Praxissemesterbericht

Entwicklung und Veröffentlichung
des MifareClassicTools (MCT)

von Gerhard Klostermeier
Studiengang: IT-Sicherheit
Matrikelnummer: 31031
Betreuer: Prof. Roland Hellmann
Praxissemester: Wintersemester 2012/2013

Inhaltsverzeichnis

Tabellenverzeichnis....3
Abbildungsverzeichnis....3
Quellen....4
Abkürzungen....4
Definitionen und Begriffsklärung....5
1 Angaben zur Praxisstelle....7
2 Kurzfassung....8
3 Einleitung....9
3.1 Motivation....9
3.2 Problemstellung und Abgrenzung....10
3.3 Ziel der Arbeit....10
3.4 Vorgehensweise....11
4 Grundlagen....12
4.1 Mifare Classic-Technik....12
4.1.1 Speicheraufbau....13
4.1.2 Rechte und Schlüssel....15
4.2 Mifare Classic Hack....18
4.2.1 Dark Side Attack....19
4.2.2 Nested Authentication Attack....20
4.3 Android App-Progammierung....21
4.3.1 Grundlagen der App-Programmierung....21
4.3.2 Android NFC-API....24
5 Entwicklung der App....26
5.1 Allgemein....26
5.2 Funktionalität 1: Mifare Classic-Tags auslesen....28
5.2.1 Das Key File Konzept....29
5.3 Funktionalität 2: Der Dump-Editor....30
5.3.1 Daten als ASCII darstellen....31
5.3.2 Mifare Classic Access Condtitions darstellen....31
5.3.3 Mifare Classic Value Blocks als Dezimalzahl darstellen....32
5.4 Funktionalität 3: Mifare Classic-Tags beschreiben....32
5.4.1 Blockweise beschreiben....33
5.4.2 Kompletten Dump auf neuen Tag schreiben....33
5.5 Funktionalität 4: Der Schlüsseldateien-Editor....34
5.6 Funktionalität 5: In-App Hilfe und Informationen....35
6 Veröffentlichung....36
6.1 Lizenz....36
6.2 Veröffentlichungsweg....36
7 Aussichten....38

Tabellenverzeichnis

Tabelle 1: Mifare Classic Speicherzugriffsoperationen
(Quelle: [MF1k11])....15
Tabelle 2: Mifare Classic Access Conditions für Sector Trailers
(Quelle: [MF1k11])....17
Tabelle 3: Mifare Classic Access Conditions für Data und Value Blocks
(Quelle: [MF1k11])....17

Terminalverzeichnis

Terminal 1: Ausgabe des Proxmark3 nach erfolgreichem Dark Side Angriff....19
Terminal 2: Gekürzte Ausgabe des Proxmark3 nach erfolgreichem Nested
Authentication-Angriff....20

Abbildungsverzeichnis

Abbildung 1: Google Nexus 7 Tablet-Computer
(Quelle: http://www.google.de/nexus/7/)....9
Abbildung 2: Speicheraufbau eines Mifare Classic 1k Chips
(Quelle: [MF1k11])....13
Abbildung 3: Speicheraufbau eines Mifare Classic 4k Chips
(Quelle: [MF4k10])....14
Abbildung 4: Aufbau eines Mifare Classic Value Blocks
(Quelle: [MF1k11])....15
Abbildung 5: Aufbau der Mifare Classic Access Conditions und deren Zuordnung
zu den Blöcken
(Quelle: [MF1k11])....16
Abbildung 6: Universal RFID-Werkzeug Proxmark3 (ohne Antenne)
(Quelle: https://secure.flickr.com/photos/cooaoo/7038277107/)....18
Abbildung 7: Lebenszyklus einer Android-App
(Quelle: https://developer.android.com/training/basics/activity-lifecycle/)....22
Abbildung 8: NFC Tag Dispatch System
(Quelle: https://developer.android.com/guide/topics/connectivity/nfc/)....24
Abbildung 9: Samsung Galaxy Nexus (i9250) Smartphone
(Quelle: http://www.samsung.com/fr/article/all-about-android-4-0)....27
Abbildung 10: Datenaustausch beim Lesen eines Tags....28
Abbildung 11: Screenshot der CreateKeyMapActivity....29
Abbildung 12: Screenshot der DumpEditorActivity....30
Abbildung 13: Screenshot der HexToAsciiActivity....31
Abbildung 14: Screenshot der AcessConditionsActivity....31
Abbildung 15: Screenshot der ValueBlockActivity....32
Abbildung 16: Screenshot der WriteTagActivity....32
Abbildung 17: Screenshot der KeyEditorActivity....34
Abbildung 18: Screenshot der HelpActivity....35

Quellen

Fin08: Klaus Finkenzeller, RFID-Handbuch: Grundlagen und praktische Anwendungen von Transpondern, kontaktlosen Chipkarten und NFC, 2008

MF1k11: NXP Semiconductors, MF1S50yyX - MIFARE Classic 1K, 2011, http://www.nxp.com/documents/data_sheet/MF1S50YYX.pdf

MF4k10: NXP Semiconductors, MF1S703x - MIFARE Classic 4K, 2010, http://www.nxp.com/documents/data_sheet/MF1S703x.pdf

Plö08: Henryk Plötz, Mifare Classic - Eine Analyse der Implementierung, 2008, http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2008-21/SAR-PR-2008-21_.pdf

CNO08: Nicolas T. Courtois, Karsten Nohl, Sean O'Neil, Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards, 2008, http://eprint.iacr.org/2008/166.pdf

Cou09: Nicolas T. Courtois, The dark side of security by obscurity, 2009, http://eprint.iacr.org/2009/137.pdf

AndTut: Google, Android Developers - Trainig, ständig aktualisiert, https://developer.android.com/training/index.html

AndRef: Google, Android Developer - Reference, ständig aktualisiert, https://developer.android.com/reference/packages.html

AndGui: Google, Android Developer - API Guide, ständig aktualisiert, https://developer.android.com/guide/components/index.html

Abkürzungen

API Application Programming Interface, Programmierschnittstelle

APK (Android) Application Package File, Dateiformat von installierbaren Android-Anwendungen

App Application, Anwendung, Programm

ASCII American Standard Code for Information Interchange, eine 7 Bit Zeichenkodierung

GUI Graphical User Interface, grafische Benutzeroberfläche

HF (-RFID) High Frequency, hier meist um 13.56MHz

MCT Mifare Classic Tool, Name der selbst programmierten App

NDEF NFC Data Exchange Format, Datenformat für den Datenaustausch via Near Field Communication

NFC Near Field Communication, kontaktlose Datenübertragung über kurze Strecken (siehe Begriff: Near Field Communication)

RFID Radio-Frequency Identification, kontaktlose Identifikation

SDK Software Development Kit, Sammlung von Werkzeugen für die Anwendungsentwicklung

UID Unique Identifier, bei Mifare Classic eine 4 Byte Nummer

Definitionen und Begriffsklärung

- **Android**
  Android ist ein Betriebssystem für Geräte wie Smartphones, Tablets, Netbooks, usw. Es basiert auf dem Linux-Kernel und wird hauptsächlich von Google entwickelt. Mit dem frei erhältlichen Android SDK ist es jedem möglich sogenannte Apps für das Betriebssystem zu programmieren.
- **Card-Only**
  Card-Only bezeichnet eine bestimmte Klasse von Angriffen auf RFID-Systeme, bei denen der Zugriff auf die Karte (RFID-Tag) dem Angreifer ausreicht, um erfolgreich zu sein.
- **Dictionary-Attack**
  Eine Dictionary-Attack ist ein Angriff, bei dem der Angreifer automatisiert Wörter aus einer Liste (Wörterbuch) als Passwort ausprobiert. Diese Angriffsmethode ist oft effektiver als das Ausprobieren jeder möglichen Kombination an Buchstaben, da Menschen zu „normalen" Wörtern tendieren wenn es um die Wahl eines Passworts geht.
- **Dump**
  Ein Dump ist typischerweise eine Kopie oder ein Auszug eines Speicherinhaltes der in Rohdaten (z.B. Zeichenkette, Hexadezimal, Binär, usw.) vorliegt. Im Zusammenhang mit der MCT-App bezeichnet Dump einen Speicherauszug oder den kompletten Speicher eines Mifare Classic-Tags, der Hexadezimal in einer Datei abgespeichert ist.
- **In-App**
  Inhalte welche typischerweise im Internet (online) zu erwarten sind (z.B. Hilfe, zusätzliche Informationen), die jedoch in die Applikation (In-App) integriert wurden.
- **Little-Endian**
  Reihenfolge in der Bytes interpretiert werden.
  - Little-Endian: Niederwertigstes Bit zuerst
    (z.B. Tag.Monat.Jahr)
  - Big-Endian: Höchstwertiges Bit zuerst
    (z.B. Stunden:Minuten:Sekunden)
- **Mifare Classic**
  Mifare Classic ist eine proprietäre, ISO 14443 kompatible HF-RFID-Technik der Firma NXP Semiconductors.
- **Mifare Classic Hack**
  Im Zusammenhang mit diesem Bericht bezeichnet *Mifare Classic Hack* einen Kombinationsangriff aus Darkside Attack und Nested Authentication Attack. Mit diesen Angriffsmethoden können die Schlüssel eines Mifare Classic-Tags in wenigen Minuten geknackt werden. Dazu ist lediglich ein RFID-Reader oder der Proxmark3 nötig.

- **Near Field Communication**
  Standard zur kontaktlosen Datenübertragung über kurze Strecken (ca. 5cm). Er umfasst die Kommunikation von Transmittern zu passiven HF-RFID-Tags (ISO 14443, ISO 15693), als auch die mit gleichwertigen Transmittern.
- **Nexus 7**
  Das Nexus 7 ist ein Tablet-Computer der im Auftrag von Asus für Google produziert wird. Ein wichtiges Merkmal ist, dass dieses Gerät NFC-fähig ist und die Mifare Classic-Technik beherrscht.
- **Proxmark3**
  Das Universalgerät rund um RFID (siehe Abbildung 6). Es ist frei programmierbar und bietet durch die Community ständig neue Möglichkeiten. So sind beispielsweise Angriffe um die Schlüssel von Mifare Classic-Tags zu knacken, sowie die Möglichkeit einen solchen Tag samt UID zu emulieren schon vorhanden.
- **Read-Only**
  Read-Only bezeichnet einen Zustand, in dem ein einen Mifare Classic Block nicht geschrieben werden kann.
- **RFID-Reader**
  Ein RFID-Reader ist jegliche Form von Hardware, die es ermöglicht RIFD-Tags zu lesen aber auch zu schreiben.
- **RFID-Tag**
  RFID-Tag bezeichnet einen RFID-Chip mit Antenne in einer beliebigen Bauform (z.B. Schlüsselanhänger, Scheckkartenformat, usw.).

1 Angaben zur Praxisstelle

Unternehmen:	SySS GmbH
Branche:	IT-Sicherheit
Abteilung:	Labor
Adresse:	Wohlboldstr. 8 72072 Tübingen (Derendingen) Deutschland
Telefon:	07071-407856-0
E-Mail:	info@syss.de
Betreuer:	Christian Rothländer, Wolfgang Zejda
Freigegeben durch SySS:	_____ Unterschrift

2 Kurzfassung

In dem Praxissemester bei der SySS GmbH wurde eine Android-Anwendung entwickelt, welche es erlaubt mit *Near Field Communication*-fähigen Android-Geräten *Mifare Classic* RFID-Tags zu lesen, zu beschreiben und zu analysieren. Die Anwendung wurde abschließend als freie Software veröffentlicht. Zusätzlich wurde eine firmeninterne Dokumentation über das Knacken von *Mifare Classic* RFID-Tags mit dem *Proxmark3* verfasst.

3 Einleitung

Die etwas veraltete Mifare Classic RFID-Technik tritt noch häufig an vielen Stellen auf. So basieren Karten wie der Studentenausweis oder einige Zugangskontrollsysteme auf dieser Technik. Hauptproblem dabei ist, dass Mifare Classic seit 2008 nicht mehr sicher ist. Ein Angreifer kann die geheimen Schlüssel eines solchen Tags knacken und die auf dem Chip gespeicherten Daten lesen oder manipulieren.

Abbildung 1: Google Nexus 7 Tablet-Computer (Quelle: http://www.google.de/nexus/7/)

Ist man im Besitzt der geheimen Schlüssel (legal oder illegal), kann unter Verwendung eines Computers und eines angeschlossenen RFID-Readers, mit einem Tag kommuniziert werden. Dies ist in der Praxis oftmals unbequem. Kompakte Geräte wie das Nexus 7 verfügen über NFC-Hardware die als RFID-Reader für Mifare Classic Tags genutzt werden kann.

Als Problem bleibt eine geeignete Android-App zu finden, die es möglich macht Mifare Classic RFID-Tags zu lesen, frei zu beschreiben und für die Analyse die Daten visuell aufzubereiten. Da es eine solche App nicht gab, war Ziel des Praxissemesters eine derartige Anwendung zu programmieren.

3.1 Motivation

Für Consulting-Unternehmen im IT-Sicherheitsumfeld ist es wichtig, Wissen über jede Sicherheitskritische Technik zu erlangen. Ein eher untypisches Wissensfeld dabei ist RFID. Die Hauptmotivation ist demnach das Unternehmen in seinem Wissen um RFID zu stärken.

Eine Weitere Motivation war es, wie im Kapitel Einleitung beschrieben, eine Android-App zu entwickeln die es so noch nicht gab. Die Programmierung der Anwendung bietet unter anderem weitere Vorteile, wie das erarbeitete Wissen praktisch anzuwenden und zu vertiefen, oder durch eine gelungene Veröffentlichung den Firmennamen werbewirksam nach Außen zu tragen.

3.2 Problemstellung und Abgrenzung

Ein großes Problem bei der Sammlung von Wissen rund um RFID ist, dass es durch verschiedene Hersteller viele verschiedene Techniken gibt. Um nicht nur oberflächliche Kenntnisse dieser Techniken zu erlangen, wurde das Projekt auf Mifare Classic von *NXP Semiconductors* begrenzt.

Zweites Problem war es, die nötigen Fähigkeiten in der Android-Programmierung zu erlangen. Anwendungen für das Android-Betriebssystem werden zwar in der schon erlernten Java Programmiersprache geschrieben, unterscheiden sich jedoch deutlich von typischen Programmen für klassische Computer.

Ein wichtiges Abgrenzungsmerkmal war, dass die App für Menschen mit Kenntnissen im Bereich Mifare Classic ist. So müssen Begriffe, der technische Aufbau sowie das Rechte- und Schlüsselsystem dem Benutzer bekannt sein. Die App ist somit zur Nutzung auf technisch naher Ebene gedacht; Ein- und Ausgaben sollten Hexadezimal sein, was unter anderem dazu führen kann, dass die Bedienung möglicherweise nicht immer intuitiv ist.

Letzte wichtige Abgrenzung war es, keine App mit möglicherweise illegalen Funktionen zu erstellen. Die Anwendung soll nicht fähig sein Schlüssel von Mifare Classic RIFD-Tags zu knacken.

3.3 Ziel der Arbeit

Hauptziel der Arbeit war es eine Android-App zu programmieren, zu dokumentieren und zu veröffentlichen, die es dem Benutzer möglich macht Mifare Classic RFID-Tags zu lesen, zu schreiben und zu analysieren.

Als weiteres Ziel galt es, Wissen über die Mifare Classic-Technik zu erlangen und zu dokumentieren. Dazu gehörte unter anderem auch den „Mifare Classic Hack“ praktisch nachzuvollziehen.

3.4 Vorgehensweise

Um von der Problemstellung zum Ziel zu gelangen wurde eine fast durchgängig lineare Vorgehensweise gewählt. Lediglich Teil III wiederholt sich zyklisch.

Teil I (Mifare Classic-Technik und Hack)

1. Wissen über die Mifare Classic-Technik erlangen.
2. „Mifare Classic Hack" praktisch nachvollziehen.
3. Wissen über den „Mifare Classic Hack" für eine firmeninterne Verwendung dokumentieren.

Teil II (Android-App Programmierung)

1. Grundlagen der App-Programmierung lernen.
2. Lernen die NFC-API des Android-SDK zu nutzen.

Teil III (Entwicklung nach vertikalem Prototyping)

1. Einen Teil des Systems wählen, planen und durchgängig implementieren.
2. Systemteil (Funktion) testen.
3. Systemteil (Funktion) dokumentieren.
4. Schritt 1 bis 3 wiederholen bis alle Teile funktionieren und korrekt zusammenarbeiten.

Teil IV (Veröffentlichung)

1. Lizenz wählen.
2. Veröffentlichungsplattform / Veröffentlichungsweg wählen.

4 Grundlagen

Die Grundlagen zu Mifare Classic lassen sich frei aus dem Internet beziehen. Alternativ kann auf Literatur wie dem „RFID-Handbuch“ [Fin08] zurückgegriffen werden. Problem ist jedoch, dass es kaum Literatur zu RFID gibt und vor allem speziell zu Mifare Classic sind die vom Hersteller herausgegebenen Dokumente um ein wesentliches ausführlicher und umfangreicher.

Auch für die Android-App Programmierung wurde auf die im Internet von *Google* erhältliche Dokumentation zurückgegriffen. Zwar gibt es zu dem Thema wesentlich mehr Literatur, jedoch machen häufige Versionsänderungen an der Android API es schwer eine aktuelle Auflage zu bekommen.

4.1 Mifare Classic-Technik

Mifare Classic RFID-Tags fallen in die Klasse der sogenannten Smart-Tags. Entwickelt wurden diese jedoch Mitte der neunziger Jahre, was sie verglichen mit heutigen Smart-Tags nicht sehr „smart“ aussehen lässt. Im Prinzip ist ein Mifare Classic-Tag ein „Memory-Tag“, der Speicher zur Verfügung stellt, welchen er mit einem einfachen, an Schlüssel gebundenen Rechtemanagement verwaltet.

4.1.1 Speicheraufbau

Der Speicher eines Mifare Classic-Chips ist typischerweise 1 kB groß (Mifare Classic 1k). Eine andere Version bietet einen Speicher von 4 kB (Mifare Classic 4k). Der Aufbau ist dabei sehr ähnlich, jedoch nicht durchgehend identisch.

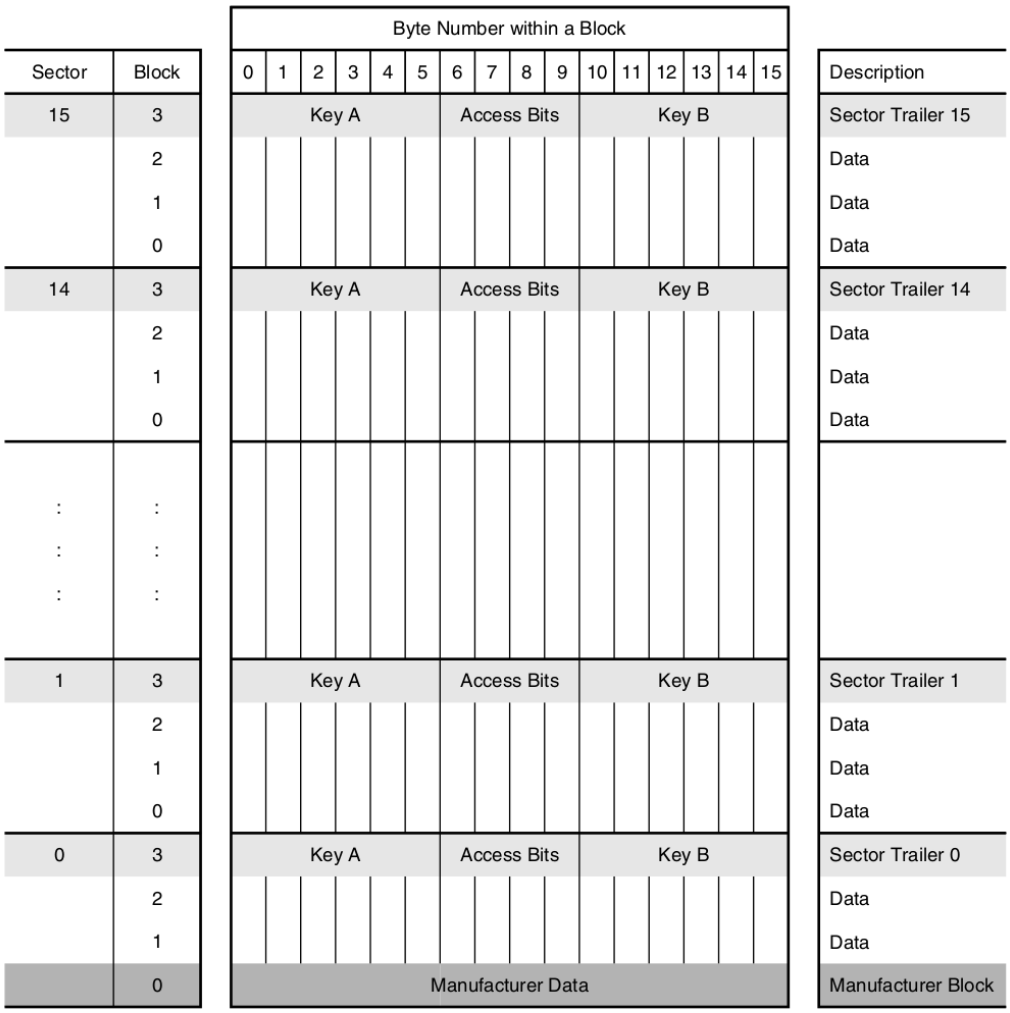

Abbildung 2: Speicheraufbau eines Mifare Classic 1k Chips
(Quelle: [MF1k11])

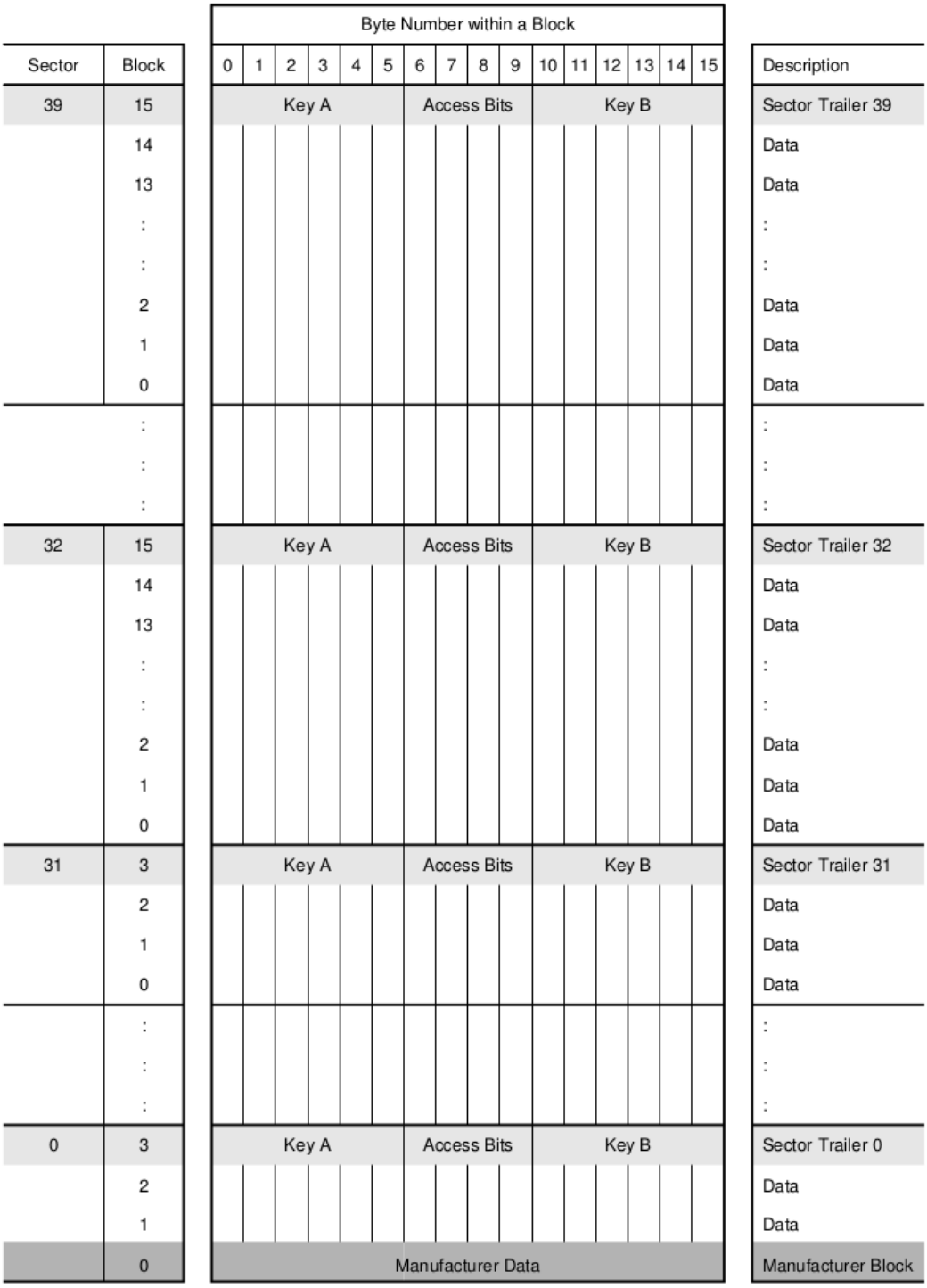

Abbildung 3: Speicheraufbau eines Mifare Classic 4k Chips
(Quelle: [MF4k10])

Der Speicheraufbau beider Mifare Classic-Chips ist zunächst gleich. Jeder Sektor hat vier Blöcke. Die ersten drei davon sind Datenblöcke, der Letzte ist der sogenannte Sector Trailer. Jeder Block besteht aus 16 Byte. Unterschied ist, dass Mifare Classic 1k 16 Sektoren in diesem Muster hat, Mifare Classic 4k jedoch 32 solcher Sektoren sowie 8 weitere, die pro Sektor 16 Blöcke enthalten.

Es kann Blockweise geschrieben werden, wobei Block 0 des Sektors 0 eine Ausnahme bildet, da er „read-only" ist und vom Hersteller vergeben wird. Er enthält die 4 Byte UID und weitere Herstellerinformationen.

Eine besondere Form eines Blocks ist der sogenannte *Value Block.* Er unterscheidet sich von typischen Blöcken lediglich in der Formatierung seines Inhalts. Falls ein Block wie nachfolgend illustriert aufgebaut ist, können weitere Operationen (increment, decrement, transfer, restore) genutzt werden, um seinen Inhalt zu manipulieren.

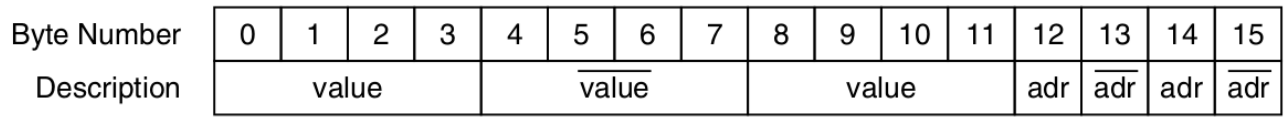

Abbildung 4: Aufbau eines Mifare Classic Value Blocks
(Quelle: [MF1k11])

4.1.2 Rechte und Schlüssel

Um auf die verschiedenen Blöcke im Speicher zuzugreifen gibt es unterschiedliche Rechte.

Operation	Beschreibung	Anmerkung
Read	Einen Block lesen	Gültig für alle Blöcke
Write	Einen Block schreiben	Gültig für alle Blöcke außer Manufacturer Block
Increment	Einen Block inkrementieren und den Wert in einem internen Register abspeichern	Gültig für Value Blocks
Decrement	Einen Block dekrementieren und den Wert in einem internen Register abspeichern	Gültig für Value Blocks
Transfer	Den Wert des internen Registers in einen Block schreiben	Gültig für Value Blocks
Restore	Den Wert eines Blocks in das interne Register schreiben	Gültig für Value Blocks

Tabelle 1: Mifare Classic Speicherzugriffsoperationen
(Quelle: [MF1k11])

Die Zugriffsrechte für jeden Datenblock eines Sektors, sowie für dessen Sector Trailer, werden in den Access Bits verwaltet. Diese Access Bits befinden sich im Sector Trailer jedes Sektors als Byte Nummer 6, 7, 8 und 9.

Access Bits	Valid Commands		Block	Description
$C1_3$, $C2_3$, $C3_3$	read, write	→	3	sector trailer
$C1_2$, $C2_2$, $C3_2$	read, write, increment, decrement, transfer, restore	→	2	data block
$C1_1$, $C2_1$, $C3_1$	read, write, increment, decrement, transfer, restore	→	1	data block
$C1_0$,$C2_0$, $C3_0$	read, write, increment, decrement, transfer, restore	→	0	data block

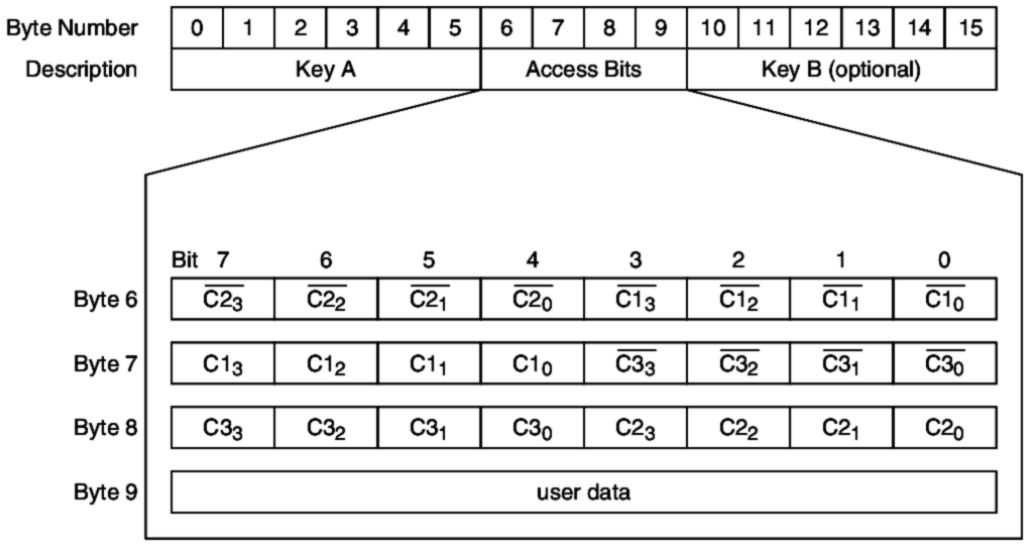

Abbildung 5: Aufbau der Mifare Classic Access Conditions und deren Zuordnung zu den Blöcken
(Quelle: [MF1k11])

Bei der Auswertung von C1, C2 und C3 ist darauf zu achten, ob sie einem Datenblock, einem Value Block oder einem Sector Trailer zugeordnet werden.

Access Bits			**Access Conditions für**					
			Key A		**Access Bits**		**Key B**	
C1	C2	C3	Read	Write	Read	Write	Read	Write
0	0	0	Nie	Key A	Key A	Nie	Key A	Key A
0	1	0	Nie	Nie	Key A	Nie	Key A	Nie
1	0	0	Nie	Key B	Key A|B	Nie	Nie	Key B
1	1	0	Nie	Nie	Key A|B	Nie	Nie	Nie
0	0	1	Nie	Key A	Key A	Key A	Key A	Key A
0	1	1	Nie	Key B	Key A|B	Key B	Nie	Key B
1	0	1	Nie	Nie	Key A|B	Key B	Nie	Nie
1	1	1	Nie	Nie	Key A|B	Nie	Nie	Nie

Tabelle 2: Mifare Classic Access Conditions für Sector Trailers
(Quelle: [MF1k11])

Access Bits			**Access Conditions für**			
C1	C2	C3	Read	Write	Increment	Decrement, Tansfer, Restore
0	0	0	Key A|B	Key A|B	Key A|B	Key A|B
0	1	0	Key A|B	Nie	Nie	Nie
1	0	0	Key A|B	Key B	Nie	Nie
1	1	0	Key A|B	Key B	Key B	Key A|B
0	0	1	Key A|B	Nie	Nie	Key A|B
0	1	1	Key B	Key B	Nie	Nie
1	0	1	Key B	Nie	Nie	Nie
1	1	1	Nie	Nie	Nie	Nie

Tabelle 3: Mifare Classic Access Conditions für Data und Value Blocks
(Quelle: [MF1k11])

Alle Tabellen, Grafiken und weitere Informationen sind den offiziellen Datenblättern des Herstellers entnommen. [MF1k11] [MF4k10]

4.2 Mifare Classic Hack

Ende 2007 präsentierte ein Sicherheitsforscherteam um Henryk Plötz und Karsten Nohl wie es ihnen gelang, durch Abätzen und Abschleifen eines Mifare Classic-Chips und die anschließende Analyse unter dem Rasterelektronenmikroskop, den proprietären Verschlüsselungsalgorithmus „Crypto-1“ zu rekonstruieren. In Folge darauf erschienen immer mehr Veröffentlichungen, welche systematische Fehler im Algorithmus oder Probleme mit dem Pseudozufallszahlengenerator aufdeckten. [Plö08] [CNO08] [Cou09]

Eine Sammlung von weiteren Veröffentlichungen um die Sicherheit von Mifare Classic kann unter http://www.proxmark.org/files/Documents/13.56%20MHz%20-%20MIFARE%20Classic/ gefunden werden.

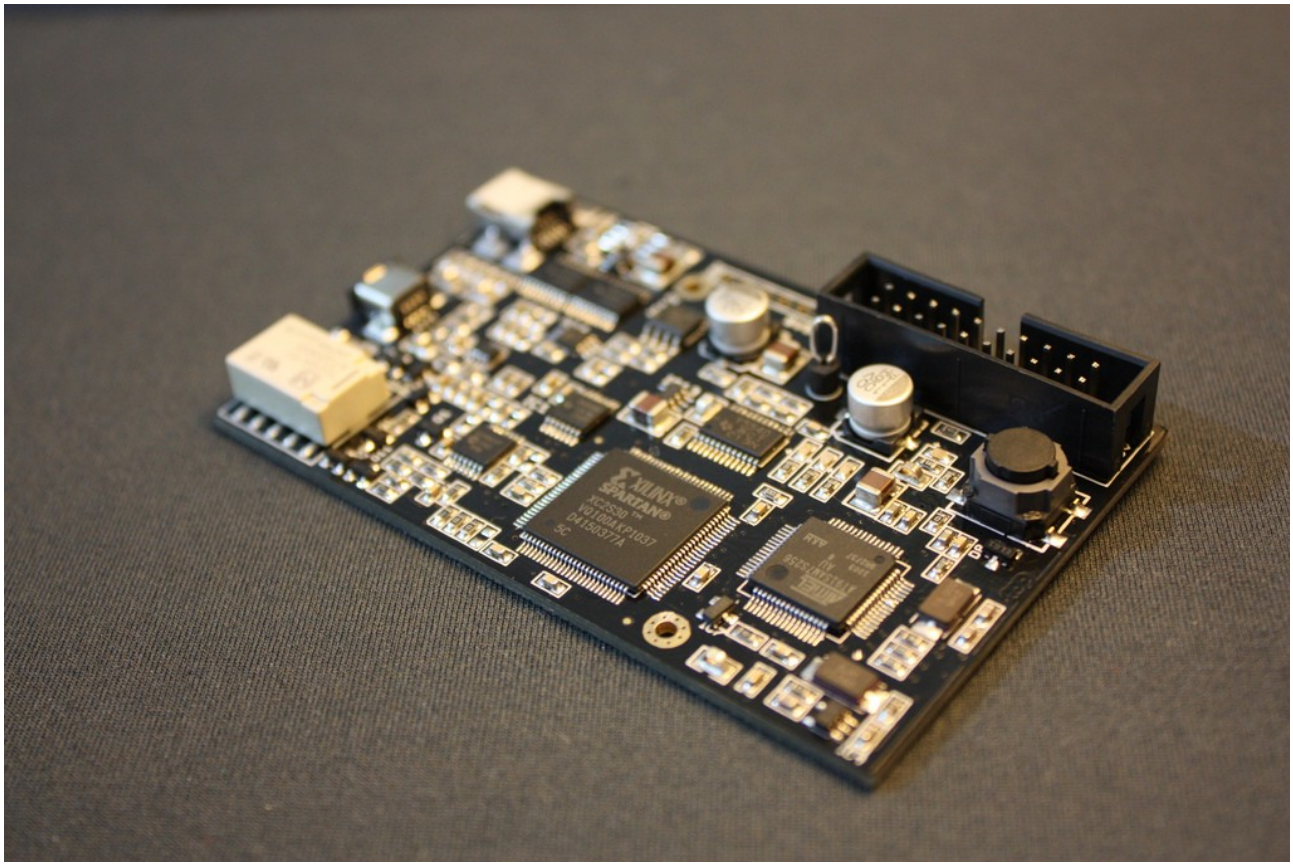

Abbildung 6: Universal RFID-Werkzeug Proxmark3 (ohne Antenne)
(Quelle: https://secure.flickr.com/photos/cooaoo/7038277107/)

Zwei der Angriffsmethoden (*Dark Side Attack* und *Nested Authentication Attack*) sind hier herauszuheben, da sie beide „card-only“ Attacken sind und kombiniert alle 32 Schlüssel eines Mifare Classic 1k-Tags innerhalb von Minuten in der Praxis knacken können. Dazu genügt ein handelsüblicher RFID-Reader, der mit spezieller Software angesprochen wird. Noch einfacher ist es, den Proxmark3 zu nutzen, bei welchem durch die Community beide Angriffe schon implementiert sind.

In den folgenden Kapiteln wird kurz vorgestellt, wie mit dem Proxmark3 die Schlüssel eines Mifare Classic 1k-Tags geknackt werden können.

4.2.1 Dark Side Attack

Der erste Schritt besteht darin, in den Besitz eines Schlüssels zu kommen. Für die darauf folgende *Nested Authentication Attack* ist es irrelevant welcher Schlüssel bekannt ist, mindestens einer wird jedoch benötigt. Die *Dark Side Attack* macht genau dies möglich, da mit ihr jeder beliebige Schlüssel eines Mifare Classic-Tags geknackt werden kann. Auf dem Promark3 ist der Angriff jedoch so implementiert, dass nur Schlüssel A des Sektors 0 geknackt wird.

```
proxmark3> hf mf mifare
--------------------------------------------------------------------------
Executing command. It may take up to 30 min.
Press the key on the proxmark3 device to abort both proxmark3 and client.
--------------------------------------------------------------------------
.....................#db# COMMAND mifare FINISHED


isOk:01


uid(1a34a0d1) nt(d681972d) par(d31b7b934323abeb) ks(07070f080c040d07)


|diff|{nr}    |ks3|ks3^5|parity         |
+----+--------+---+-----+---------------+
| 00 |00000000| 7 |  2  |1,1,0,0,1,0,1,1|
| 20 |00000020| 7 |  2  |1,1,0,1,1,0,0,0|
| 40 |00000040| f |  a  |1,1,0,1,1,1,1,0|
| 60 |00000060| 8 |  d  |1,1,0,0,1,0,0,1|
| 80 |00000080| c |  9  |1,1,0,0,0,0,1,0|
| a0 |000000a0| 4 |  1  |1,1,0,0,0,1,0,0|
| c0 |000000c0| d |  8  |1,1,0,1,0,1,0,1|
| e0 |000000e0| 7 |  2  |1,1,0,1,0,1,1,1|
00ff7596|00ff5769
--------------------------------------------------------------------
Key found:ffffffffffff

Found valid key:ffffffffffff
proxmark3>
```

Terminal 1: Ausgabe des Proxmark3 nach erfolgreichem Dark Side Angriff.

4.2.2 Nested Authentication Attack

Nach dem nun ein Schlüssel durch die *Dark Side Attack* bekannt ist, können die restlichen 31 Schlüssel durch die *Nested Authentication Attack* schnell gewonnen werden. Es wäre durchaus möglich mit der *Dark Side Attack* auch die restlichen Schlüssel zu knacken, jedoch ist der *Nested Authentication* Angriff wesentlich effektiver.

```
proxmark3> hf mf nested 1 0 a ffffffffffff
--block no:00 key type:00 key:a0 a1 a2 a3 a4 a5  etrans:0
Block shift=0
Testing known keys. Sector count=16
nested...


uid:a75c8a90 len=1 trgbl=0 trgkey=1
.uid:a75c8a90 len=1 trgbl=0 trgkey=1
.uid:a75c8a90 len=1 trgbl=0 trgkey=1
.uid:a75c8a90 len=1 trgbl=0 trgkey=1
.uid:a75c8a90 len=1 trgbl=0 trgkey=1
.-----------------------------------------------------------------
Total keys count:477934


.uid:a75c8a90 len=4 trgbl=4 trgkey=0
.uid:a75c8a90 len=1 trgbl=4 trgkey=0
.uid:a75c8a90 len=4 trgbl=4 trgkey=0
.uid:a75c8a90 len=4 trgbl=4 trgkey=0
.-----------------------------------------------------------------
Total keys count:1009397
Found valid key:aaaaaaaaaaaa

[...]

Total keys count:393974
Iterations count: 113
|---|----------------|---|----------------|---|
|sec|key A           |res|key B           |res|
|---|----------------|---|----------------|---|
|000|  ffffffffffff  | 1 |  111111111111  | 1 |
|001|  222222222222  | 1 |  333333333333  | 1 |
|002|  444444444444  | 1 |  555555555555  | 1 |
|003|  666666666666  | 1 |  777777777777  | 1 |
|004|  888888888888  | 1 |  999999999999  | 1 |
|005|  aaaaaaaaaaaa  | 1 |  bbbbbbbbbbbb  | 1 |
|006|  cccccccccccc  | 1 |  dddddddddddd  | 1 |
|007|  eeeeeeeeeeee  | 1 |  ffffffffffff  | 1 |
|008|  ffffffffffff  | 1 |  eeeeeeeeeeee  | 1 |
|009|  dddddddddddd  | 1 |  cccccccccccc  | 1 |
|010|  bbbbbbbbbbbb  | 1 |  aaaaaaaaaaaa  | 1 |
|011|  999999999999  | 1 |  888888888888  | 1 |
|012|  777777777777  | 1 |  666666666666  | 1 |
|013|  555555555555  | 1 |  444444444444  | 1 |
|014|  333333333333  | 1 |  222222222222  | 1 |
|015|  111111111111  | 1 |  000000000000  | 1 |
|---|----------------|---|----------------|---|
```

Terminal 2: Gekürzte Ausgabe des Proxmark3 nach erfolgreichem Nested Authentication-Angriff.

Nun sind alle 32 Schlüssel (A und B) des Mifare Classic 1k-Tags bekannt (siehe „Tabelle" am Ende der Ausgabe von Terminal 2).

4.3 Android App-Progammierung

Da Android-Programme typischerweise in Java entwickelt werden, ist es für Programmierer die dieser Sprache mächtig sind, ein nicht allzu großer Aufwand schnelle erste Ergebnisse zu erzielen. Die übersichtliche Dokumentation und die vielen Selbstlerneinheiten (*Tutorials*) sind eine weitere Hilfe, mit der schnell voran gekommen werden kann. [AndTut] [AndRef]

4.3.1 Grundlagen der App-Programmierung

Um die Grundlagen zu erlernen, wurden systematisch die von Google angebotenen Tutorials der Kategorie „Getting Started" durchgearbeitet. Das erlernte Wissen dabei lässt sich in folgende Punkte gliedern (wichtigere Punkte sind **fett** hervorgehoben):

- Eine erste App schreiben (Hello-World-App)
  - Ein Projekt erstellen
  - **Eine App ausführen**
    Hier gibt es die Möglichkeit eine Anwendung direkt auf einem Android-Gerät auszuführen, oder den mitgelieferten Android-Emulator zu nutzen. Zur Entwicklung einer NFC-App ist jedoch ein Android-Gerät nötig, da ein klassischer PC keine NFC-Hardware besitzt.
  - Eine einfache Oberfläche erstellen
  - **Eine andere „Activity" starten**
    Prinzipiell basiert jede Oberfläche mit der ein Benutzer interagieren kann auf der „Activity"-Klasse. Apps lassen sich demnach in mehrere Activities unterteilen.

- Lebenszyklus einer Android App
  - **Eine Activity starten**
  - **Eine Activity pausieren bzw. fortsetzen**
  - **Eine Activity stoppen bzw. neustarten**

    Es ist wichtig richtig zu Entscheiden wann welche Ressourcen freigegeben werden und wann welche Inhalte abgespeichert werden.

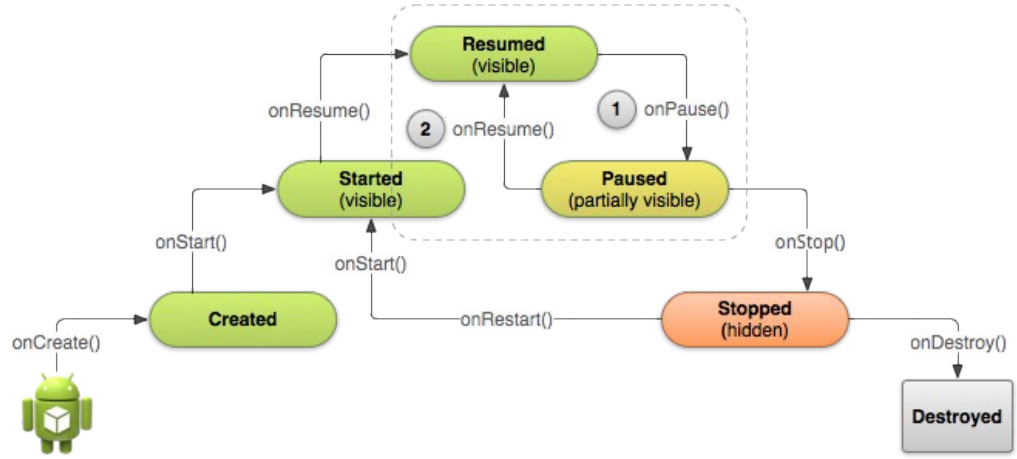

Abbildung 7: Lebenszyklus einer Android-App
(Quelle: https://developer.android.com/training/basics/activity-lifecycle/)

- Verschiedene Geräte unterstützen
  - **Verschiedene Sprachen unterstützen**
    Wenn Sprachinhalte (Text) die auf dem Display erscheinen sollen in eine XML-Datei anstatt in den Quelltext geschrieben werden, kann später die App bequem in verschiedene Sprachen übersetzt werden.
  - **Verschiedene Bildschirmgrößen unterstützen**
    Die Bildschirmauflösungen und die Bildschirmgrößen unterscheiden sich bei Android-Geräten häufig. Auch kann es für das Design der Oberfläche wichtig sein, ob sich das Gerät im Hochformat oder im Querformat befindet. Für die verschieden großen Bildschirme, sowie für die verschiedenen Formate lassen sich XML-Dateien erstellen, die je nach Größe bzw. Format das richtige Design (Layout) wählen.
  - **Verschiedene Android-Versionen unterstützen**
    Um möglichst viele Geräte zu unterstützen sollte eine alte Android-API Version gewählt werden. Probleme treten jedoch auf falls es nötig ist Module zu nutzen, die es nur in neueren Versionen gibt. Abhilfe kann in manchen Fällen die Android Support Library schaffen.

- Dynamische Oberflächen erstellen
  - Nutzung der Android Support Library
  - Erstellung eines Fragments
  - Erstellung einer dynamischen Oberfläche
  - Kommunikation mit anderen Fragmenten
- Daten speichern
  - Kleine Datenmengen in den „Shared Preferences" speichern
  - **Dateien speichern**
    Dateien können wie mit Java gewohnt geschrieben werden. Wichtig ist, ob man in den internen oder in den externen Gerätespeicher (meist eine SD Karte) schreiben will.

    Der interne Speicher ist immer vorhanden, die gespeicherten Daten können nur von der eigenen App gelesen werden und alle gespeicherten Daten sind nach dem Deinstallieren der App gelöscht.

    Der externe Speicher ist möglicherweise nicht vorhanden, die auf ihm gespeicherten Dateien können von Dritten gelesen werden und beim Deinstallieren der App werden Daten die an beliebigen Stellen abgespeichert wurden (nicht unter `getExternalFilesDir()`) nicht gelöscht.
  - Daten in SQL-Datenbanken speichern
- **Mit anderen Apps/Activities kommunizieren**
  Um zwischen den Activities der eigenen App oder gar mit einer Activity einer anderen App zu kommunizieren wird die „Intent"-Klasse genutzt.
  - Eine andere App starten (mit Aktion, z.B. Telefon App starten und eine Nummer wählen)
  - Den Rückgabewert einer anderen Activity (App) abgreifen (wenn sie von der eigenen App gestartet wurde)
  - Anderen Apps erlauben die Eigene zu starten
- Inhalt mit anderen Apps teilen
  - Inhalt an andere Apps senden
  - Inhalt von anderen Apps empfangen

4.3.2 Android NFC-API

Nachdem die Grundlagen der Android-Programmierung bekannt sind, ist für NFC-Apps, wie der entstandenen MifareClassicTool-App, das Wissen rund um die Android-NFC-API wichtig.

Google stellt neben den einfachen Tutorials (auch *Training* genannt), zusätzlich zur technischen Dokumentation (*Reference*), eine weitere Dokumentation in Prosa zur Verfügung (*API Guide*). Sie behandelt eher spezielle Teile der Android-API wie beispielsweise NFC. [AndGui]

Das Tag Dispatch System

Im Allgemeinen verwaltet das Betriebssystem Android selbst NFC-Ereignisse. Wenn beispielsweise ein Mifare Classic-Tag in den Antennenbereich kommt, wird dies von Android registriert und der Chip anschließend analysiert. Die Analyse kann entweder zu dem Ergebnis kommen, dass es sich um einen NDEF-formatierten Tag, oder um einen nicht NDEF-formatierten Tag handelt. Wenn ein NDEF-formatierter Tag vorliegt, kontrolliert Android, ob es Activities gibt die diese Art von Tags verarbeiten möchten. Sollte es keine Activity geben die angegeben hat solche Tags zu bearbeiten, fragt das Betriebssystem als nächstes ob es Activities gibt, die sich für eine bestimmte Art von Technik (z.B. Mifare Classic) interessieren. Falls dies auch nicht der Fall sein sollte, wird als letztes getestet ob es Activities gibt, die egal welche Technik vorliegt, den Tag verarbeiten möchten. Sollte in irgendeinen der drei Stadien eine Activity den Tag haben wollen, so wird er per *Intent* an sie weitergegeben.

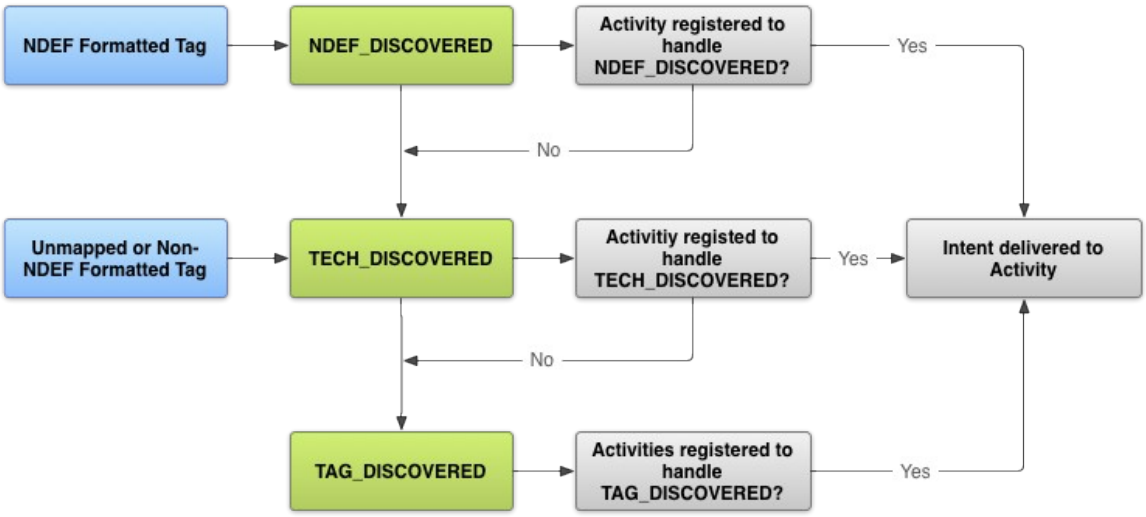

Abbildung 8: NFC Tag Dispatch System
(Quelle: https://developer.android.com/guide/topics/connectivity/nfc/)

Das Foreground Dispatch System

Wenn sich eine App im Vordergrund befindet, räumt ihr das Android-Betriebssystem das zusätzliche Recht ein, die NFC-Hardware exklusiv zu nutzen. Wenn eine Anwendung das *Foreground Dispatch System* nutzt, wird nicht wie in Abbildung 8 beschrieben der RFID-Tag an eine der registrierten Apps weitergegeben, stattdessen wird er direkt an die im Vordergrund laufende Applikation übergeben.

NFC-Hardware- und Technikverfügbarkeit

Es besitzen bei weitem nicht alle Android-Geräte NFC-Hardware und selbst Geräte die diese haben, unterstützen nicht immer die Mifare Classic-Technik.

- **NFC-Hardware**
  Die Android-AHelveticaPI bietet eine einfache Möglichkeit festzustellen, ob das Gerät NFC-Fähigkeiten besitzt.
  (`android.nfc.NfcAdapter.getDefaultAdapter(Context context) != null`)
- **Mifare Classic-Technik**
  Da es sich bei Mifare Classic um eine proprietär Technik handelt, ist es optional dass ein Gerät mit NFC-Hardware diese Technik unterstützt. Um herauszufinden ob die Technik verfügbar ist, wird kontrolliert ob das Java-Paket „com.nxp.mifare“ vorhanden ist.
  (`getPackageManager().hasSystemFeature("com.nxp.mifare") == true`)

5 Entwicklung der App

Im Allgemeinen wurde die Entwicklung der App nach dem Modell des vertikalen Prototyping durchgeführt. Das heißt, jede Funktionalität wurde einzeln betrachtet und durchlebte die Phasen Planung, Implementierung, Testen und Dokumentierung.

5.1 Allgemein

Bevor mit dem eigentlichen Programmieren begonnen werden kann, muss für eine Android-App das sogenannte Android-Manifest (`AndroidManifest.xml`) erstellt werden. Es enthält Anforderungen welche das Gerät erfüllen muss, wenn es die App ausführen möchte.

- Android Version
  - Mindestens API Version 10 (Android 2.3.3), da die NFC-Funktionalität erst mit dieser Version richtig ausgeschöpft wurde.
  - Maximal API Version 17 (Android 4.2.2). Dies war zum Zeitpunkt 15.02.2013 die aktuelle Version.
- Rechte
  - Nutzung der NFC-Hardware.
  - Nutzung des externen Datenspeichers (oftmals: SD Karte), um Schüssel und ausgelesene RFID-Tags zu speichern.
- Tag Dispatch System
  - Registrieren, dass das Hauptmenü (`MainActivity`) `TECH_DISCOVERED`-Tags verarbeiten möchte (siehe Android NFC-API).
  - Filter erstellen, sodass nur Tags mit der Mifare Classic-Technik an die App durch das Tag Dispatch System weiter gegeben werden.
  - `MainActivity`, Hautmenü.
  - `ReadTagActivity`, einen Mifare Classic-Tag auslesen.
  - `WriteTagActivity`, einen Mifare Classic-Tag beschreiben (Blockweise oder einen ganzen Dump).
  - `DumpEditorActivity`, einen ausgelesenen oder abgespeicherten Mifare Classic-Tag darstellen.
  - `KeyEditorActivity`, Schlüsseldateien darstellen.
  - `HelpActivity`, Hilfe und Informationen zur Nutzung der App darstellen.

- `CreateKeyMapActivity`, Schlüssel aus den Schlüsseldateien den Sektoren eines Tags zuordnen.
- `FileChooserActivity`, Generischer Dialog zum Auswählen von Schlüssel- oder Dump-Dateien.
- `AccessConditionsActivity`, darstellen der Mifare Classic Access Conditions (siehe Mifare Classic-Technik).
- `ValueBlocksActivity`, darstellen der Mifare Classic Value Blocks als dezimale Zahl (siehe Mifare Classic-Technik).
- `HexToAsciiActivity`, Daten eines Mifare Classic-Tags als US-ASCII (7 Bit) darstellen.

Sprache

Als Sprache für Quellcodekommentare, die Bezeichner im Quellcode, die technische Dokumentation sowie die in die App integrierte Hilfe wurde (US-)Englisch gewählt.

Geräte

Als Android-Geräte auf denen die Anwendung getestet werden kann, standen das Google Nexus 7 Tablet (siehe Abbildung 1) sowie das Samsung Galaxy Nexus (i9250) Smartphone (siehe Abbildung 9) zur Verfügung.

Abbildung 9: Samsung Galaxy Nexus (i9250) Smartphone
(Quelle: http://www.samsung.com/fr/article/all-about-android-4-0)

5.2 Funktionalität 1: Mifare Classic-Tags auslesen

Eine der wichtigsten Grundfunktionen der App ist das Auslesen eines Mifare Classic RFID-Tags. Um sich von anderen Anwendungen mit ähnlicher Funktionalität abzugrenzen, sollte dies auf eine für den Benutzer möglichst einfach Art geschehen. Des Weiteren war Anforderung, den Lesebereich einschränken zu können, sodass nur von Sektor X bis Sektor Y gelesen werden kann.

Um diese Aufgaben zu erfüllen wurden drei Klassen erstellt, die sich die Arbeit sinnvoll teilen.

- `ReadTagActivity (GUI)`
  Kümmert sich darum, dass eine Zuordnung zwischen Schlüsseln und Sektoren erstellt wird und liest anschließend auf deren Basis so viele Daten wie möglich vom Tag.
- `CreateKeyMapActivity (GUI)`
  Hier kann der Benutzer einstellen, welche Schlüsseldateien er verwenden möchte und in welchem Bereich die Zuordnung zwischen Schlüssel und den dazu passenden Sektoren gemacht werden soll (siehe Das Key File Konzept). Die Zuordnung wird mit Hilfe der MCReader-Klasse erstellt. Damit die Klasse auch beim Schreiben von Blöcken wieder zum Einsatz kommen kann, wurde sie generisch programmiert.
- `MCReader`
  Repräsentiert die Schnittelle zum RFID-Tag und bietet die grundlegenden Funktionen wie „Authentifizierung" oder „Sektor lesen" (für spätere Zwecke dann auch „Block schreiben" und weitere).

Der Datenaustausch läuft wie folgend dargestellt ab:

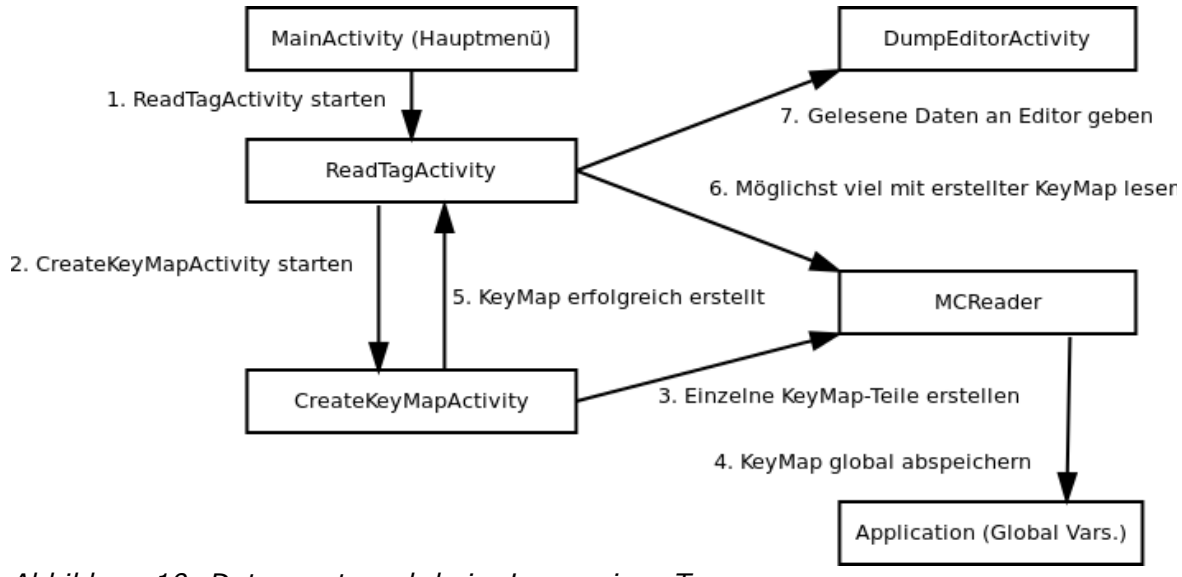

Abbildung 10: Datenaustausch beim Lesen eines Tags

5.2.1 Das Key File Konzept

Da es für den Benutzer umständlich ist, für jeden Schlüssel anzugeben zu welchem Sektor er gehört und ob er als Schlüssel A oder als Schlüssel B zum Einsatz kommen soll, wird auf ein „Dictionary-Attack" ähnliches Verfahren zurückgegriffen. Dazu werden beliebig viele Schlüssel in eine einfache Textdatei eingetragen (einen Schlüssel pro Zeile). Diese Schlüsseldatei (Key File) repräsentiert das Wörterbuch der *Dictionary Attack*. Nun probiert die App systematisch für jeden Sektor aus, ob eine Authentifizierung mit einem der in der Datei enthaltenen Schlüssel erfolgreich beendet werden kann. Dies wird für jeden Schlüssel sogar zweimal geprüft, wobei der Schlüssel beim ersten Mal als Schlüssel A verwendet wird und beim zweiten Mal als Schlüssel B. Dadurch entsteht eine Zuordnung (KeyMap) zwischen einem Sektor X und einem Schlüssel Y, mit der Eigenschaft ob dieser als Schlüssel A oder B verwendet werden muss.

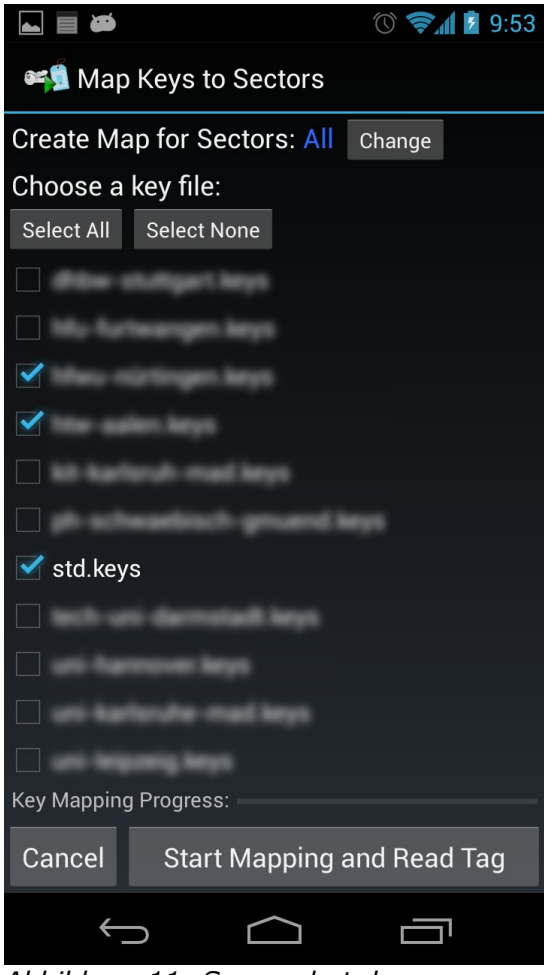

Abbildung 11: Screenshot der CreateKeyMapActivity

Vorteile des Key File Konzepts sind:

- Für den Benutzer ist es unwichtig welcher Schlüssel zu welchem Sektor gehört. Eine aufwendige Eingabe mit Zuordnung ist somit nicht nötig.
- Es ist nicht nötig dass alle Schlüssel bekannt sind. Falls kein passender Schlüssel zum lesen in einer der gewählten Schlüsseldateien gefunden werden kann, wird der Sektor übersprungen.

5.3 Funktionalität 2: Der Dump-Editor

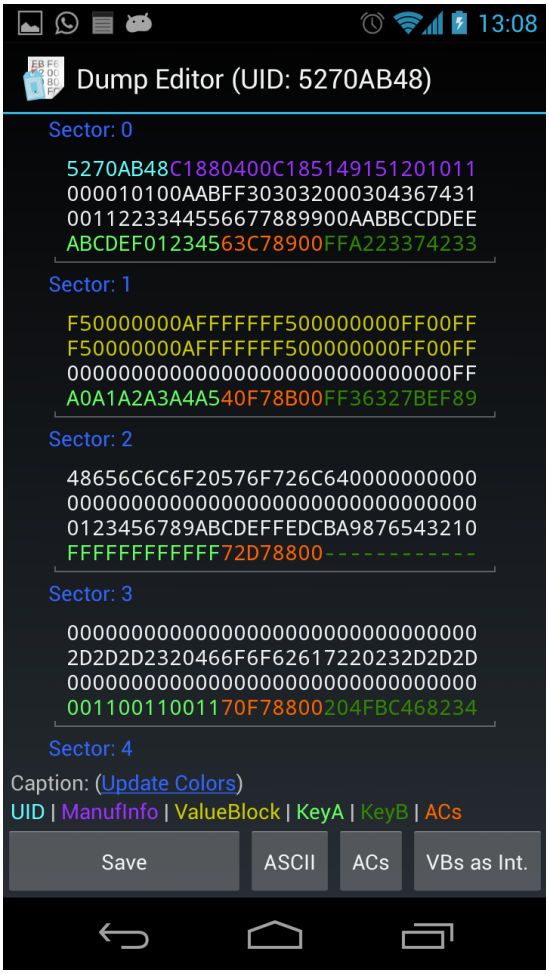

Abbildung 12: Screenshot der DumpEditorActivity

Damit der Benutzer gerade ausgelesene oder bereits abgespeicherte Daten von Mifare Classic-Tags betrachten und manipulieren kann, musste eine GUI (Activity) geschaffen werden, die diesen Aufgaben nachkommt. Die `DumpEditorActivity` erfüllt alle Anforderungen und bietet zusätzliche Möglichkeiten zur Analyse.

Zur besseren Lesbarkeit werden UID, die Herstellerinformationen, Value Blocks, Schlüssel A, Schlüssel B und die Access Conditions farbig hervorgehoben.

Die durch den Dump-Editor abgespeicherten Tags, befinden sich auf dem externen Speicher unter: `MifareClassicTool/dump-files/`.

Zur Analyse bietet der Dump-Editor drei weitere wichtige Funktionen, die in den folgenden Kapiteln vorgestellt werden.

5.3.1 Daten als ASCII darstellen

Abbildung 13: Screenshot der HexToAsciiActivity

Mifare Classic-Tags haben zwar im Vergleich mit anderen Speichermedien wenig Speicherplatz, dennoch ist es immer wieder üblich darauf abgespeicherte Daten in ASCII zu kodieren. Deswegen erweist es sich immer wieder als hilfreich, zur ersten Analyse die hexadezimalen Daten in einfach lesbaren ASCII-Text zu überführen. Die `HexToAsciiActivity` führt diese Aufgabe durch, wobei sie sich an US-ASCII mit 7 Bit hält und nicht sichtbare Zeichen (Steuerzeichen, Zeilenumbruch, usw.) durch „." ersetzt.

5.3.2 Mifare Classic Access Condtitions darstellen

Da die Mifare Classic Access Conditions aus beispielsweise „72D788" für das menschliche Auge nicht einfach abzulesen sind, sollte die App eine Darstellung ermöglichen in der leicht nachvollzogen werden kann, welche Speicherzugriffsoperationen mit welchen Schlüsseln erlaubt sind (siehe Rechte und Schlüssel).

Die `AccessConditionsActivity` interpretiert die Access Conditions nach den Tabellen 2 und Fehler: Referenz nicht gefunden des Kapitels Rechte und Schlüssel und stellt diese für Menschen leicht lesbar dar.

Abbildung 14: Screenshot der AcessConditionsActivity

5.3.3 Mifare Classic Value Blocks als Dezimalzahl darstellen

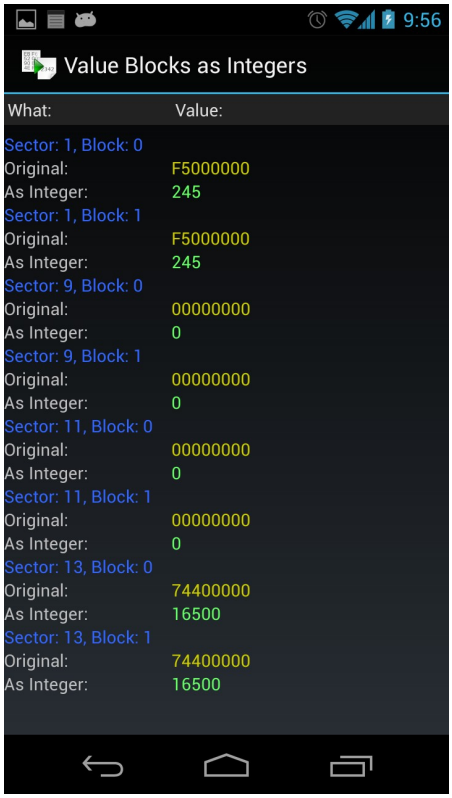

Abbildung 15: Screenshot der ValueBlockActivity

Value Blocks unterliegen dem selben Problem wie die Access Conditions: Sie sind für Menschen nicht einfach interpretierbar. Zwar sind Value Blocks einfach hexadezimal und little-endian codierte Zahlen (siehe Speicheraufbau), doch für ungeübte Benutzer ist es unbequem diese von „Hand" in eine dezimale Integer-Zahl umzurechnen. Genau dieser Umrechnungsaufgabe nimmt sich die `ValueBlockActivity` an. Sie erkennt Value Blocks an der speziellen Formatierung und führt diese in eine Integer-Zahl über.

5.4 Funktionalität 3: Mifare Classic-Tags beschreiben

Dritte Hauptfunktion und eines der größten Abgrenzungsmerkmale zu anderen RFID-Apps ist, dass „freie" Beschreiben von Mifare Classic-Tags. Üblicherweise bieten andere Apps die Möglichkeit Webseitenadressen, Kontakte, usw. im NDEF-Format auf RFID-Tags zu schreiben, jedoch nicht frei wählbare Daten (in hexadezimal) an beliebige Stellen zu schreiben. Um dem Ziel, eine hardwarenahe und universale App zu schaffen gerecht zu werden, wurden Funktionen implementiert, die das Beschreiben von Tags auf beliebige Art und Weise erlauben.

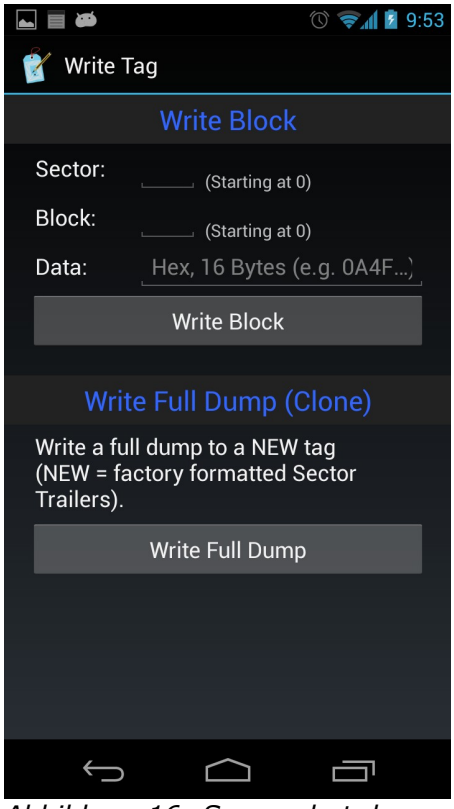

Abbildung 16: Screenshot der WriteTagActivity

Das „freie" Beschreiben führt allerdings bei unerfahrenen Benutzern möglicherweise zu Problemen. Wenn beispielsweise ungültige Access Conditions in den Sector Trailer geschrieben werden, wird damit der ganze Sektor unwiderruflich unbrauchbar. Damit dies nicht unbedacht passiert, wird vor dem Schreiben in einem Sector Trailer eine Warnung angezeigt.

Die App unterstützt zwei verschiedene Möglichkeiten einen RFID-Tag zu beschreiben. Diese werden in den folgenden Kapiteln vorgestellt.

5.4.1 Blockweise beschreiben

Beim blockweisen Schreiben kann genau ein Mifare Classic Block (16 Byte) an eine beliebige Stelle des Tags geschrieben werden. Voraussetzung ist selbstverständlich, dass die Stelle nicht in den Access Conditions als „read-only" markiert ist und dass der Schlüssel mit den Schreibrechten bekannt ist.

Wie auch beim Lesen wurde hier darauf geachtet, es dem Benutzer möglichst leicht zu machen. So muss er vor dem Schreibvorgang lediglich die Schlüsseldateien angeben, die möglicherweise einen passenden Schlüssel enthalten. Wie auch beim Lesen werden die zum Zielsektor gehörigen Schlüssel per *Dictionary Attack* ausfindig gemacht (siehe Das Key File Konzept). Falls beide Schlüssel (A und B) gefunden wurden, wird zuerst versucht mit dem Schlüssel B zu schreiben, da dieser in der Praxis häufig der Schlüssel mit Schreibrechten ist. Falls jedoch der Schreibvorgang fehlschlägt, wird versucht mit dem Schlüssel A zu schreiben.

5.4.2 Kompletten Dump auf neuen Tag schreiben

Beim Schreiben eines vollständigen Dumps geht es darum, einen Klon eines Mifare Classic-Tags zu erstellen. Erreicht werden kann das Ergebnis auch durch blockweises Schreiben (siehe Blockweise beschreiben), was jedoch für den Benutzer einen erheblichen Aufwand darstellt. Um dem Benutzer diese umständlichen einzelnen Aufgaben abzunehmen, wurde der Vorgang automatisiert. Dabei müssen folgende Fragen programmiertechnisch beantwortet werden:

- Ist der gewählte Dump vollständig? (Enthält der Dump alle Sektoren?)
- Haben Dump und Tag die gleiche Größe (Mifare Classic 1k oder 4k)?
- Ist der Tag leer/neu? (Sind die Sector Trailer so gesetzt, wie bei einem fabrikneuen Mifare Classic Tag?)

Wenn alle Fragen mit „Ja" beantwortet werden können, so schreibt die App automatisch den kompletten Dump auf den Tag.

Achtung: Der Block 0 des Sektors 0 in dem UID und Herstellerinformationen enthalten sind, kann nie geschrieben werden („read-only"). Das heißt dass der Klon nicht 100% korrekt ist, sondern sich in diesem Block unterscheidet. Sollte es notwendig sein ein „100%-Klon" zu erstellen, muss auf Werkzeuge wie den Proxmark3 zurückgegriffen werden. Er kann Mifare Classic-Tags inklusive der UID und Herstellerinformationen emulieren.

5.5 Funktionalität 4: Der Schlüsseldateien-Editor

Eine für die Veröffentlichung wichtige Abgrenzung war es, dass die App keine illegalen Funktionen enthält. Daraus, sowie auch aus anderen technischen Gründen folgt, dass die Anwendung selbst keine Schlüssel knacken kann, was wiederum heißt, dass die Schlüssel anderweitig auf das Android-Gerät kommen müssen.

Abbildung 17: Screenshot der KeyEditorActivity

Wenn die Schlüsselsammlung recht groß ist, empfiehlt es sich die Schlüssel einfach von einem Computer auf das Android-Gerät zu übertragen. Abgespeichert werden die Schlüsseldateien auf dem externen Speicher unter `MifareClassicTool/key-files/`.

Für Benutzer die kleine Mengen an Schlüsseln verwalten, oder für Nutzer die einzelne Schlüssel aus den Schlüsseldateien hinzufügen oder entfernen möchten, gibt es die `KeyEditorActivity`. Diese GUI ist ein kleiner Editor, der das Bearbeiten der Schlüsseldateien erlaubt. Zusätzlich können auch neue Schlüsseldateien erstellt und mit Schlüsseln gefüllt werden.

Da es nach dem Key-File Konzept genügt jeden Schlüssel nur einmal in einer Schlüsseldatei zu haben (siehe Das Key File Konzept), können in dem Editor per Knopfdruck Duplikate entfernt werden um Platz zu sparen.

Zeilen die mit „#“ **beginnen**, werden als Kommentar gewertet und nicht als Schlüssel interpretiert.

5.6 Funktionalität 5: In-App Hilfe und Informationen

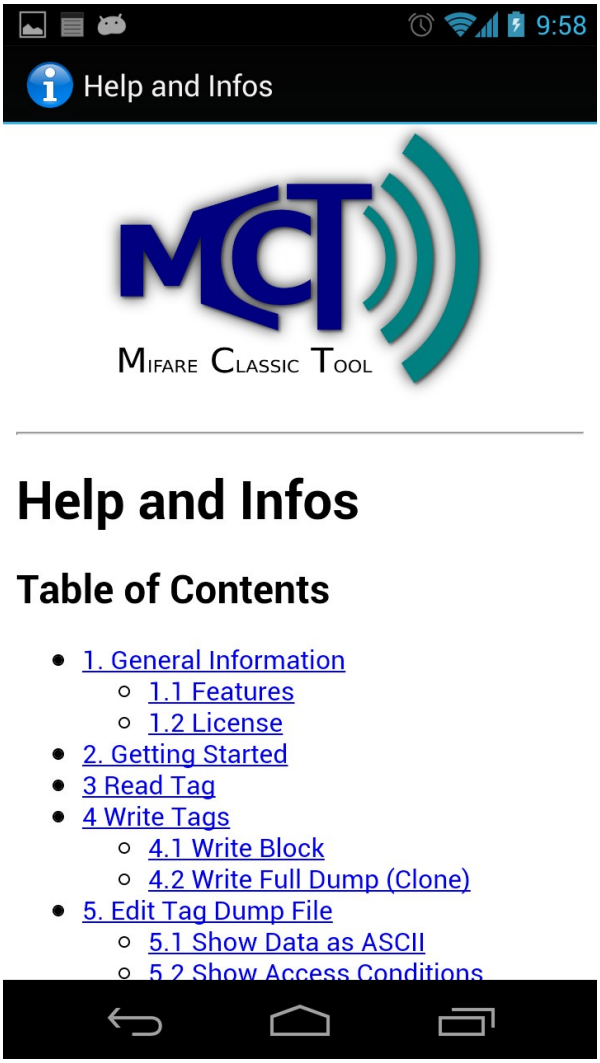

Abbildung 18: Screenshot der HelpActivity

Ein Problem mit dem die App sicherlich zu kämpfen hat ist, dass Benutzer die sich weniger gut mit Mifare Classic auskennen, möglicherweise schnell frustriert sind. Um dem vorzubeugen wurde in die App eine offline („In-App“) Hilfe integriert. Sie erklärt was getan werden muss, um die Anwendung sinnvoll zu benutzen („Getting Started“), wie welche Funktionen zu handhaben sind und wo weitere Informationen zu Mifare Classic-Technik zu finden sind. Außerdem enthält die `HelpActivity` Informationen über die verfügbaren Features, über die Lizenz und weitere allgemeine Hinweise.

6 Veröffentlichung

Das Unternehmen SySS GmbH stimmte freundlicherweise einer Veröffentlichung der App und deren Quellcode zu. Damit eine Veröffentlichung gelingt, sind mehrere Vorbereitungen nötig und Fragen zu beantworten.

6.1 Lizenz

Eine entscheidende Frage beim Veröffentlichen von Software ist die nach der Lizenz. Da der Quellcode mit veröffentlicht werden sollte, wurden die Unterschiede typischer Open Source Lizenzen betrachtet. In Frage kamen:

1. GNU General Public License, Version 3 (GPLv3)
2. BSD Lizenz
3. MIT Lizenz
4. zlib Lizenz

Die Lizenzen 2 bis 4 sind sich alle sehr ähnlich und auch für „Nicht-Juristen" meistens verständlich. Angenehm ist auch, dass sie die sogenannte „Werbeklausel" nicht enthalten, die abgeleitete Produkte verpflichtet den ursprünglichen Urheber zu nennen. Alle drei fallen in die Kategorie der „freizügigen Open-Source-Lizenzen".

Ein mögliches Szenario war jedoch ausschlaggebend, dass die (persönliche) Wahl auf die GPLv3 (1.) fiel. Abgeleitete Projekte von einem unter den Lizenzen 2 bis 4 stehen Projekt, können ihre Lizenz ändern, auf eine beispielsweise kommerzielle „Closed-Source" Lizenz. Um freie, offene Software zu fördern, erlaubt die GPL eine solche Lizenzänderung bei abgeleiteten und erweiterten Projekten nicht.

6.2 Veröffentlichungsweg

Entscheidendes Kriterium beim Veröffentlichungsweg war die Frage, welche Zielgruppe erreicht werden soll. Dazu wurden verschiedene Veröffentlichungsplattformen betrachtet und ihre Vor- und Nachteile untersucht.

- **Google Play Store**
  - Vorteile:
    - Auf fast allen modernen Android-Geräten enthalten
    - Einfache Installation
    - Integriertes Bewertungssystem

- Nachteile:
  - Kostenpflichtige Anmeldung für Entwickler
  - Bewertungen für fachspezifische Apps fallen durch frustrierte Benutzer, die sich mit der Materie nicht auskennen, häufig schlecht aus
  - Eine Veröffentlichung von Quellcode ist auf dieser Plattform nicht vorgesehen

- **Foren (fachbezogen auf RFID und Mifare Classic)**
  - Vorteile:
    - Die „richtige" Zielgruppe wird erreicht
    - Einfache Möglichkeiten für direktes Feedback
    - Einfache Möglichkeiten um sich mit anderen über Probleme, Features oder Anderes auszutauschen
  - Nachteile:
    - Viele verschiedene (RFID-) Foren machen es schwer, alle Interessenten dieses Gebietes zu erreichen

- **Github**
  - Vorteile:
    - Quellcode kann gut versioniert werden (durch Git-Repositories)
    - Quellcode kann einfach geteilt werden
    - Es kann einfach zusammen am Projekt gearbeitet werden (Social Coding)
    - Meilensteinverwaltung, „Issue-Tracker" und Wiki werden direkt mitgeliefert
  - Nachteile:
    - Binärdateien (Releases im APK-Format) können schlecht verwaltet werden
    - Eine „große", fachbezogene Zielgruppe ist nicht garantiert, da auf der Plattform sehr viele verschiedene Projekte zu finden sind

- **Eigene Webseite**
  - Vorteile:
    - Kann nach eigenen Wünschen gestaltet und eingerichtet werden
  - Nachteile
    - Die eigene Webseite unter der Zielgruppe bekannt zu machen ist nicht immer leicht

Durch die verschiedenen Vor- und Nachteile der unterschiedlichen Plattformen liegt die Idee nahe, mehrere dieser zu kombinieren. Für die Veröffentlichung der MCT-App wurden drei der oben genannten Möglichkeiten genutzt:

- **Forum**
  - Das Proxmark3 Forum wurde gewählt, da sich hier fachlich kompetente RFID-begeisterte Personen austauschen.
  - Link: http://www.proxmark.org/forum/viewtopic.php?id=1535
- **Github**
  - Github bietet durch zahlreiche Funktionen und eine aktive Benutzergemeinschaft Vorteile, die zu diesem Zeitpunkt keine andere „Code-Hosting"-Plattform bereitstellen konnte.
  - Link: https://github.com/ikarus23/MifareClassicTool
- **Eigene Webseite**
  - Da für einige Veröffentlichungen im Rahmen des Studiums oder auch privat bereits zuvor eine Webseite geschaffen wurde, konnte diese auf einfache Art genutzt werden, Dinge zu bieten die Foren und Github fehlen (z.B. APK-Datein, Screenshots und doxigen-Dokumentation bereitstellen).
  - Link: http://publications.icaria.de/

7 Aussichten

Durch das ausnahmslos positive Feedback nach der Veröffentlichung zusätzlich motiviert, wurde die persönliche Entscheidung gefällt, in der Freizeit die Anwendung weiter zu entwickeln. Bereits jetzt (01.03.2013) wird an einem Feature gearbeitet, welches auf einer Anregung aus dem Proxmark3 Forum basiert.

Auffällig ist, dass mehrere (chinesische) Blogs über die App berichtet haben, was dazu führte, dass die Anwendungen innerhalb von drei Wochen schon 300 mal heruntergeladen wurde, ohne aktiv „Werbung" zu machen.

Für die Zukunft wird eine zusätzliche Veröffentlichung im Google Play Store und im F-Droid Store in Betracht gezogen.