MifareClassicTool

Projektarbeit über die Weiterentwicklung einer Android-App

Gerhard Klostermeier

Hochschule für Technik und Wirtschaft Aalen

8. Oktober 2013

Kurzbeschreibung. Ein Bericht zur Projektarbeit über die Weiterentwicklung der Android-App „MifareClassicTool“. Der Fokus liegt dabei auf der Implementierung von neuen Funktionen. Des Weiteren werden wichtige Fehler und deren Behebung vorgestellt, sowie Statistiken die Aufschlüsse über Download- und Nutzerzahlen geben sollen.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen der Projektarbeit an der Hochschule für Technik und Wirtschaft in Aalen, wurde die „Open Source“-Software „MifareClassicTool“ (MCT) weiterentwickelt. Es handelt sich dabei um eine Android-Applikation (App), die zum Lesen, Schreiben und Analysieren von RFID-Transpondern mit Mifare Classic-Technik ist.

Im Verlauf dieser Ausarbeitung werden mehrfach das Proxmark-Forum, die eigene Homepage, der Eintrag im Google Play App-Store und weitere Webseiten genannt, welche eng mit der Veröffentlichung des MifareClassicTools zu tun haben. Die Links dazu sind dem Abschnitt 6 zu entnehmen.

1.1 Bisheriger Stand

Um diese Ausarbeitung besser verstehen zu können, kann es notwendig sein den beigelegten Praxissemesterbericht zu lesen. Er fasst die Entwicklung des MifareClassicTools bis zur Version 1.0.0 zusammen und behandelt unter anderem Grundlagen über Mifare Classic und die Android-App-Entwicklung, sowie einfache Begriffsdefinitionen und Fragen zur Lizenz oder zur Veröffentlichung.

1.2 Motivation

Die Veröffentlichung der Applikation im Rahmen des Praxissemesters brachte viel positives Feedback ein. Nutzer des Proxmark-Forums bedankten sich, schrieben produktive Kritiken und Wünsche nach neuen Möglichkeiten. Besonders nachdem in einigen (chinesischen) Blogs Artikel über das MifareClassicTool erschienen, häuften sich die Downloads.

Da nun klar war, dass trotz der kleinen Zielgruppe ein großes Interesse besteht, wurde beschlossen die Anwendung weiterzuentwickeln.

1.3 Ziel der Arbeit

Hauptziel ist es, neue Features zu entwickeln. Zusätzlich dazu sollten aber auch bestehende oder neu entstandene Probleme gelöst werden („Bugfixes“). Als weiteres Ziel wird die Veröffentlichung der Android-Anwendung in Google Play, einem sogenannten „App-Store“ angestrebt.

1.4 Problemstellung und Abgrenzung

Eines der grundlegenden Probleme ist die Frage danach, welche neuen Features implementiert werden sollen. Zur Auswahl stehen dabei Vorschläge von Nutzern (über das Proxmark Forum) und eigene Ideen.

Einige der Vorschläge sind genau richtig für die Umsetzung im Rahmen der Projektarbeit, andere etwas zu groß oder gar zu abwegig, sodass sie mit den Zielen des MifareClassicTools nicht mehr übereinstimmen.

Um die Vorschläge und Ideen etwas einzugrenzen sollen für diese Projektarbeit nur jene umgesetzt werden, welche der App dazu verhelfen ein einfaches Werkzeug mit grundlegenden Funktionen für Mifare Classic Transponder zu sein. Mit „grundlegenden" Funktionen sind vor allem Funktionen gemeint, die sehr nah an der Mifare Classic-Technik selbst sind, da die Applikation für Anwender gedacht ist, welche sich mit dieser Technik auskennen. Des Weiteren können selbstverständlich nur Features umgesetzt werden, die im Zeitrahmen der Projektarbeit liegen.

Spezifische Probleme die für jede Weiterentwicklung einzeln zu lösen sind, werden in der Planungsphase behandelt und gelöst (siehe Abschnitt 1.5).

1.5 Vorgehensweise

Features: Die einzelnen Features werden immer nach dem selben Muster umgesetzt:

- *Planung*: Anforderungen, Auswirkungen und Probleme analysieren und eine Strategie entwerfen, wie das Feature im vorhandenen Code sinnvoll umgesetzt werden kann.
- *Implementierung*: Die eigentliche Umsetzung in Code aber auch der Test der neuen Komponente.
- *Veröffentlichung*: Jedem neuen Feature soll eine Veröffentlichung („Release") folgen, in der die neue Version sowie der Quelltext und weitere Informationen für die Öffentlichkeit zugänglich gemacht werden.

Neue Features werden in Abschnitt 2 behandelt.

Bugfixes: „Altlasten" oder neu entstandene Probleme („Bugs") werden unabhängig der Weiterentwicklungen behandelt. Je nachdem wie dringlich ein Bug ist, muss die aktuelle Arbeit unterbrochen und eine Lösung ausgearbeitet werden. Bugs werden in Abschnitt 3 behandelt.

Versionsnummern: Um größere Veröffentlichungen mit neuen Features von den kleineren, die hauptsächlich nur Bugfixes enthalten zu unterscheiden, wird ein dreistelliges Versionierungssystem eingesetzt. Die erste Stelle markiert die Hauptversionsnummer. Sie wird nur dann hochgezählt, wenn das Programm eine neue „Reife" erreicht (was eher selten der Fall ist). Die zweite Stelle wird erhöht, wenn eine neue, größere Funktion (Feature) eingeflossen ist. Die letzte Stelle wird immer dann hochgezählt, wenn die neue Veröffentlichung „nur" kleinere Verbesserungen oder Bugfixes (z.B. für die Stabilität) mit sich bringt.

Ein Beispiel für eine solche Versionsnummer ist „1.3.2".

2 Entwicklung neuer Features

In den folgenden Abschnitten werden größere Features vorgestellt, die zwischen Version 1.0.0 (vor der Projektarbeit) und Version 1.5.2 (nach der Projektarbeit) umgesetzt wurden.

Die Abfolge mit der die neuen Funktionen vorgestellt werden, orientiert sich an dem Implementierungszeitraum. D.h. die Reihenfolge der Abschnitte entspricht der Reihenfolge der Implementierung.

2.1 Schreiben von Dumps auf bereits belegte Tags

Das MifareClassicTool konnte in Version 1.0.0 einen kompletten Dump lediglich auf einen Tag schreiben, dessen Formatierung der eines fabrikneuen Transponders entsprach.

Als Neuerung sollte nun das Schreiben auf beliebig belegte Tags hinzukommen unter, der Voraussetzung dass der Benutzer im Besitz der Schlüssel mit Schreibrechten ist.

Zunächst wurde begonnen diese neue Funktion aus eigenem Interesse umzusetzen. Kurze Zeit später kam jedoch zusätzlich die Bitte eines Nutzers, die Anwendung um genau solch eine Neuerung zu erweitern (https://github.com/ikarus23/MifareClassicTool/issues/2).

2.1.1 Planung

In der Planung stellte sich schnell heraus, dass dieses Feature nicht einfach umzusetzen ist und daher ein größerer Eingriff in den Code nötig sein wird.

Der Ablauf für den Benutzer sollte dabei so einfach wie möglich sein:

- Wahl eines gespeicherten Dumps.
- Wahl von einer oder mehreren Schlüsseldateien.
- Die Applikation schreibt den Dump auf den Tag.

Im Hintergrund muss jedoch viel passieren, damit der Vorgang reibungslos abläuft:

- Wahl des gespeicherten Dumps:
  - Die Wahl des Dumps kann mit der bereits vorhandenen `FileChooserActivity` durchgeführt werden.
- Wahl von einer oder mehreren Schlüsseldateien:
  - Die Wahl der Schlüsseldateien und die Zuordnung zu den Sektoren kann mit der bereits vorhandenen `CreateKeyMapActivity` durchgeführt werden.
- Die Applikation schreibt den Dump auf den Tag:

- Zunächst muss getestet werden, ob der gespeicherte Dump von der Größe her auf den vorliegenden Tag passt.
- Als nächstes ist eine Prüfung notwendig die sicherstellt, ob für die im Dump gespeicherten Sektoren die Schlüssel vorhanden sind.
- Des Weiteren muss geprüft werden, ob die Schlüssel Schreibrechte haben.
- Auch ist es möglich, dass einige Sektoren mit keinem Schlüssel beschrieben werden können („read-only“).
- Für jeden dieser Tests muss der Benutzer passende Fehlerausgaben erhalten, wenn ein solcher negativ ausgefallen ist.
- Zusätzlich soll der Anwender die Möglichkeit haben, trotz fehlender Schlüssel und read-only Sektoren, so viel wie möglich auf den Tag zu schreiben.
- Nachdem alle Sektoren die nicht geschrieben werden können aussortiert sind, sollen die Übrigen auf den Tag geschrieben werden.

2.1.2 Implementierung

In der Implementierungsphase entstanden für das Schreiben des Dumps drei wichtige neue Funktionen.

`createKeyMapForDump(...)`
Diese Methode liest den bereits zuvor gewählten Dump von dem externen Speicher des Android-Gerätes. Dabei werden alle Sektoren die unbekannte Daten enthalten (dargestellt durch „ - “) entfernt. Des Weiteren werden die Positionsinformationen (Sektornummer und Blocknummer) ermittelt und zusammen mit den gültigen Daten in einer Member-Variable (`mDumpWithPos`) festgehalten.

Um als nächstes die Schlüssel und deren Zuordnung zu den Sektoren (*Key Map*) für den vorliegenden Dump zu ermitteln, wird die `CreateKeyMapActivity` aufgerufen. Ist sie erfolgreich in der Erstellung einer Key Map, folgt darauf die Ausführung von `checkTag()`.

`checkTag()`
Diese Funktion stellte die größte Herausforderung dar. Sie muss alle Tests die in der Planungsphase ermittelt wurden implementieren und Sektoren die nicht geschrieben werden können aussortieren.

Der interne Ablauf ist prinzipiell in drei Teile gegliedert:

- *Teil I:* Hier wird getestet ob der vorliegende Tag groß genug für den Dump ist. Dazu vergleicht die Methode die höchste Sektornummer des Dumps mit der Anzahl der Sektoren auf dem Tag. Ist die Sektornummer kleiner oder gleich der Anzahl der Sektoren, passt der Dump auf den Tag.

- *Teil II:* Dieser Teil dient dazu herauszufinden, ob der Tag an den nötigen Stellen beschreibbar ist, ob Schlüssel für diese Stellen vorhanden sind und ob einer der Schlüssel die Schreibrechte hat.

  Dazu wurde im `MCReader`, der Klasse die für die direkte Kommunikation mit dem Tag verantwortlich ist, eine Funktion (`isWritableOnPositions(...)`) hinzugefügt. Sie enthält eine Liste von Positionen (Sektornummer, Blocknummer) sowie die Key Map, um damit die Zugriffsrechte des Tags (*Access Conditions*) zu lesen, zu interpretieren und zurückzugeben. Da die Auswertung von Zugriffsrechten bereits in der `AccessConditionsActivity` vorhanden war, wurde diese um Codeverdopplung zu vermeiden, überarbeitet und in die `Common`-Klasse ausgelagert.

- *Teil III:* Für jeden Block des Dumps ist es möglich, dass Sonderfälle auftreten. Dieser Abschnitt der Funktion ermittelt alle Komplikationen und erstellt einen Dialog der den Benutzer über die Besonderheiten informiert. Der Anwender hat dann die Möglichkeit den ganzen Vorgang abzubrechen, oder so viel wie möglich auf den Tag zu schreiben. Treten keine Sonderfälle auf wird direkt `wirteDump(...)` ausgeführt.

  Sonderfälle sind:

  - Read only: Block ist nicht beschreibbar.
  - Schlüssel unbekannt: Block ist mit Schlüssel A beschreibbar, welcher jedoch nicht in der Key Map ist.
  - Schlüssel unbekannt: Block ist mit Schlüssel B beschreibbar, welcher jedoch nicht in der Key Map ist.
  - Schlüssel unbekannt: Block ist mit Schlüssel A oder B beschreibbar, welche beide jedoch nicht in der Key Map sind.
  - Access Conditions read-only: Block (*Sector Trailer*) kann mit Schlüssel A beschrieben werden. Die Access Conditions bleiben dabei jedoch unverändert.
  - Access Conditions read-only: Block (*Sector Trailer*) kann mit Schlüssel B beschrieben werden. Die Access Conditions bleiben dabei jedoch unverändert.
  - Schlüssel unveränderbar: Block (*Sector Trailer*) kann mit Schlüssel B beschrieben werden. die Schlüssel (A und B) bleiben jedoch unverändert.
  - Kaputter Sektor: Die Zugriffsrechte des Sektors konnten nicht ausgewertet werden, da dieser Fehlerhaft ist.

  Blöcke die nicht beschreibbar sind werden von diesem Teil der Methode entfernt, sodass nur noch Blöcke übrig bleiben die mit hundertprozentiger Sicherheit auf den Tag geschrieben werden können. D.h. die Schlüssel mit den Schreibrechten sind bekannt und der Block ist nicht read-only.

`writeDump(...)`
Da nach der Ausführung von `checkTag()` diese Methode davon ausgehen kann, dass die übergebenen Daten sicher geschrieben werden können, nutzt sie einfach die `writeBlock(...)`-Methode der `MCReader`-Klasse, um die gültigen Abschnitte des Dumps auf den Tag zu schreiben. Während des Schreibvorgangs wird für den Benutzer ein „Bitte-Warten-Dialog“ angezeigt.

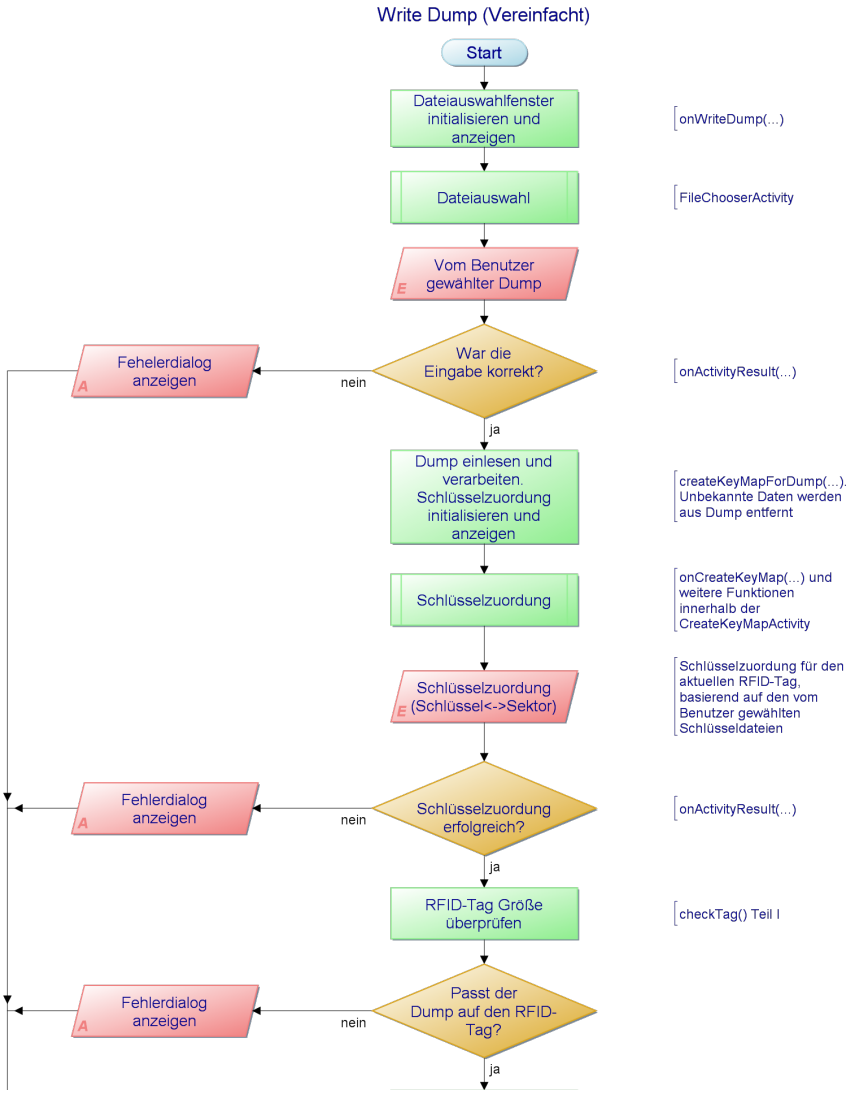


Abbildung 1: Programmablauf beim Schreiben eines Dumps (vereinfacht, Fortsetzung in Abbildung 2)

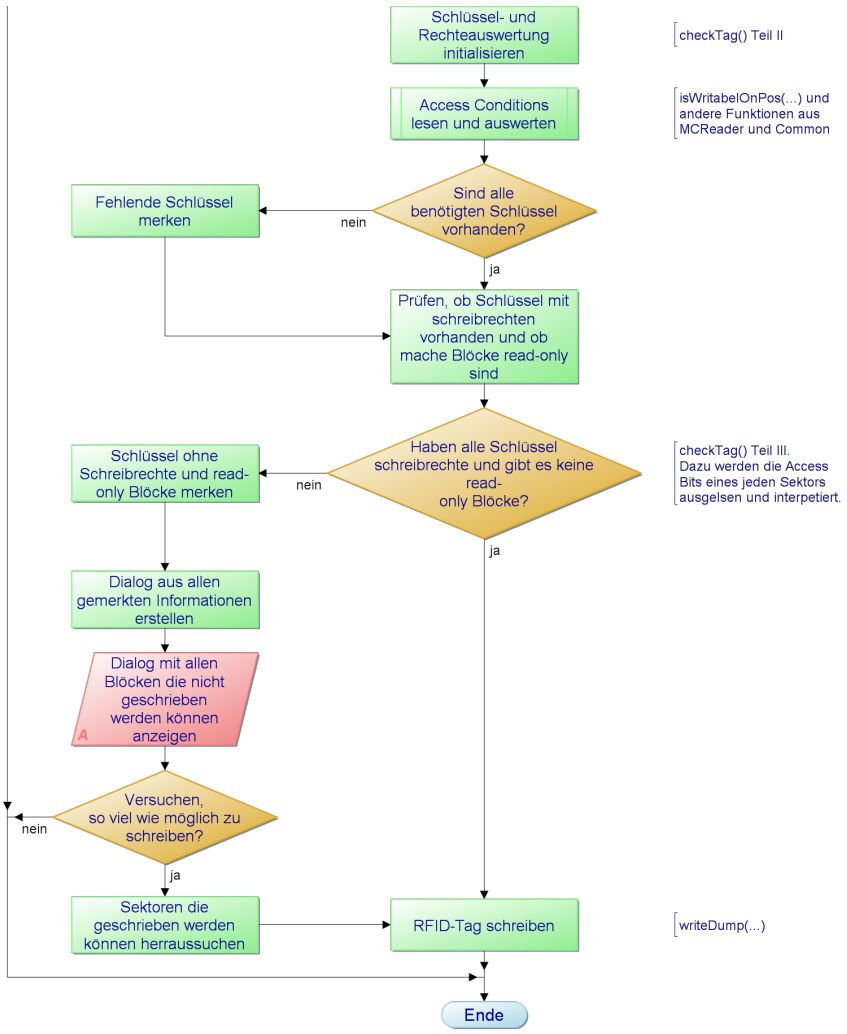


Abbildung 2: Fortsetzung der Abbildung 1: Programmablauf beim Schreiben eines Dumps (vereinfacht)

2.1.3 Veröffentlichung

Die neue Version des MifareClassicTools (Version 1.1.0) wurde über die eigene Webseite veröffentlicht, über die auch schon Version 1.0.0 herausgegeben wurde. Die Dokumentation wurde ebenfalls erneuert und publiziert. Der dazu passende Quelltext kann unabhängig von neuen Releases auf „github“, einer auf git-Versionierung basierende Webplattform eingesehen werden. Ein Veröffentlichungshinweis im Proxmark-Froum stieß abermals auf positives Feedback.

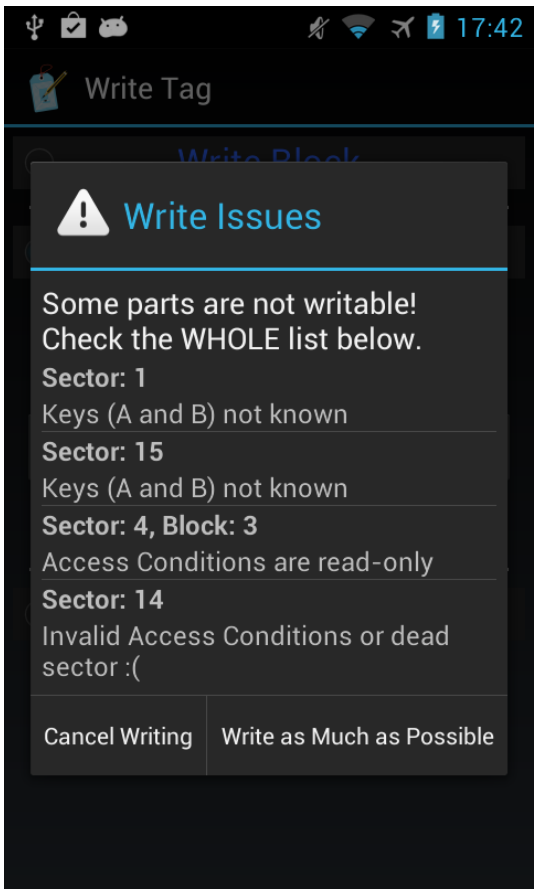


Abbildung 3: Screenshot: Informationen über Sonderfälle vor dem schreiben eines Dumps

2.2 Allgemeine Tag-Informationen anzeigen

Um sich einen Überblick über den vorliegenden Tag machen zu können, sollte die Applikation allgemeine Informationen auslesen und diese aufbereitet anzeigen.

2.2.1 Planung

Die Informationen die angezeigt werden sollten sind in zwei Bereiche zu unterteilen. Der erste umfasst Daten die von jedem Tag gewonnen werden können, der sich an den ISO 14443A Standard hält. Der andere Teil beinhaltet Informationen speziell zu Mifare Classic.

Die Informationen im einzelnen:

- Allgemeine Informationen:
  - UID (mit Länge in Byte)
  - Die RFID-Technik

  - ATQA (*Answer To Request acc. to ISO/IEC 14443-4*)
  - SAK (*Select Acknowledge*)
  - ATS (*Answer To Select*)
  - Tag-Typ und Hersteller
- Mifare Classic spezifische Informationen:
  - Speichergröße (in Byte)
  - Größe eines Blocks (in Byte)
  - Anzahl der Sektoren
  - Anzahl der Blöcke

Da sich dieses neue Werkzeug in keinen bereits vorhandenen Bereich der Anwendung eingliedern lässt, wurde beschlossen einen neuen Hauptmenüeintrag zu schaffen. Dieser sollte so gestaltet werden, dass später auch andere allgemeine Werkzeuge dort aufgenommen werden können.

Des Weiteren sollte mit dieser Neuerung auch die Probleme der Mifare Classic Support-Erkennung beseitigt werden. Genauere Informationen befinden sich im Abschnitt 3.1.

2.2.2 Implementierung

Für die Implementierung war lediglich die Problembehebung der Mifare Classic Support-Erkennung (siehe Abschnitt 3.1) und das Herausfinden des Tag-Typs, sowie des Herstellers von besonderer Bedeutung. Die anderen Informationen (UID, ATQA, SAK, etc.) sind direkt durch Androids NFC-System gegeben.

Um auf den Tag-Typ und Hersteller Rückschlüsse zu ziehen, werden die Werte von ATQA, SAK und ATS einer Tabelle zugeordnet. Eine solche Tabelle basiert auf Werten, welche in der Praxis gesammelt und von Herstellern in ihren Datenblättern angegeben wurden.

Konkret wurde sich an die Tabelle der nfc-tool.org Webseite gehalten: `http://nfc-tools.org/index.php?title=ISO14443A`. Diese wurde um ein paar weitere Werte ergänzt, die bei vorliegenden Karten ausgelesen wurden.

Um die Tabelle zu einem späteren Zeitpunkt ohne Aufwand vergrößern zu können, wurde diese als XML-Datei implementiert.

2.2.3 Veröffentlichung und Feedback

Nach der Veröffentlichung der Version 1.2.0, welche noch nicht Tag-Typ und Hersteller anzeigen konnte, folgte eine Anfrage über das Proxmark-Forum die Anzeige dieser Information auch noch zu implementieren. Da nötige Daten (ATQA, SAK und ATS) durch das neue Feature schon vorlagen, war die Implementierung dieser Anfrage naheliegend. Die „kleine“ Erweiterung wurde mit Version 1.2.1 veröffentlicht.

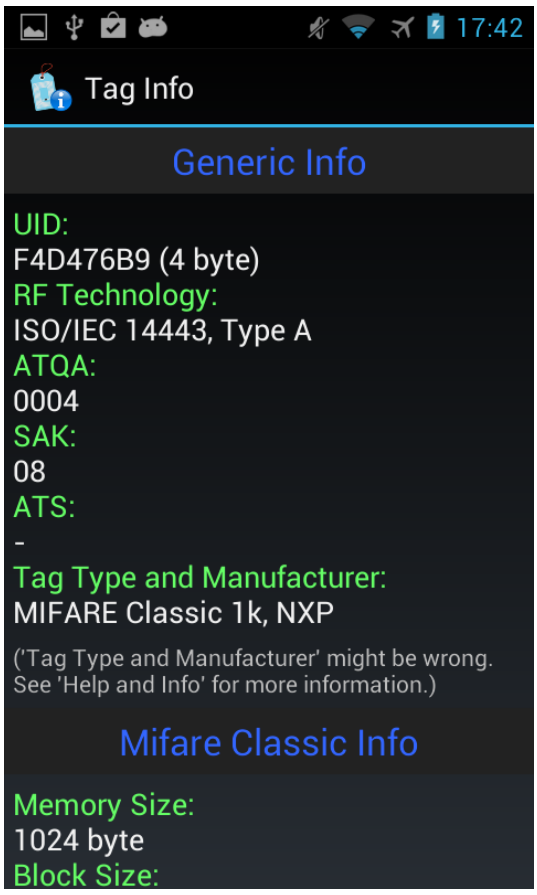


Abbildung 4: Screenshot: Tag-Informationen einer Mifare Classic 1k Karte

2.3 Value Block Decoder/Encoder

Value Blocks sind speziell formatierte Mifare Classic Blöcke, welche einen 4 Byte langen und vorzeichenbehafteten Integer darstellen. Ein solcher Block kann nicht nur gelesen oder geschrieben werden, sondern auch durch spezielle Kommandos (*increment*, *decrement*, *transfer*, *restore*) manipuliert werden. Weitere Informationen können dem Datenblatt von NXP entnommen werden: http://www.nxp.com/documents/data_sheet/MF1S503x.pdf (Kapitel: 8.6.2.1).

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	value				$\overline{\text{value}}$				value				adr	$\overline{\text{adr}}$	adr	$\overline{\text{adr}}$

Abbildung 5: Aufbau eines Mifare Classic Value Blocks

Für Nutzer ist es relativ aufwendig ein Integer in das Format eines Value Blocks umzuwandeln. Dies gilt natürlich auch für den umgekehrten Weg (Value Block als dezimale Ganzzahl darstellen). Vermutlich aus diesem Grund fragte

einer der Benutzer über das Proxmark-Forum, ob es möglich sei ein Tool zu implementieren, welches die De- bzw. Enkodierung von Value Blocks umsetzt.

2.3.1 Planung

Für dieses Feature war keine besondere Planung notwendig, da es sich um eine einfache (offline) Konvertierung von Daten handelt. Die Neuerung sollte in den mit Version 1.2.0 eingeführten Hauptmenüeintrag „Tools“ untergebracht werden.

2.3.2 Implementierung

Beide Konvertierungen (Dekodieren von Value Block nach Integer, Enkodieren von Integer nach Value Block) sind mit Funktionen möglich die Java bereits bietet.

- Dekodieren (Value Block nach Integer):
  Als erstes wird getestet ob es sich um einen korrekten Value Block handelt. D.h. die Formatierung muss mit der in Abbildung 5 dargestellten übereinstimmen.
  Der zweite Schritt besteht in der Konvertierung der Darstellung, was wiederum eine Konvertierung der Typen mit sich zieht.

```
// data = Value Block als Zeichenkette.
// data.substring(0, 8) = 4 Byte Value extrahieren;
// Common.hexStringToByteArray(...) = Hex-Zeichenkette
//   zu Byte-Array umwandeln.
byte[] vbAsBytes = Common.hexStringToByteArray(data.substring(0, 8));

// ByteBuffer.wrap(vbAsBytes).getInt() = Byte-Array
//   zu Integer umwandeln.
// Integer.reverseBytes(...) = Byte-Reihenfolge
//   eines Integers umkehren.
int vbAsInt = Integer.reverseBytes(
    ByteBuffer.wrap(vbAsBytes).getInt());

// "" + int = Integer in Zeichenkette umwandeln.
mVBasInt.setText("" + vbAsInt);
```

- Enkodieren (Integer nach Value Block):
  Beim Enkodieren wird ähnlich vorgegangen. Wichtig ist nur, dass die bitweise Invertierung auf den Integer angewandt wird, bevor dieser zu einem Array bzw. String konvertiert wird.

```
// vbAsInt = Integer Wert des Value Blocks.
// ~ = Bitweise Invertierung.
// Integer.reverseBytes(...) = Byte-Reihenfolge
//   eines Integers umkehren.
// ByteBuffer.allocate(4).putInt(...).array() =
//   Integer in Byte-Array umwandeln.
// Common.byte2HexString(...) = Byte-Array in
//   Hex-Zeichenkette umwandeln.

String vb = Common.byte2HexString(ByteBuffer.allocate(4).putInt(
    Integer.reverseBytes(vbAsInt)).array());
String vbInverted = Common.byte2HexString(ByteBuffer.allocate(4)
    .putInt(Integer.reverseBytes(~vbAsInt)).array());
```

Da das Addr. Feld (siehe Abbildung 5) direkt vom Benutzer in hexadezimal angegeben wird, ist lediglich eine Invertierung von Nöten um das Value Block-Format einzuhalten.

2.3.3 Veröffentlichung

Parallel zu diesem Feature wurde eine weitere Anfrage bearbeitet, welche nach einer Möglichkeit fragte, Dumps der App in ein Rohdatenformat zu konvertieren und umgekehrt. Mit der Veröffentlichung von Version 1.3.0 erhielt das neue Value Block-Werkzeug Einzug in die MCT-App und die gewünschten Konvertierungsskripte wurden über die github-Seite freigegeben.

Die Version 1.3.0 war auch die erste Version die über Google Play heruntergeladen werden konnte. Alle weiteren Releases wurden über Google Play, aber auch über die eigene Seite veröffentlicht.

Andere „kleinere“ Neuerungen die zwischen Version 1.3.0 und Version 1.3.3 umgesetzt wurden sind:

- Dumps über Mail, Bluetooth, etc. teilen.
  (Anfrage aus dem Proxmark-Forum.)
- Dateien über den Dateiauswahldialog löschen.
  (Inspiriert durch einen Kommentar, welcher unter einem Blogeintrag über die Applikation zu finden war.)

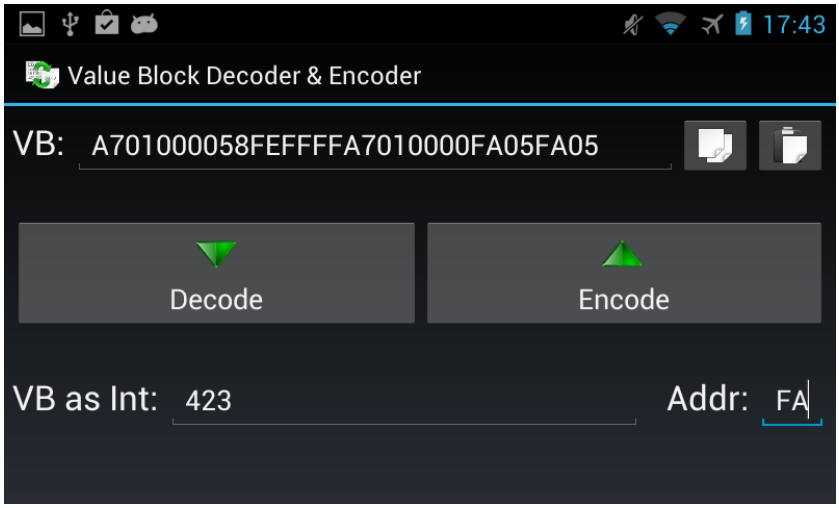


Abbildung 6: Screenshot: Konvertierung von Value Blocks von und in Ganzzahlwerte

2.4 Auslieferungszustand wiederherstellen und Manufacturer Block schreiben

Wie bereits für größere Veröffentlichungen üblich, war die Umsetzung eines aufwendigeren Features geplant: Die Möglichkeit Tags zurück in den Auslieferungszustand zu setzen. In der Entwicklungsphase dieser Neuerung reichte jedoch ein (schwedischer) Nutzer einen Erweiterungsvorschlag samt passenden Quellcode ein (https://github.com/ikarus23/MifareClassicTool/pull/4). Dies führte dazu, dass die Version 1.4.0 zwei größere Änderungen mit sich brachte.

2.4.1 Planung

Obwohl der eingereichte Quellcode funktionstüchtig war, mussten einige, hauptsächlich optische Dinge angepasst werden. Deshalb wurde die Planung für beide Features unabhängig durchgeführt.

- **Auslieferungszustand wiederherstellen:**
  Durch die Möglichkeit beliebige Dumps auf einen Tag zu schreiben (siehe Abschnitt 2.1), ist die Implementierung dieser Weiterentwicklung ohne allzu großen Aufwand möglich.

  Bei einem fabrikneuen Tag sind alle Datenblöcke „0“. In den Sector Trailern haben beide Schlüssel (A und B) den Wert „FFFFFFFFFFFF“ und die Access Conditions sind auf „FF078069“ gesetzt. Der letzte Block eines Tags unterscheidet sich darin, dass dort die Access Conditions den Wert „FF0780BC“ haben.

  ```
  // Beispiel: Sektor im Auslieferungszustand
  00000000000000000000000000000000
  00000000000000000000000000000000
  00000000000000000000000000000000
  FFFFFFFFFFFFFF078069FFFFFFFFFFFF
  ```

  Um einen Tag in den Auslieferungszustand zurück zu setzen muss lediglich ein virtueller Dump erstellt werden, welcher von der Größe dem vorliegenden Tag entspricht und dessen Daten nach den oben genannten Regeln aufgebaut sind. Wurde ein solcher Dump (virtuell) erstellt, kann er wie jeder andere auf einen Tag geschrieben werden (siehe dazu auch Abschnitt 2.1)

- **Manufacturer Block schreiben:**
  Der Manufacturer Block ist der erste Block im ersten Sektor. Die führenden 4 Byte (bzw. 7 Byte) stellen die UID dar, während die anderen Bytes für herstellerspezifische Informationen genutzt werden.

  Besonderheit ist, dass dieser Block unabhängig der gesetzten Rechte (Access Conditions) nicht geschrieben werden kann. Dies verhindert, dass ein vollständiger Klon einer Karte erstellt werden kann, da sich Tag und Klon immer in Sektor 0, Block 0 unterscheiden werden.

Andere (chinesische) Hersteller die nicht NXP unterliegen, vertreiben (vermutlich illegal) über das Internet Mifare Classic Tags, bei denen auch Veränderungen an UID oder des gesamten Manufacturer Blocks vorgenommen werden können.

Von diesen Tags gibt es zwei unterschiedliche Typen:

- *Typ 1*:
  Dieser allgemein bekannte Typ lässt das Verändern des Manufacturer Blocks durch eine bestimmte Kommandofolge zu. Diese Kommandos können jedoch nicht durch eine einfach App von einem klassischen Android-Gerät ausgestrahlt werden. Dies macht es für das MifareClassicTool unmöglich, die Sonderfunktionen eines solchen Tags wahrzunehmen.
- *Typ 2*:
  Bei diesem Typ ist es möglich, den Manufacturer Block mit einem normalen Mifare Classic „write"-Kommando zu beschreiben. Er unterscheidet sich damit nicht mehr von den anderen Datenböcken auf einem Tag. Die Möglichkeit den Block 0 des Sektors 0 zu schreiben ist damit trivial.

  Die bis zu diesem Zeitpunkt fehlende Implementierung ist darauf zurück zuführen, dass dieser Typ mir persönlich nicht bekannt war. Durch den eingereichten Quellcode war schnell klar, dass es nach dem ersten einen weiteren Typen geben muss, der das normale „write"-Kommando für den Manufacturer Block akzeptierte. Ein kurzes Gespräch mit dem Nutzer bestätigte dies. Dieser ist in Besitz solcher Karten und war so freundlich einen der Tags kostenfrei zur Verfügung zu stellen.

  Weitere Nachforschungen ergaben, dass dieser Typ noch nicht sehr alt ist und dadurch auch allgemein wesentlich unbekannter.

Da beim Schreiben des Manufacturer Blocks für original NXP Mifare Classic Tags ein Fehler auftritt, sollte der „Write Dump"-Funktion (siehe Abschnitt 2.1) eine passende Option hinzugefügt werden. Außerdem sollten erklärende Informationen über diese Option angezeigt werden können, da sonst bei Anfängern Missverständnisse entstehen könnten.

2.4.2 Implementierung

Die Implementierung des Features zum Zurücksetzen eines Tags in den Auslieferungszustand wurde wie in der Planung verdeutlicht durch das Erstellen eines virtuellen Tags erreicht. Der mit `createFactoryFormatedDump()` erstellte Dump wird wieder in der Member-Variable `mDumpWithPos` abgelegt, um dann `checkTag()` aufzurufen (vergleiche Abschnitt 2.1.2).

Auch den eingereichten Quelltext in den vorliegenden einfließen zu lassen stellte kein Problem dar. Lediglich ein paar kleine, vor allem optische Änderungen an Code und Oberfläche waren nötig, um das Feature (Manufacturer Block schreiben) vollständig zu machen.

2.4.3 Veröffentlichung

Trotz der ausführlichen Hinweise bei der Veröffentlichung der Version 1.4.0 kam es häufig zu Missverständnissen. Vielen Nutzern war es nicht klar, dass es zwei verschiedene Typen der Spezialtags gibt, die das Schreiben von UID und Herstellerinformationen zulassen. Da die Sonderfunktionen der weiter verbreiteten Typ 1-Karten nicht mit dem MifareClassicTool angesprochen werden können, wurde öfters bemängelt, dass das neue Feature nicht funktionieren würde.

Durch etwas Aufklärungsarbeit über das Proxmark-Forum und über die github-Projektseite konnten jedoch die Missverständnisse ausgeräumt werden.

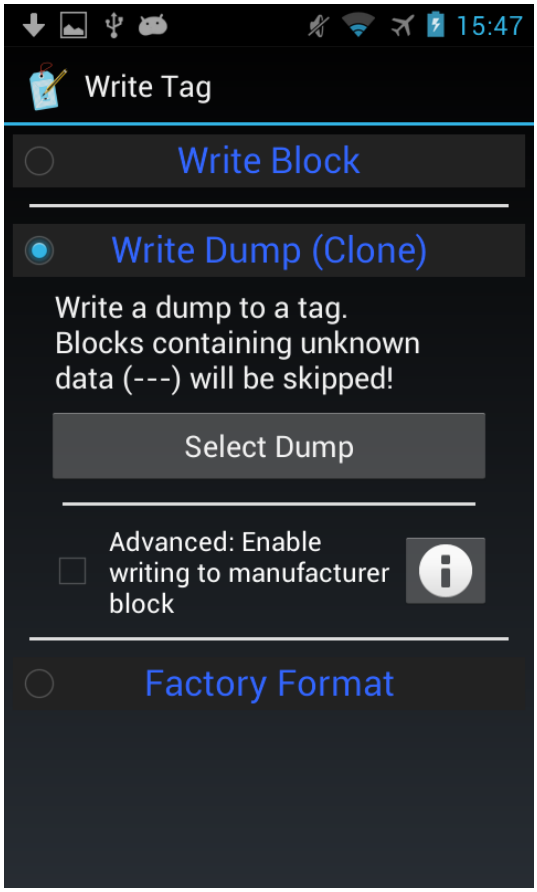


Abbildung 7: Screenshot: Option zum Schreiben des Manufacturer Blocks. (Unten: Funktion um Tags in den Auslieferungszustand zurück zu setzen.)

2.5 Dump mit angepassten Rechten schreiben

Häufig haben Tags eine Konfiguration, welche das Ändern von Datenblöcken, vor allem aber von Sector Trailer nicht mehr erlaubt (read-only). Wird ein solcher Dump auf eine frabrikneue Karte geschrieben kann diese zu einem späteren Zeitpunk möglicherweise nicht mehr für andere Zwecke eingesetzt werden, da nicht alle Daten änderbar sind.

Des öfteren werden jedoch die gesetzten Rechte in den Access Conditions nicht durch das Lesegerät überprüft. Dadurch ist es möglich eigene Rechte zu vergeben, die es möglich machen den Tag später wieder vollständig verändern zu können.

Beispiel:

- Rechte des Originaltags: `72D788`

Tabelle 1: Auflösung der Access Condtions 72D788

Was?	Lesen	Schreiben
Block 0	Key A|B	Key B
Block 1	Key A|B	Key A|B
Block 2	Key A|B	Key B
Key A	Niemals	Niemals
Key B	Niemals	Niemals
AC Bits	Key A|B	Niemals

Wie zu sehen ist, kann der Sector Trailer (Key A, Key B und die AC Bits) nie mehr verändert werden.

- Angepasste Rechte: `7A5788`

Tabelle 2: Auflösung der Access Condtions 7A5788

Was?	Lesen	Schreiben
Block 0	Key A|B	Key B
Block 1	Key A|B	Key A|B
Block 2	Key A|B	Key B
Key A	Niemals	Key B
Key B	Niemals	Key B
AC Bits	Key A|B	Key B

Die Rechte sind beinahe identisch zu den Originalen, sodass das Lesegerät ungehindert mit dem Tag arbeiten kann. Einziger Unterschied ist, dass der Sector Trailer jederzeit mit Schlüssel B beliebig neu geschrieben werden kann.

Einer Weiterverwendung des Tags für andere Zwecke steht somit nichts im Weg.

2.5.1 Planung

Um den Benutzern die Möglichkeit zu geben die Access Conditions vor dem Schreiben eines Dumps zu ändern, sollte der „Write Dump"-Funktion ein weiteres Feature als Option hinzugefügt werden.

Diese sollte dafür sorgen, dass beim Einlesen des Dumps gleich alle Access Conditions mit denen vom Nutzer gewählten ersetzt werden.

Um diese Option (zusätzlich zur „Manufacturer Block Schreiben"-Option, siehe Abschnitt 2.4) im Layout unterzubringen, musste außerdem die Oberfläche der „Write Dump"-Funktion überarbeitet werden.

2.5.2 Implementierung

Um diese Neuerung umzusetzen wurde direkt in die vorhandene `createKeyMapForDump(...)`-Methode eingegriffen (siehe Abschnitt 2.1.2). Da die Funktion unter anderem auch den Dump vom externen Speicher des Android-Geräts liest, wurde sie zunächst umbenannt: `readDumpAndCreateKeyMapForDump(...)`.

Der eigentliche Austausch der Access Conditions wird direkt nach dem Lesen des Dumps durchgeführt. Dazu wird der jeweils letzte Block eines Sektors (Sector Trailer, Block 3 oder Block 15) ermittelt und die Rechte mit den vom Benutzer gewählten ausgetauscht (3 Byte, Byte 7-9).

2.5.3 Veröffentlichung

Für die Veröffentlichung der Version 1.5.0 gab es wieder positives Feedback. Nutzer freuten sich über die Neuerung, da damit Zeit und vor allem auch Geld gespart werden kann (Wiederverwendung von Tags). Besonders für Anfänger ist diese Option zu empfehlen, um nicht ausversehen beim „Experimentieren" den Tag durch permanente Rechte für weitere Tests unbrauchbar zu machen.

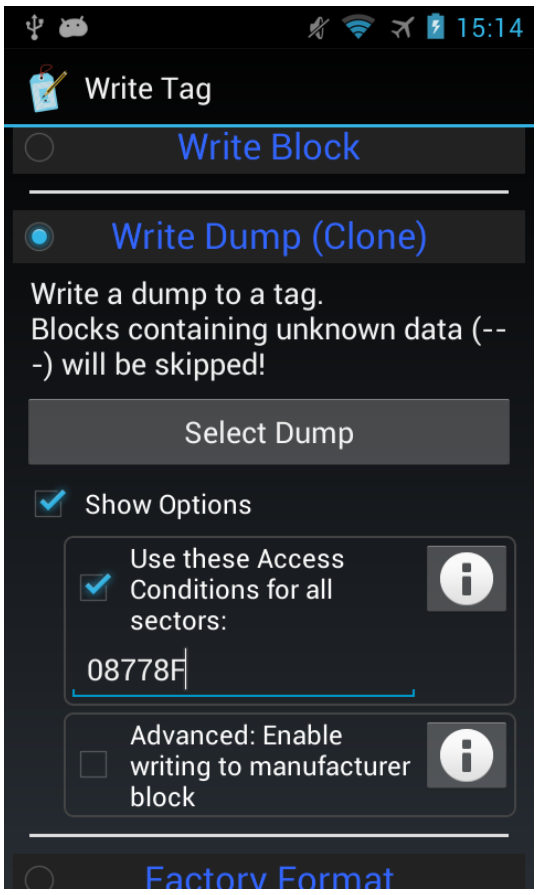


Abbildung 8: Screenshot: Option um die Access Conditions des Dumps durch eigene zu ersetzen

3 Wichtige Bugfixes

Im Laufe der Weiterentwicklung von Version 1.0.0 zu Version 1.5.2 wurden nicht nur neue Features eingeführt, sonder auch grundlegende Probleme behoben. In den folgenden Abschnitten werden drei der wichtigsten Bugs vorgestellt, deren Ursachen erläutert und die Implementierung ihrer Lösung aufgezeigt.

3.1 Mifare Classic-Support Check

Da nicht jedes NFC-fähige Android-Gerät die Mifare Classic-Technik unterstützen muss, testet die MCT-App beim Start ob die Technik zur Verfügung steht. Für diesen Zweck definiert die „Android Compatibility Device Description“ (ACDD) ein „System Feature“ welches vom Hersteller gesetzt werden muss, falls das Gerät Mifare Classic unterstützt.

```
if (getPackageManager().hasSystemFeature("com.nxp.mifare") == false)
{
   // Kein Mifare Classic-Support!
   ...
}
```

Problem ist, dass diese „Flag“ erst in den ACDD 4.1 Einzug gehalten hat und dass manche Hersteller vergessen das System Feature zu setzten.

Ein Beispiel dafür war das Smartphone „HTC X One“ eines Benutzers. Er beklagte sich darüber, dass die Applikation nicht starten würde weil sein Handy keine Mifare Classic-Unterstützung habe. Andere Apps (z.B. „NFC Tag Info“) arbeiten jedoch korrekt mit Mifare Classic-Tags (https://github.com/ikarus23/MifareClassicTool/issues/3). Wie sich herausstellte, hat HTC das `com.nxp.mifare`-System Feature nicht gesetzt.

Nach einigen Nachforschungen war jedoch klar, dass es keine andere Testmöglichkeit zur Startzeit gibt. Ein weiterer Weg herauszufinden ob Android einen Mifare Classic-Tag bearbeiten kann, ist die `TechList` des aktuell vorliegenden Tag zu überprüfen.

```
if(Arrays.asList(tag.getTechList()).contains(
    "android.nfc.tech.MifareClassic") == false)
{
   // RFID-Tag ODER Hardware hat kein Mifare Classic-Support!
   ...
}
```

Dieser Weg hat jedoch Nachteile:

- Falls `tag.getTechList()` nicht `android.nfc.tech.MifareClassic` enthält gibt es zwei mögliche Ursachen: Das Gerät hat keine Mifare Classic-Unterstützung *oder* der Tag ist kein Mifare Classic-Tag!
- Der Test kann nicht zum Startzeitpunk durchgeführt werden, sondern erst wenn ein Tag erkannt wurde.

Da bis zu diesem Zeitpunk keine weitere, bessere Möglichkeit bekannt war, wurde versucht mit diesem Test eine Lösung für das Problem zu entwickeln.

Die Änderungen für eine Lösung wurden wie folgend implementiert:

- Der Test zur Startzeit entfällt, da er „false positives“ liefert.
- Einführung eines neuen Features um allgemeine Tag-Informationen darzustellen (siehe Abschnitt 2.2).
- Bei Erkennung eines neuen Tags wird geprüft, ob `tag.getTechList()` in `android.nfc.tech.MifareClassic` enthalten ist.
- Falls keine Unterstützung festgestellt wurde, wird das Werkzeug für Tag-Informationen angezeigt (falls es nicht schon im Vordergrund ist), welches grundlegende Informationen (UID, ATQA, SAK, etc.) anzeigt und einen deutlichen Hinweis gibt, dass es keinen Mifare Classic-Support gibt.

- Durch einen Knopfdruck können weitere Informationen zu dem Fehler angezeigt werden. Hier wird der Benutzer darüber aufgeklärt, dass der vorliegende Transponder kein Mifare Classic-Tag ist (wahrscheinlich), oder dass das Gerät keine Mifare Classic-Unterstützung hat (eher unwahrscheinlich).

Bekante Geräte ohne Mifare Classic-Support sind:

- Google Nexus 4
- Google Nexus 7 (2013)
- Google Nexus 10
- Samsung Galaxy S4

Abbildung 9: Screenshot: Fehlermeldung falls Tag oder Gerät nicht Mifare Classic unterstützen

3.2 Sprachunabhängige Dumps

Um die Möglichkeit zu bewahren die Anwendung später einfach in neue Sprachen übersetzten zu können, wurde das von Android vorgesehene Konzept für Sprachunabhängigkeit eingehalten. Dieses sieht vor, alle Zeichenketten die ein Benutzer zu Gesicht bekommen kann in deine XML-Datei auszulagern.
Aufbau der `strings.xml`:

```
...
<!-- Allgemein: -->
<string name="RESOURCE-ID">Text in Sprache X</string>

<!-- Beispiele: -->
<string name="text_key_map_progress">Key Mapping Progress:</string>
<string name="text_sector">Sector</string>
<string name="text_block">Block</string>
...
```

Im Quelltext wird über den Namen (`RESOURCE-ID`) die anzuzeigende Zeichenkette referenziert.

```
...
// Allgemein:
String text = getString(R.string.RESOURCE-ID);

// Beispiel:
TextView tv = new TextView(this);
tv.setText(getString(R.string.text_sector));
...
```

Um verschiedene Sprachen zu unterstützen können dem Projekt weitere XML-Dateien hinzugefügt werden, welche für den selben Zeichenkettennamen unterschiedliche Übersetzungen liefern.
Die Projektstruktur muss dafür wie folgt aussehen:

```
MyProject/
|-res/
  |-values/
  | |-strings.xml
  |
  |-values-es/
  | |-strings.xml
  |
  |-values-fr/
    |-strings.xml
```

Das Problem welches es zu lösen galt war, dass durch Unachtsamkeit die Sektor-Informationen der Dumps abhängig von der Sprache (`strings.xml`) waren.

```
# Auszug aus einem Dump
...
+Sector: 12 <-- Sektor-Information
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
...
```

Wenn die MifareClassicTool-App in z.B. Deutsch und Englisch übersetzt wäre, hätten die Dumps der beiden Sprachen unterschiedlich ausgesehen (`+Sector: 12` oder `+Sektor: 12`). Da Dumps jedoch für die Verarbeitung mit Maschinen gedacht sind, sollten diese immer identisch aussehen.

Die Lösung für das Problem war trivial (aber wichtig). Die Sektor-Informationen wurden fest in den Quelltext geschrieben, sodass kein Zugriff auf auf eine XML-Datei mit sprachabhängigen Zeichenketten durchgeführt wird.

3.3 Relevante Daten vor dem Start neuer Activities sichern

Ein (französischer) Nutzer nahm über E-Mail Kontakt auf und berichtete davon, dass die Applikation bei Nutzung der „Write Dump"-Funktion mit einer „`NullPointerException`" beim Zugriff auf `mDumpWithPos` abstürzen würde.

Besonders interessant war dieses Problem, da der Benutzer bis dahin der einzige war der diesen Fehler auslösen konnte und die Funktion schon seit vielen Versionen im Code war und es keine offensichtliche Fehlimplementierung gab.

Nach längerer Suche wurde klar, dass die Ursache im Ablauf der „Write Dump"-Funktion selbst liegen muss:

1. Der Nutzer wählt den Hauptmenüeintrag „Write Tag".
2. Die `WriteTagActivity` wird gestartet.
3. Benutzer wählt die Funktion „Write Dump".
4. Die `FileChooserActivity` wird gestartet.
5. Der Nutzer wählt den Dump den er schreiben möchte.
6. Die `WriteTagActivity` wird wieder in den Vordergrund geholt, um den gewählten Dump in `mDumpWithPos` einzulesen.
7. Die `CreateKeyMapActivity` wird gestartet.
8. Der Benutzer wählt Schlüsseldateien und die Activity erstellt die Key Map.
9. Die `WriteTagActivity` wird wieder in den Vordergrund geholt, um mit der Key Map den Tag zu überprüfen und `mDumpWithPos` (den zuvor eingelesenen Dump) zu schreiben.

Genau im letzten Schritt (9.) tritt der Fehler auf. Es wird auf `mDumpWithPos` zugegriffen, obwohl dieser `null` ist. Wie in Schritt 6. aber zu sehen ist, wurde `mDumpWithPos` mit Daten initialisiert.

Nach weiterer Recherche wurde das Problem deutlich: Das Android-Betriebssystem kann nach belieben Activities die nicht im Vordergrund sind freigeben, um Speicher zu gewinnen.

Dem Ablauf zufolge initialisiert die App mit Schritt 6. `mDumpWithPos`. Nach dem Start der `CreateKeyMapActivity` (Schritt 7.) wird jedoch die `WriteTagActivity` freigegeben, was zur Folge hat, dass auch `mDumpWithPos` verloren geht. In Schritt 9. wird dann die `WriteTagActivity` neu erstellt – ohne `mDumpWithPos`.

Als das Problem erst einmal gefunden war, konnte die Implementierung der Lösung einfach umgesetzt werden. Bevor die `WriteTagActivity` von Android freigegeben wird, speichert sie die Daten der Member-Variable `mDumpWithPos`. Passend dazu wird bei der Neuerstellung der Activity darauf geachtet, ob gespeicherte Daten vorhanden sind. Falls das der Fall ist, wird `mDumpWithPos` mit diesen Daten initialisiert.

In Zusammenarbeit mit dem Nutzer konnte überprüft werden, ob das Problem erfolgreich beseitigt wurde.

4 Statistiken und Feedback

Im Verlauf der Entwicklung und der Veröffentlichung neuer Versionen konnten einiger Nutzer gewonnen werden. Die folgenden Statistiken sollen genaueren Aufschluss darüber geben, von wie vielen diese App genutzt wird. Des Weiteren könnte das Feedback vielleicht auch die Frage nach dem „Warum?“ klären.

4.1 Veröffentlichung auf eigener Webseite

Die Anwendung wurde von Version 1.0.0 bis zur Version 1.2.1 ausschließlich über eine eigene Webseite verteilt. Um potenziell interessierte Anwender zu erreichen, wurde für jedes Release ein Eintrag im Proxmark-Forum verfasst. Diese Praxis wurde auch später (Version 1.3.0 - aktuell) parallel zur Veröffentlichung über Google Play (siehe Abschnitt 4.2) beibehalten.

4.1.1 Statistiken

Die folgende Statistik gibt lediglich Aufschluss über die Zugriffszahlen. Die Anzahl der Nutzer kann aus ihr nur grob geschätzt werden.

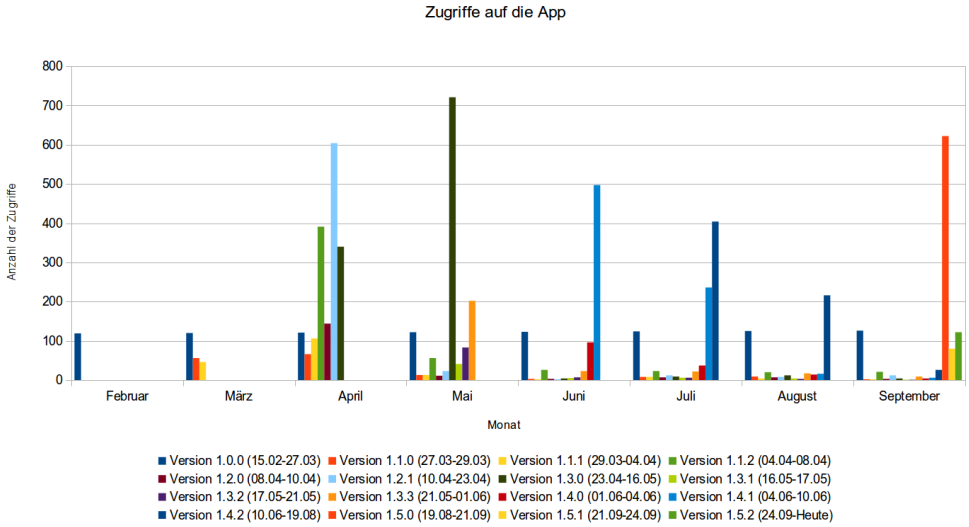


Abbildung 10: Anzahl der Zugriffe auf die einzelnen App-Versionen (15.02.2013 - 28.09.2013)

4.1.2 Feedback

Zitate aus dem Proxmark-Forum („ikarus“ ist das Entwickler-Pseudonym):

- „Looks good ikarus! Open source too. Two thumbs up“
- „Great good job!“
- „Thanks a lot!! Good job!!!! [...]“
- „Hi ikarus and thank you really much for your software ! [...]“
- „I also want to say big THANKS to You, ikarus“
- „so nothing more to say except: it's great“
- „This tool is awesome, I love it! [...]“
- „Very nice app. Thanks for making it available!! [...]“
- „First i will thank you for all you effort. [...]“

Referenzen auf das MifareClassicTool:

- YouTube-Video in dem der Geldbetrag auf einer Nahverkehrskarte mit der MCT-App manipuliert wird:
  http://www.youtube.com/watch?v=0iMK_gKXEKU
- Chinesischer Blogeintrag über das MifareClassicTool:
  http://www.freebuf.com/tools/7531.html

- Chinesischer Blogeintrag in dem erläutert wird, wie unter anderem mit der MCT-App der Geldbetrag einer Karte für die Wasserversorgung manipuliert werden kann.
  http://www.hackdig.com/?05/hack-3408.htm

4.2 Veröffentlichung über Google Play

Von Version 1.3.0 an wurde die MCT-App auch über Google Play verbreitet. Da viele Android-Geräte diesen App-Store verwenden, wurden einige neue Nutzer erreicht.

Ein weiterer Vorteil der dadurch entstand ist, dass Benutzer der Anwendung durch einen einfachen „Klick“ einen Absturzbericht senden können. So konnten Stabilitätsprobleme schneller gefunden und behoben werden.

4.2.1 Statistiken

Die folgenden Statistiken sind exakt, da Google Play die Installationen von Apps über den Store genau überwacht.

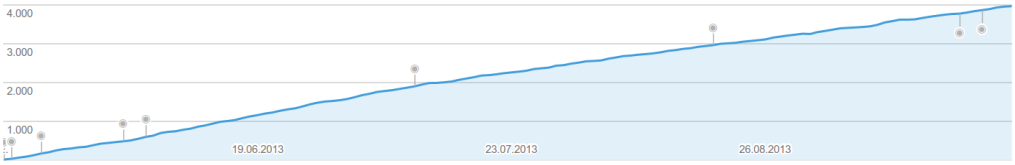
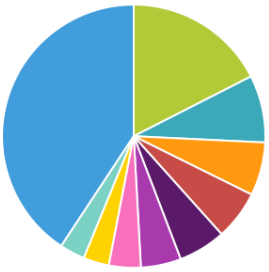
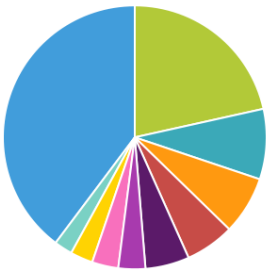


Abbildung 11: Anzahl der aktiven Geräte, auf denen die App momentan installiert ist (16.05.2013 - 28.09.2013). Neue Releases sind durch einen grauen Punkt markiert.



Polen	694	17,49 %
Deutschland	328	8,27 %
Frankreich	260	6,55 %
Niederlande	240	6,05 %
Vereinigte Staaten	231	5,82 %
Vereinigtes Königreich	196	4,94 %
Schweden	155	3,91 %
Russische Föderation	125	3,15 %
Spanien	123	3,10 %
Weitere	1.615	40,71 %

Abbildung 12: Anzahl der aktiven Geräte, auf denen die App momentan installiert ist nach Ländern aufgelöst (16.05.2013 - 28.09.2013)



Samsung Galaxy S III (m0)	854	21,53 %
Google Nexus 7 (grouper)	342	8,62 %
Samsung Galaxy Note II ...	283	7,13 %
Samsung Galaxy Nexus ...	239	6,02 %
HTC HTC One (m7)	214	5,39 %
HTC HTC One X (endea...	131	3,30 %
Sony Ericsson Xperia Z ...	128	3,23 %
Samsung Galaxy S III LT...	110	2,77 %
Sony Ericsson LT26i (L...	89	2,24 %
Weitere	1.577	39,75 %

Abbildung 13: Anzahl der aktiven Geräte, auf denen die App momentan installiert ist nach Geräte-Typ aufgelöst (16.05.2013 - 28.09.2013)

4.2.2 Feedback

Bewertung auf Google Play:

- „Finally A long awaited good tool. Thanks!“
- „Unvergleichlich super“ (übersetzt aus dem Russischen)
- „Simply awesome! Awesome tool with useful functions that work really well!“
- „Worked brilliant on Nexus 7“
- „Sehr gut. Derzeit eine der besten Anwendungen zu diesem Thema in diesem Store“ (übersetzt aus dem Spanischen)

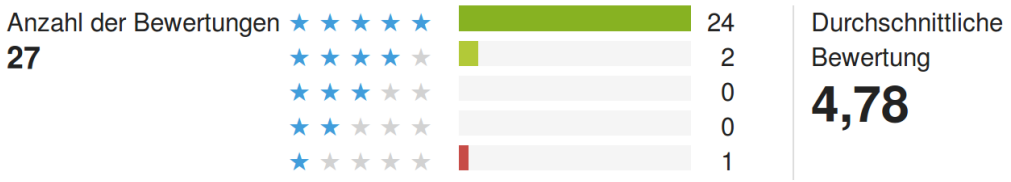


Abbildung 14: Bewertung in Google Play (Stand: 08.10.2013)

Die schlechte Bewertung von einem Stern ist auf ein Missverständnis zurück zuführen. Der Benutzer war frustriert da er glaubte, die App würde nicht funktionieren. Problem war aber, dass er einen Transponder mit anderer Technik nutzte (nicht Mifare Classic).

5 Fazit und Ausblick

Die Umsetzung neuer Features und das Beheben von schwerwiegenden Bugs, aber auch zahlreiche kleine Verbesserungen und Problembehebungen, brachten die Applikation über den Zeitraum der Projektarbeit ein großes Stück voran. Dass diese Weiterentwicklung gern gesehen und gewürdigt wird, ist an dem durchweg positiven Feedback und den steigenden Nutzerzahlen zu erkennen.

In der Zukunft soll auch weiterhin an der Android-App gearbeitet werden. Es gibt noch zahlreiche Ideen und Anfragen von Nutzer die Anwendung durch neue Funktionen zu verbessern. Einige davon sind auf der öffentlichen „ToDo“-Liste einzusehen:
https://github.com/ikarus23/MifareClassicTool/blob/master/TODO.txt .

6 MifareClassicTool im Internet

- MifareClassicTool auf Google Play (Android App-Store):
  https://play.google.com/store/apps/details?id=de.syss.MifareClassicTool
- Projektseite auf github.com (auf git-basis versionierter Quelltext):
  https://github.com/ikarus23/MifareClassicTool
- Eigene Seite (APK-Datein, Dokumentation, etc.):
  http://publications.icaria.de/mct/
- Forumsthread zu MifareClassicTool auf proxmark.org:
  http://www.proxmark.org/forum/viewtopic.php?id=1535
- Alle Änderungen im Detail (github.com):
  https://github.com/ikarus23/MifareClassicTool/commits/master