634/Share-your-live-location (besucht am 25. 05. 2018).

[Kou17b] Jan Koum. *WhatsApp Status*. 20. Feb. 2017. URL: https://blog.whatsapp.com/10000630/WhatsApp-Status (besucht am 08. 05. 2018).

[KA16] Jan Koum und Brian Acton. *end-to-end encryption*. WhatsApp Inc. 5. Apr. 2016. URL: https://blog.whatsapp.com/10000618/end-to-end-encryption (besucht am 25. 07. 2018).

[lca07] lcamtuf. *memfetch.tgz*. 3. Apr. 2007. URL: http://lcamtuf.coredump.cx/soft/memfetch.tgz (besucht am 27. 07. 2018).

[LM90] Edward A. Lee und David G. Messerschmitt. *Digital Communication*. Zweite. Dordrecht: Kluwer Academic Publishers, 1990. Kap. 1. URL: https://books.google.de/books?id=GjTwCAAAQBAJ (besucht am 17. 07. 2018).

[MZ06] A. Melnikov und K. Zeilenga. *Simple Authentication and Security Layer (SASL)*. Internet Engineering Task Force (IETF). Juni 2006. URL: https://www.rfc-editor.org/rfc/rfc4422.txt (besucht am 01. 06. 2018).

[Mil86] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology – CRYPT0 '85*. Berlin: Springer, 1986, S. 417–426. URL: https://link.springer.com/content/pdf/10.1007%2F3-540-39799-X_31.pdf (besucht am 22. 05. 2018).

[Ols14] Parmy Olson. *Exclusive: The Rags-To-Riches Tale Of How Jan Koum Built WhatsApp Into Facebook's New $19 Billion Baby*. Forbes. 19. Feb. 2014. URL: https://www.forbes.com/sites/parmyolson/2014/02/19/exclusive-inside-story-how-jan-koum-built-whatsapp-into-facebooks-new-19-billion-baby/#4b86db922fa1 (besucht am 07. 05. 2018).

[Ols15] Parmy Olson. *Facebook's Phone Company: WhatsApp Goes To The Next Level With Its Voice Calling Service*. Forbes. 7. Apr. 2015. URL: https://www.forbes.com/sites/parmyolson/2015/04/07/facebooks-whatsapp-voice-calling/#575f0dc1388c (besucht am 08. 05. 2018).

[Per16] Sarah Perez. *WhatsApp adds support for document sharing, but only PDFs at launch*. TechCrunch. 2. März 2016. URL: https://techcrunch.com/2016/03/02/whatsapp-adds-support-for-document-sharing-but-only-pdfs-at-launch/ (besucht am 07. 05. 2018).

[Pet18] Harry Pettit. *Pay back a friend with a WhatsApp message: Facebook-owned app rolls out a 'new payments service' in India ahead of a possible global launch*. MailOnline. 9. Feb. 2018. URL: http://www.dailymail.co.uk/sciencetech/article-5372559/WhatsApp-rolls-payment-message-service-India.html (besucht am 16. 05. 2018).

[Pia+18] Mauricio Piacentini, Raquel Ravanini, Jens Miltner, Pete Morgan, René Peinthor, Martin Kleusberg, Justin Clift und John T. Haller. *DB Browser for SQLite. The Official home of the DB Browser for SQLite*. 8. Juni 2018. URL: http://sqlitebrowser.org/ (besucht am 21. 07. 2018).

[Pic14] Francesco Picasso. *hotoloti*. Zena Forensics. 26. März 2014. URL: https://code.google.com/archive/p/hotoloti/downloads (besucht am 19. 07. 2018).

[Pri04] Burkhard Priemer. "Logfile-Analysen: Möglichkeiten und Grenzen ihrer Nutzung bei Untersuchungen zur Mensch-Maschine-Interaktion". In: *Medien Pädagogik. Einzelbeiträge* (2. Juni 2004). URL: http://medienpaed.com/article/view/188 (besucht am 02. 07. 2018).

[Ras18] Raspberry Pi Foundation, Hrsg. *Raspbian*. 2018. URL: https://www.raspberrypi.org/downloads/raspbian/ (besucht am 21. 06. 2018).

[RMS18] Paul Rösler, Christian Mainka und Jörg Schwenk. "More is Less: On the End-to-End Security of Group Chats in Signal, WhatsApp, and Threema". In: *Proceedings of 3rd IEEE European Symposium on Security and Privacy (EuroS&P 2018)*. Ruhr-Universität Bochum. Bochum, 15. Jan. 2018. URL: https://eprint.iacr.org/2017/713.pdf (besucht am 19. 07. 2018).

[Sah14] Shubham Sahu. "An Analysis of WhatsApp Forensics in Android Smartphones". In: *International Journal of Engineering Research* 3 (5 1. Mai 2014), S. 349–350. ISSN: 2319-6890. URL: https://s3.amazonaws.com/academia.edu.documents/33614013/IJER_2014_514.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1531733613&Signature=tU3MBk%2Fiuo5PmMyjJ4vLzE6Ziyc%3D&response-content-disposition=inline%3B%20filename%3DIjer_2014_514.pdf (besucht am 17. 07. 2018).

[Sai11] Peter Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. Internet Engineering Task Force (IETF). März 2011. URL: https://www.rfc-editor.org/rfc/rfc6120.txt (besucht am 09. 05. 2018).

[SST09] Peter Saint-Andre, Kevin Smith und Remko Troncon. *XMPP: The Definitive Guide*. Hrsg. von Mary E. Treseler. Erste Auflage. Sebastopol: O'Reilly Media Inc., Apr. 2009. URL: https://books.google.de/books?id=ChTCQYLIDfoC (besucht am 09. 05. 2018).

[SYS65] A. K. M. Nazmus Sakib, Fauzia Yasmeen und Sa Sir. "Key Agreement and Authentication Protocol for IEEE 802.11". In: *Global Journal of Computer Science and Technology* 11.20 (10. Dez. 1965), S. 59–64. ISSN: 0975-4172. URL: https://computerresearch.org/index.php/computer/article/view/1041/1039 (besucht am 19. 06. 2018).

[SDS16] Gandeva Bayu Satrya, Philip Tobianto Daely und Soo Young Shin. "Android Forensics Analysis: Private Chat on Social Messenger". In: *The Eighth International Conference on Ubiquitous and Future Networks 2016 (ICUFN 2O16)*. Juli 2016. URL: https://www.researchgate.net/profile/Philip_Daely/publication/305317074_Android_Forensics_Analysis_Private_Chat_on_Social_Messenger/links/5ac23c03aca27222c75c0c89/Android-Forensics-Analysis-Private-Chat-on-Social-Messenger.pdf (besucht am 07. 08. 2018).

[SB16] Mansur Zakariyya Shuaibu und Alhassan Bala. "WhatsApp Forensics and its Challenges for Android Smartphones". In: *Global Journal of Advanced Engineering Technologies and Sciences* 3 (5 Mai 2016), S. 68–75. ISSN: 2349-0292. URL: http://www.gjaets.com/Issues%20PDF/Archive-2016/May-2016/11.pdf (besucht am 19. 07. 2018).

[TT15] Rohit Tamma und Donnie Tindall. *Learning Android Forensics. A hands-on guide to Android forensics, from setting up the forensic workstation to analyzing key forensic artifacts*. Birmingham: Packt Publishing, Apr. 2015, S. 234–236. URL: https://books.google.de/books?id=hYnwCAAAQBAJ (besucht am 16. 05. 2018).

[Tha13] Neha S. Thakur. "Forensic Analysis of WhatsApp on Android Smartphones". In: *University of New Orleans Theses and Dissertations* 1706 (8. Juni 2013). URL: https://scholarworks.uno.edu/cgi/viewcontent.cgi?article=2736&context=td (besucht am 17. 07. 2018).

[The18] The Android Open Source Project. *logger.c*. Google Inc. 2. Juli 2018. URL: https://android.googlesource.com/kernel/msm/+/android-8.1.0_r0.81/drivers/staging/android/logger.c (besucht am 06. 08. 2018).

[Unc18] Uncensored Communications Group, Hrsg. *Citadel. The leader in true open source email and collaboration*. 2018. URL: http://www.citadel.org/doku.php (besucht am 21. 06. 2018).

[Wha17] WhatsApp Inc., Hrsg. *WhatsApp Encryption Overview. Technical white paper*. 19. Dez. 2017. URL: https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf (besucht am 10. 05. 2018).

[Wha18a] WhatsApp Inc., Hrsg. *Nachrichten löschen*. 2018. URL: https://faq.whatsapp.com/en/android/26000068?lang=de (besucht am 12. 08. 2018).

[Wha18b] WhatsApp Inc., Hrsg. *Support für ältere Betriebssysteme*. 2018. URL: https://faq.whatsapp.com/de/s40/26000006/ (besucht am 07. 05. 2018).

[Wha18c] WhatsApp Inc., Hrsg. *WhatsApp herunterladen*. 2018. URL: https://www.whatsapp.com/download/ (besucht am 07. 05. 2018).

[Wha18d] WhatsApp Inc., Hrsg. *WhatsApp Web*. 2018. URL: https://web.whatsapp.com/ (besucht am 07. 05. 2018).

[wxP18] wxPython Team. *Overview of wxPython*. 2018. URL: https://wxpython.org/pages/overview/ (besucht am 02. 07. 2018).

Selbstständigkeitserklärung

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.

Elias Ullrich

Mittweida, 22. August 2018