RFID Workshop

by Vanhoecke Vinnie

Date: 2024-08-10

1. Table of content

2. Introduction

Welcome to the theory bundle of the Playing with RFID workshop. Here you can find some basic information about RFID and the protocols touched upon in the Workshop. This can be useful to look back at some information while performing the exercises, especially the Mifare Classic Memory Layout section.

Section 4 also contains documentation on installing the tools on the different operating systems.

3. OctoBox

OctoBox will be the device used during the training to simulate the applications and RFID systems we are trying to compromise. It's essentially a Raspberry Pi with a RF component attached and some custom written software in Python using Kivy to display the interfaces:

Components such as:

- 7" screen
- High Frequency reader (RC522)
- WiFi USB
- Raspberry Pi

3.1. Wifi Access Point

The machine will be broadcasting a WiFi access point:

OctoBox

PSK: RFIDWorkshop

This WiFi access point will be used for the following:

- File Server:
  http://files.rfidwork.shop
- Score Board (when enabled)
  http://score.rfidwork.shop

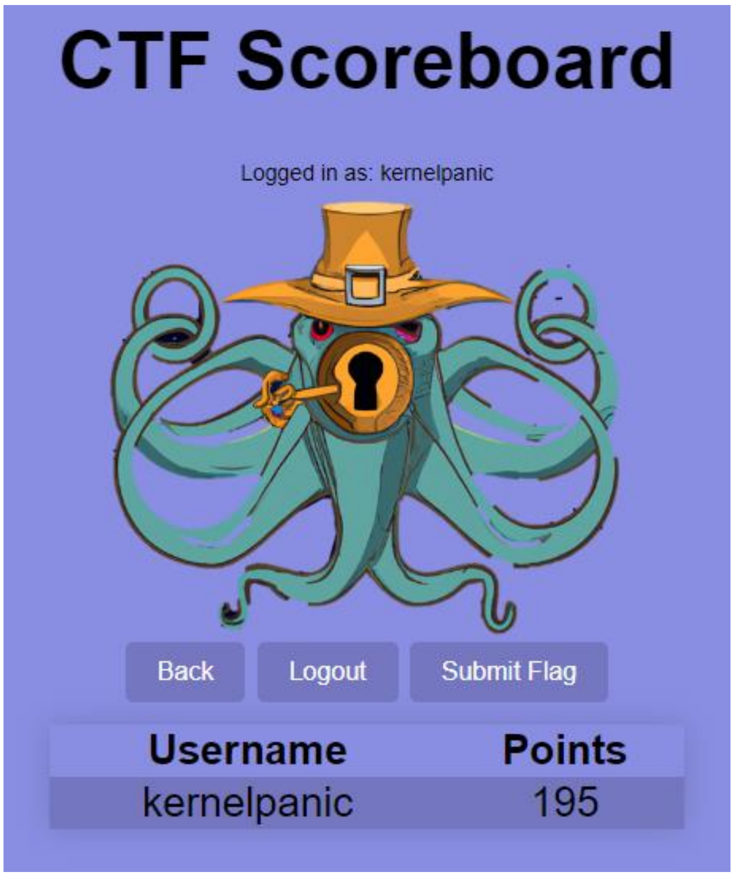
Most of the functionalities available on the OctoBox Access Point make use of DNS resolution. So please validate if VPN or other DNS configurations are not preventing you from accessing these.

3.2. Scoreboard

When connected to the WiFi access point of the workshop you should be able browse to the following link:

http://score.rfidwork.shop

As soon as workshop participants performed the username challenge, usernames should be showing up on the scoreboard on the board automatically, but its advised to register for extra points.


CTF Scoreboard

Logged in as: kernelpanic

Username	Points
kernelpanic	195

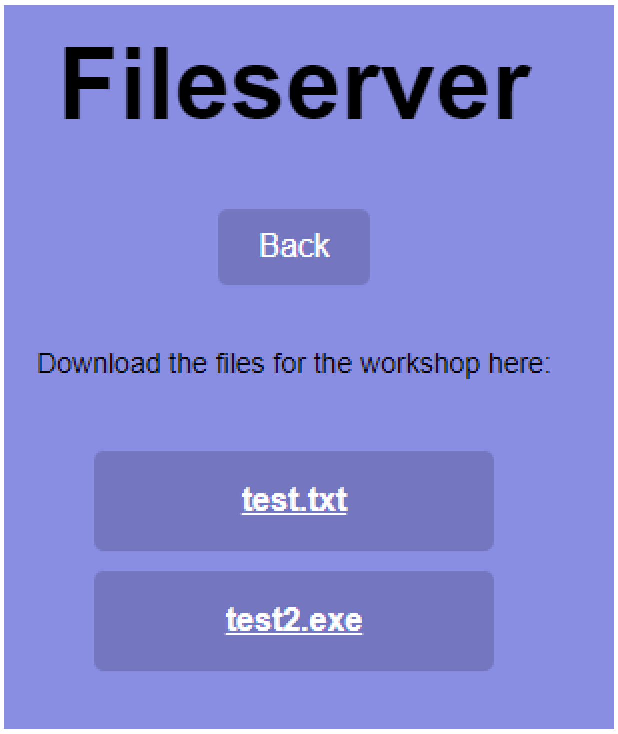
Scoreboard web interface

3.3. Fileserver

http://files.rfidwork.shop

It contains the following files:

- Workshop PDF materials
- Latest ProxSpace.zip
- Latest Proxmark3
- ISO standard
- MIFARE Docs
- Flipper software and documentation



Fileserver

Back

Download the files for the workshop here:

test.txt

test2.exe

Fileserver web interface

4. Tool setup/installation

The workshop provides detailed guidelines for three main devices, which is the proxmark3, Flipper Zero and ACR122U. If you do not have an RFID device, you can use an ACR122U from the instructor provided during the workshop. The challenges can be done with other RFID reader/writers as well.

Keep in mind that these instructions are NOT tested within Virtual Machine's (VM's) and VM's are not suggested since almost always issues occur when working inside VM's to perform the exercises.

4.1. Proxmark3

The Proxmark3 can be called the Swiss-army tool for RFID Research. The code base is mostly written in C and is fully open source and relatively easy to use. The tool is easy to use for people who have some experience with embedded programming, electronics, and Linux. Without these it might not be the tool for you.

One of key advantages of the Proxmark is its community support and active maintenance with frequent updates, which is awesome for being an open source tool. It's easy to extend and learn from the work that others have done. I used it to create fuzzing libraries and exploits for EMV and the Proxmark3 is just the tool to use for this.

Please read the installation instructions on how to setup the device here below depending if you are on Windows or Linux.

4.2. Flipper

Introduced in 2020 and received a lot of attention as it's a very useful pocketsize device. It has a Low Frequency and High Frequency reader built-in and great community support. It might not have the same specification and capabilities as the Proxmark, but it can also do much more than RFID protocols alone.

A little less user friendly in some activities such as manipulating data on RFID cards but perfect for other simpler tasks.

4.3. ACR 122U

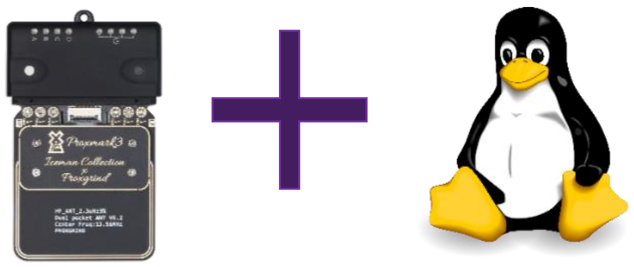
It's a cheap NFC reader/writer of around 20 - 50 euro, works well with several NFC libraries. It does not have the same capabilities as the Proxmark and Flipper, but you can still compromise most of the RFID systems in the world with just this tool.

For this reader I created a custom Python tool you can use to do the courses. Obviously, you are free to use any other tools, but the walkthroughs will be written for that tool.

https://github.com/VinnieV/crid

There are a couple more tools that are required, such as mfoc, mfcuk and milazycracker.

For more instructions on how to setup the tools and device please refer to the installation page for ACR122U.



Proxmark3 Linux Installation

The documentation to flash and build the Proxmark client for Linux is documented here:

https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/Linux-Installation-Instructions.md

Installing the software on Linux should be the easiest way to interact with the Proxmark. Follow these steps below:

1. Perform package update:

```
sudo apt-get update
```

2. Install following dependencies:

```
sudo apt-get install --no-install-recommends git ca-certificates build-essential pkg-config \
libreadline-dev gcc-arm-none-eabi libnewlib-dev qtbase5-dev \
libbz2-dev liblz4-dev libbluetooth-dev libpython3-dev libssl-dev
```

3. Clone the repository:

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

4. Build the code:

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

Building might take some time and hopefully no errors occurred that required additional troubleshooting. Once successful, it created the firmware binaries, ready to be flashed on the Proxmark, by following these steps:

⚠ <u>Debugging Tip:</u> Check if the USB cable is not charge-only type cable.

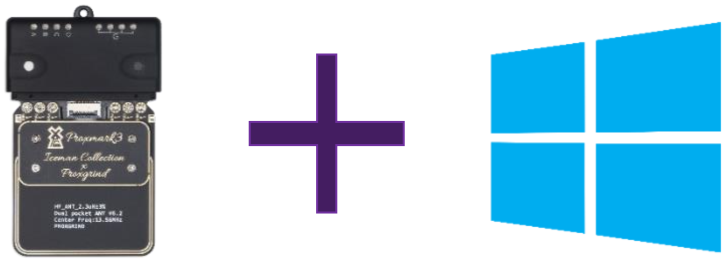
1. Connect your Proxmark device to your Windows computer.
2. Once connected, you can run the flash binary to install the bootloader and firmware on the Proxmark:

```
./pm3-flash-all
```

3. Once that's completed you should be good to go to enter the Proxmark shell by typing in:

```
./pm3
```

This Proxmark3 shell will provide all the power to perform the exercises.



Proxmark3 Windows Installation

The documentation to flash and build the Proxmark client for Windows is documented here:

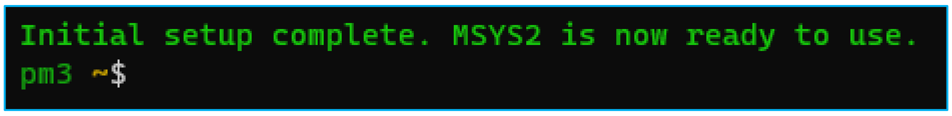
https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/Windows-Installation-Instructions.md

Feel free to follow that documentation but I have broken down the generic steps here. The first step would be to download and install the ProxSpace Tool. This is a Linux subsystem within windows where we will have all the tools and dependencies installed to build and execute the Proxmark project. To install ProxSpace perform the following:

1. Download the ProxSpace ZIP file, available here: https://github.com/Gator96100/ProxSpace/releases
2. Extract The ProxSpace ZIP file to a location path without spaces.
3. Execute the bat file within the ProxSpace folder:

```
.\runme64.bat
```

It might take some time the first time to install everything but that should give you a new command prompt starting with pm3:



```
Initial setup complete. MSYS2 is now ready to use.
pm3 ~$
```

This shell provides you with the code building environment to build the Proxmark repository for Windows. To perform this, please follow these steps:

1. Clone the Proxmark Repository:

```
cd
git clone https://github.com/RfidResearchGroup/proxmark3.git
cd proxmark3
```

2. Then build the code:

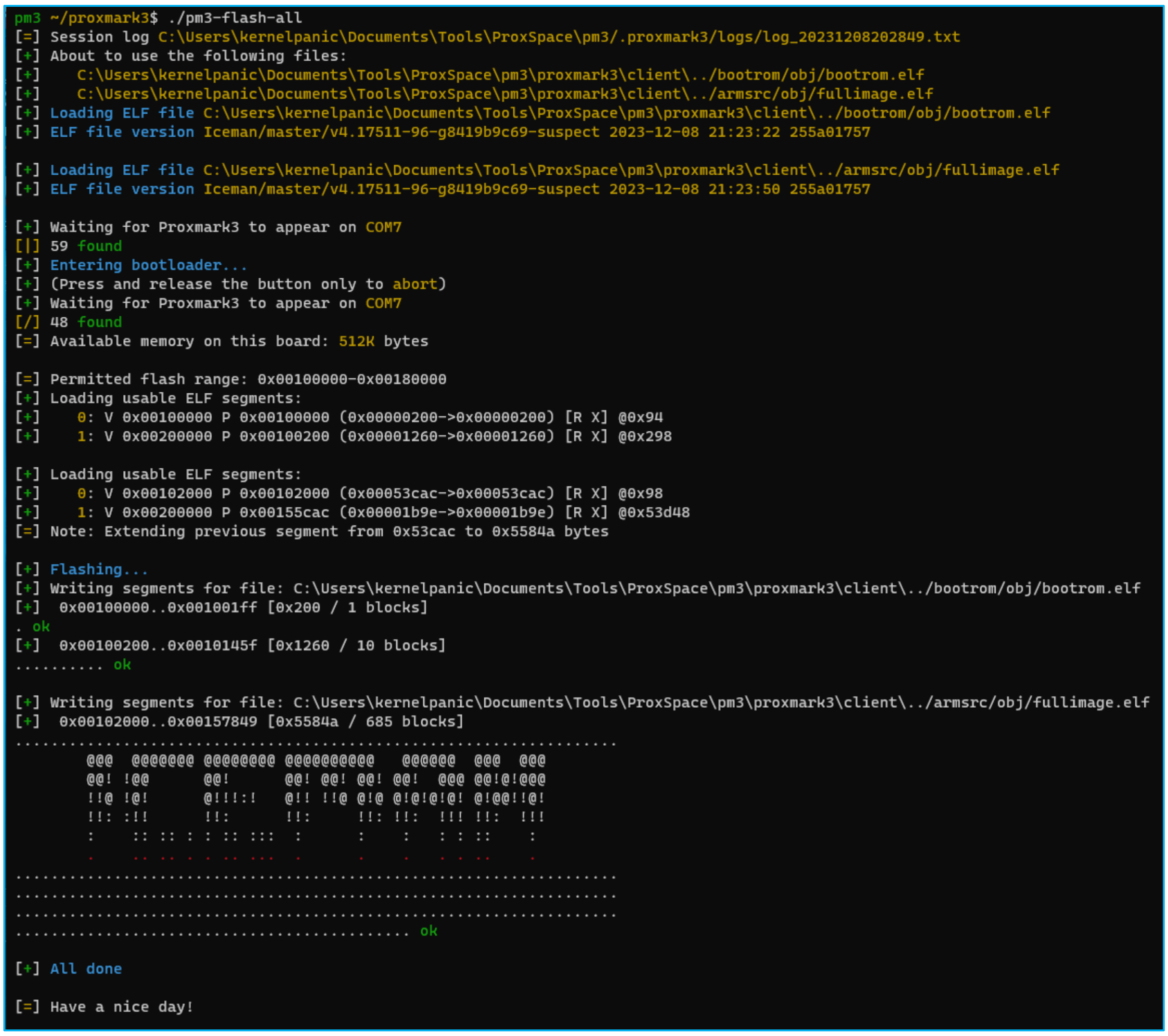
```
make clean && make all
```

Building might take some time and hopefully no errors occurred that required additional troubleshooting. Once successful, it created the firmware binaries, ready to be flashed on the Proxmark, by following these steps:

⚠ Debugging Tip: Check if the USB cable is not charge-only type cable.

1. Connect your Proxmark device to your Windows computer.
2. Once connected, you can run the flash binary to install the bootloader and firmware on the Proxmark:

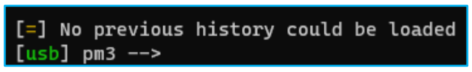
```
./pm3-flash-all
```



```
pm3 ~/proxmark3$ ./pm3-flash-all
[=] Session log C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3/.proxmark3/logs/log_20231208202849.txt
[+] About to use the following files:
[+]    C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../bootrom/obj/bootrom.elf
[+]    C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../armsrc/obj/fullimage.elf
[+] Loading ELF file C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../bootrom/obj/bootrom.elf
[+] ELF file version Iceman/master/v4.17511-96-g8419b9c69-suspect 2023-12-08 21:23:22 255a01757

[+] Loading ELF file C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../armsrc/obj/fullimage.elf
[+] ELF file version Iceman/master/v4.17511-96-g8419b9c69-suspect 2023-12-08 21:23:50 255a01757

[+] Waiting for Proxmark3 to appear on COM7
[|] 59 found
[+] Entering bootloader...
[+] (Press and release the button only to abort)
[+] Waiting for Proxmark3 to appear on COM7
[/] 48 found
[=] Available memory on this board: 512K bytes

[=] Permitted flash range: 0x00100000-0x00180000
[+] Loading usable ELF segments:
[+]    0: V 0x00100000 P 0x00100000 (0x00000200->0x00000200) [R X] @0x94
[+]    1: V 0x00200000 P 0x00100200 (0x00001260->0x00001260) [R X] @0x298

[+] Loading usable ELF segments:
[+]    0: V 0x00102000 P 0x00102000 (0x00053cac->0x00053cac) [R X] @0x98
[+]    1: V 0x00200000 P 0x00155cac (0x00001b9e->0x00001b9e) [R X] @0x53d48
[=] Note: Extending previous segment from 0x53cac to 0x5584a bytes

[+] Flashing...
[+] Writing segments for file: C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../bootrom/obj/bootrom.elf
[+]  0x00100000..0x001001ff [0x200 / 1 blocks]
. ok
[+]  0x00100200..0x0010145f [0x1260 / 10 blocks]
.......... ok

[+] Writing segments for file: C:\Users\kernelpanic\Documents\Tools\ProxSpace\pm3\proxmark3\client\../armsrc/obj/fullimage.elf
[+]  0x00102000..0x00157849 [0x5584a / 685 blocks]
...................................................................
        @@@  @@@@@@@ @@@@@@@@ @@@@@@@@@@   @@@@@@  @@@  @@@
        @@! !@@      @@!      @@! @@! @@! @@!  @@@ @@!@!@@@
        !!@ !@!      @!!!:!   @!! !!@ @!@ @!@!@!@! @!@@!!@!
        !!: :!!      !!:      !!:     !!: !!:  !!! !!:  !!!
        :    :: :: : : :: :::  :      :    :   : : ::    :
        .    .. .. . . .. ...  .      .    .   . . ..    .
...................................................................
...................................................................
...................................................................
............................................ ok

[+] All done

[=] Have a nice day!
```

3. Once that's completed you should be good to go to enter the Proxmark shell by typing in:

```
./pm3
```



```
[=] No previous history could be loaded
[usb] pm3 -->
```

This Proxmark3 shell will provide all the power to perform the exercises.

Flipper Zero Installation

The documentation to flash the firmware of the is documented here:

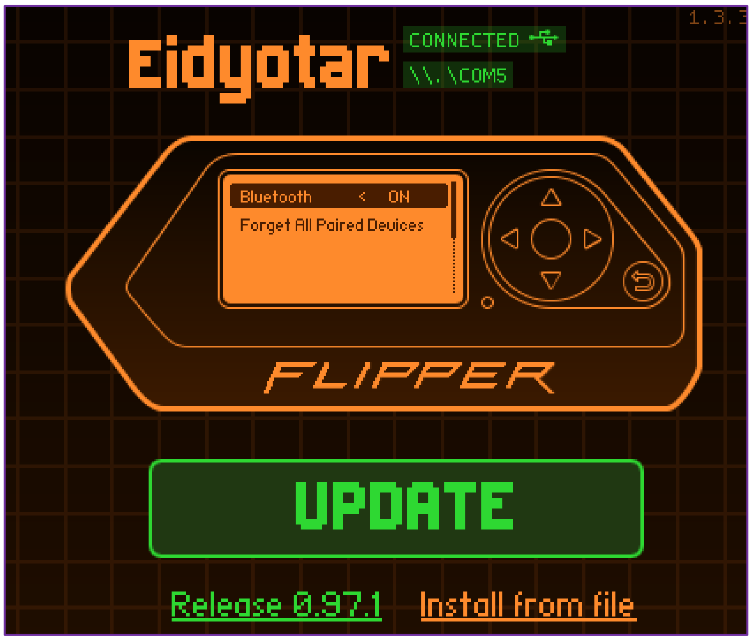
https://docs.flipper.net/basics/first-start

It is required to have a micro-SD Card available for your flipper device and it is advised to check for firmware updates. To achieve this, follow either the instructions to update through qFlipper application or the Flipper Mobile Application:

qFlipper Application

Install qFlipper tool from here: https://flipperzero.one/update

1. Connect your Flipper Zero to your computer with USB-C cable.
2. Open the qFlipper tool and Update the software

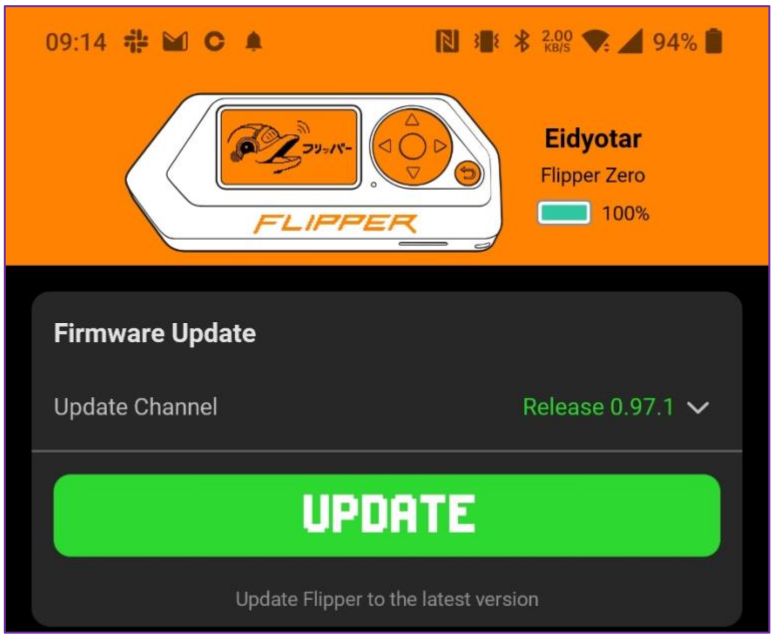


Flipper Mobile Application

iOS App Store: https://apps.apple.com/app/flipper-mobile-app/id1534655259
Android Play Store: https://play.google.com/store/apps/details?id=com.flipperdevices.app

1. Make sure the Flipper Zero has an SD card inserted.
2. Connect mobile phone and Flipper Zero with Bluetooth.

3. Initiate the Update through the mobile application.

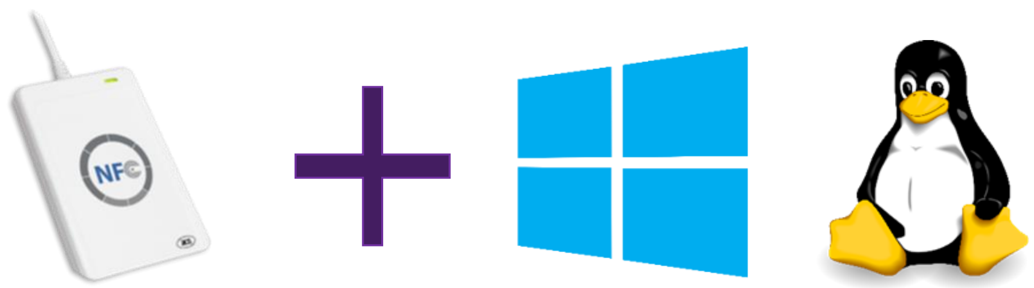


Eidyotar
Flipper Zero
100%

Firmware Update

Update Channel Release 0.97.1

Update Flipper to the latest version



ACR122U Setup for Windows and Linux

These installation guidelines will be similar for Windows and Linux since the tools we are going to use are created and python and work on both Operating systems. If you don't have any Python3 version installed yet, please follow the respective links to install Python3 for your system:

Python3: https://www.python.org/downloads/

Pip3: https://pip.pypa.io/en/stable/installing/

Additionally, on Linux you need to install the following packages from your OS package manager (might not need all of them):

```
sudo apt-get update
sudo apt-get install libpcsclite-dev libpcsclite1 pcscd pcsc-tools
```

Once Python is installed you have a couple of options to install the tools:

PIP

You can use pip3 to install the required tools by running the following command:

```
pip3 install swig crid
```

If you don't have pip3 installed, check if the command pip is a working alias for pip3 for python3 or install Python3/pip3.

You can also do pip3 install from the source directly:

```
git clone https://github.com/VinnieV/crid.git
cd crid
pip3 install .
```

To make sure you can just run the command crid in your terminal do the following:

<u>Windows:</u>

In an Administrative Command Line shell the following oneliner will add the Python scripts folder to the PATH in the current shell:

```
for /f "delims=" %a in ('python3 -m site --user-site') do set PATH=%PATH%;%a\..\Scripts
```

or for PowerShell:

```
$env:PATH += ";" + (python3 -m site --user-site) + "\..\Scripts"
```

Cloning and running from source

You can directly clone

```
git clone https://github.com/VinnieV/crid.git
cd crid
pip3 install -r requirements.txt
python3 main.py
```

Optionally for some challenges, the crid CLI will use some other tools to recover the keys. Install instructions through :

```
sudo apt-get install mfoc
```

OR

```
git clone https://github.com/nfc-tools/mfoc.git
cd mfoc
sudo apt-get install autoconf build-essential
autoreconf -is
./configure
make && sudo make install
```

To install the hardnested tool you can follow these instructions:

```
git clone https://github.com/VinnieV/crypto1_bs
cd crypto1_bs
make get_craptev1
make get_crapto1
make
sudo cp -a libnfc_crypto1_crack /usr/local/bin
```

Installing MFCUK for the darkside can be done as follows:

```
git clone https://github.com/nfc-tools/mfcuk.git
cd mfcuk
autoreconf -is
./configure
make && sudo make install
```

Installing MiLazyCracker can be done as follows:

```
git clone https://github.com/nfc-tools/miLazyCracker.git
cd miLazyCracker
./miLazyCrackerFreshInstall.sh
```

Debugging

- (Linux) sudo nfc-list error libnfc.driver.acr122_usb Unable to claim USB interface (Device or resource busy) nfc-list: ERROR: Unable to open NFC device: acr122_usb:001:020.
  Perform the following commands:
  `sudo modprobe -r pn533_usb && sudo modprobe -r pn533`
  But for a more permanent solution, create the following file:
  `sudo nano /etc/modprobe.d/blacklist-libnfc.conf` and then add the following lines:
  ```
  blacklist pn533
  blacklist pn533_usb
  blacklist nfc
  ```
- (Windows) error: command 'swig.exe' failed: None
  make sure to install download and unzip this file
  http://prdownloads.sourceforge.net/swig/swigwin-4.2.0.zip and copy swig.exe
  Then put the location of the of the unzipped folder in your Path environment variable.
- (Windows) error: Microsoft Visual C++ 14.0 or greater is required.
  Install and run this tool: https://visualstudio.microsoft.com/visual-cpp-build-tools/
  Make sure to install at least the package: MSVC v142 - VS 2019 C++ x64/x86 build tools (v14.0 or greater)

ACR122U installation MacOS Installation

```
brew install swig
pip3 install crid
```

Might be required for the following as well on MacOS version 14 (Sonoma)

```
pip3 install py122u smartcard pcsc-lite libusb --force
```

Or when having some issues when running the tools related to Transaction Error:

Failed to transmit with protocol T1

Then follow this to enable another driver:

https://blog.apdu.fr/posts/2023/11/apple-own-ccid-driver-in-sonoma/

Which is essentially running the following command:

```
sudo defaults write /Library/Preferences/com.apple.security.smartcard useIFDCCID -bool yes
```

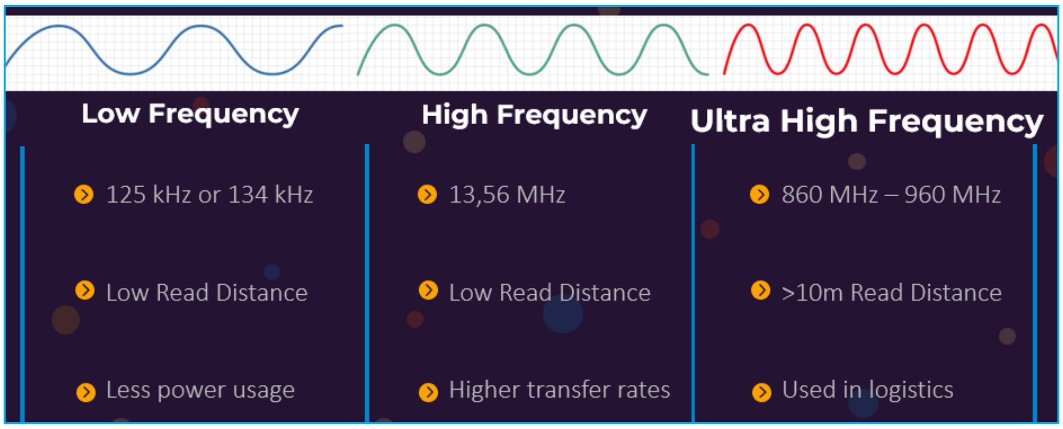
And might require reboot.

5. General RFID Theory

RFID (Radio Frequency Identification) uses radio signals to send data wirelessly. It is a passive technology meaning that the card itself doesn't require an active power source. Radio waves of a specific frequency reach the card's antenna where the magnetic field is transferred as electricity to power the electric circuit and microcontroller in the card. Its commonly used as an identification control during physical access checkpoints but more complex sequences can be performed such as contactless payments or ticket/subscription system as well. Because of the this, the security of these cards is important where companies tend to choose the cheaper side which has less secure cards, and migrating to secure systems is not a priority.

5.1. RFID Frequencies

Each card operates on a specific frequency which can be separated in three common frequencies, low, high and ultra-high frequency. Most of the smartcards you will find will use low or high frequency. Ultra-high is almost only used for logistical systems.



Low Frequency	High Frequency	Ultra High Frequency
125 kHz or 134 kHz	13,56 MHz	860 MHz – 960 MHz
Low Read Distance	Low Read Distance	>10m Read Distance
Less power usage	Higher transfer rates	Used in logistics

Low Frequency

The Low Frequency (LF) band covers frequencies from 30 kHz to 300 kHz but most LF RFID systems operate at 125 kHz, although there are some that operate at 134 KHz. The biggest advantage is that a lower power supply is needed for low frequency. That is why this frequency is usually found in places like doors where it needs to be powered with a small battery.

High frequency

The High frequency (HF) RFID cards almost always uses 13,56 MHz as frequency with the biggest advantage of having a bigger data transmission speed.

Ultra-high frequency

The UHF frequency band convers the range from 300 MHz to 3 GHz. Systems complying with the UHF Gen2 standard for RFID use the 860 to 960 MHz band. A big advantage for UHF is the read distance. Tags can be read from a larger distance (meters). This is useful for warehouse management.

	Frequency	**Read range**
Low frequency	125kHz 134.2khz	8 cm

High frequency	13.56 MHz	5 – 8 cm
Ultra-high frequency	865 – 929 MHz	1,5 – 2 m

5.2. RFID Layers

RFID solutions can be dissected into different layers, similar on how the OSI layer exist to help us understand the different layers of networking.

The International Organization Standardization (ISO) has several standards defined on how to RFID systems should operate. Standards commonly used are the following:

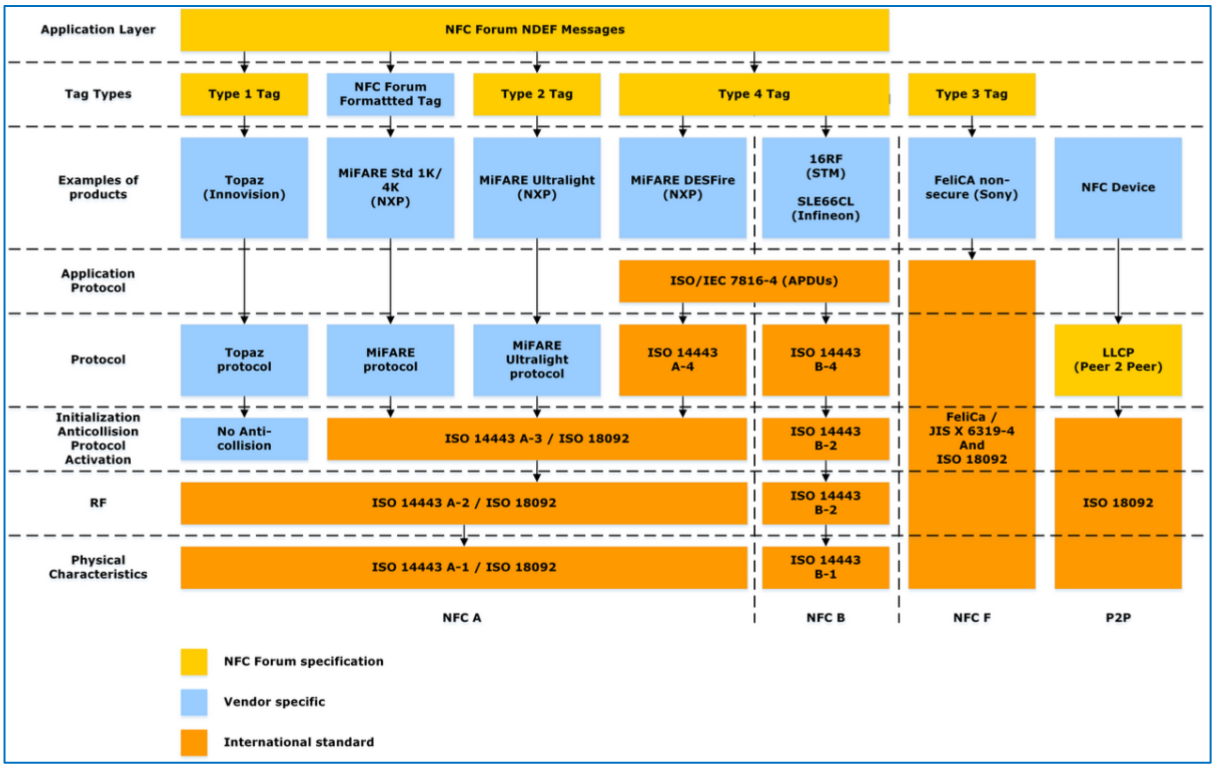
- ISO 14443
  For contactless smart cards
- ISO 15693
  For vicinity cards
- ISO 18000
  For Air Interface Communications
- ISO 19794
  Biometric

Most common RFID protocol is ISO 14443 which includes documentation for the following aspects:

- **ISO/IEC 14443-1:2018 - Part 1: Physical characteristics:**
  Focus: Describes the physical characteristics of the proximity cards, including dimensions, surface properties, and tolerances.
- **ISO/IEC 14443-2:2018 - Part 2: Radio frequency power and signal interface:**
  Focus: Specifies the power and signal interface between the card and the reader. It defines the radio frequency (RF) characteristics and communication parameters.
- **ISO/IEC 14443-3:2018 - Part 3: Initialization and anticollision:**
  Focus: Covers the initialization process and anticollision procedures, ensuring that multiple cards can be in the proximity field without interference.
- **ISO/IEC 14443-4:2018 - Part 4: Transmission protocol:**
  Focus: Defines the transmission protocol for communication between the card and the reader, including protocols for data exchange and error detection.

5.3. Near Field Communication (NFC)

Near Field Communication (NFC) is a subset of radio-frequency identification (RFID) technology, specifically designed for short-range communication. An NFC label within the context of RFID serves as a physical tag or identifier that incorporates NFC technology. The goal of an NFC label is to enable wireless communication and data exchange between the label (tag) and an NFC-enabled device, such as a smartphone, tablet, or other compatible devices.



https://en.wikipedia.org/wiki/Near-field_communication#/media/File:NFC_Protocol_Stack.png

5.4. RFID Protocols

During this workshop we continue to explore the HID and MIFARE Classic RFID protocols, since these are both commonly used quite frequently for RFID systems.

RFID Protocol	Date of Invention	Frequency
Low Frequency		
EM4100	1970s	125 kHz
Indala	1980s	125 kHz
HID Prox	1991	125 kHz
Hitag1, Hitag2	1994	125 kHz
Temic T55x7	Not specified	125 kHz
Unique	Not specified	125 kHz
High Frequency		
FeliCa	1987	13.56 MHz
Legic	1992	13.56 MHz

Mifare	2000	13.56 MHz
ISO 15693	2001	13.56 MHz
NFC (Near Field Communication)	2002	13.56 MHz
Ultra-High Frequency		
EPCglobal UHF Class 1 Gen 2	2004	860-960 MHz
ISO 18000-6A, 6B, 6C	Various	860-960 MHz

RFID protocols history table

Many of these RFID protocols have been identified as vulnerable to cloning because either these systems have no security controls implemented or the controls are vulnerable to various attacks. For example, the following protocols do not have any security controls implemented:

- EM4100
- HID
- Indala
- Temic T55X7

There are also other RFID protocol that have vulnerabilities:

- Mifare
    - Nested attack
    - Hardnested attack
    - Darkside attack
- Legic

Security is important for RFID systems, to ensure the integrity of the data and prevent unauthorized access.

5.5. APDU Commands

You have all these different layers within RFID, from the physical to timings, sessions and anti-collision, but what matters the most to us is which data we need to send to invoke the different functionality of the RFID chip within the card. These exchanges are done through:

Application **P**rotocol **D**ata **U**nits

These structured binary formatted commands exchanged between the two entities, provide a lot of functionality, such as reading and writing to the RFID chip memory sections, authenticating and so forth.

The following structure is defined as ADPU command for loading an authentication key on the reader as follows:

Load Authentication Keys APDU Command

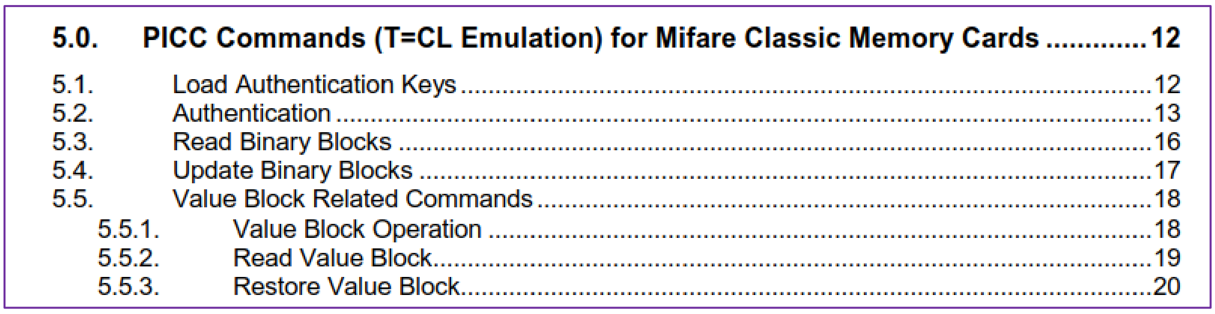
Example: FF82000006FFFFFFFFFFFF										
Length is 11 Bytes:										
1	2	3	4	5	6	7	8	9	10	11
Class	INS*	Key Structure	Key Location	Lc	Key (6 bytes)					
FF	82	00	00|01	06	XX XX XX XX XX XX					

*INS is the instruction identifier, meaning loading authentication keys in this example

Luckily there is some good documentation about these APDU commands and they are not so hard to understand, the following documents are good references:

- http://downloads.acs.com.hk/drivers/en/API-ACR122U-2.02.pdf

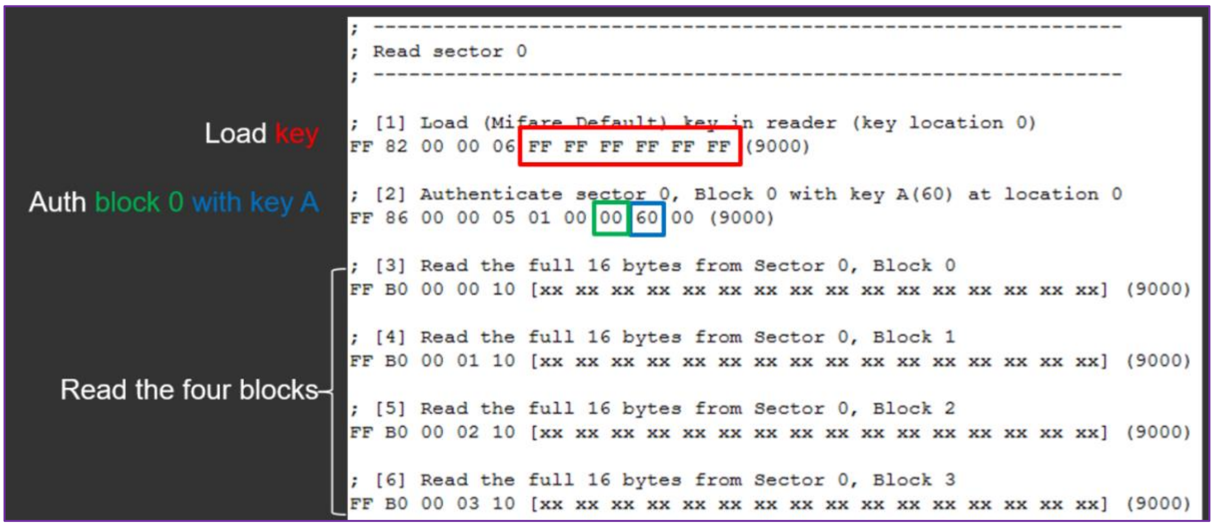
In one of these you will find the commands on how to read and write Mifare Classic cards:



5.0. PICC Commands (T=CL Emulation) for Mifare Classic Memory Cards 12

5.1. Load Authentication Keys 12
5.2. Authentication 13
5.3. Read Binary Blocks 16
5.4. Update Binary Blocks 17
5.5. Value Block Related Commands 18
5.5.1. Value Block Operation 18
5.5.2. Read Value Block.. 19
5.5.3. Restore Value Block... 20

As an example, the following commands would need to be executed to read the full contents of sector 0 using key A FFFFFFFFFFFF on a Mifare Classic 1K card:



```
; ---------------------------------------------------------------
; Read sector 0
; ---------------------------------------------------------------

; [1] Load (Mifare Default) key in reader (key location 0)
FF 82 00 00 06 FF FF FF FF FF FF (9000)

; [2] Authenticate sector 0, Block 0 with key A(60) at location 0
FF 86 00 00 05 01 00 00 60 00 (9000)

; [3] Read the full 16 bytes from Sector 0, Block 0
FF B0 00 00 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [4] Read the full 16 bytes from Sector 0, Block 1
FF B0 00 01 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [5] Read the full 16 bytes from Sector 0, Block 2
FF B0 00 02 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)

; [6] Read the full 16 bytes from Sector 0, Block 3
FF B0 00 03 10 [xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx xx] (9000)
```

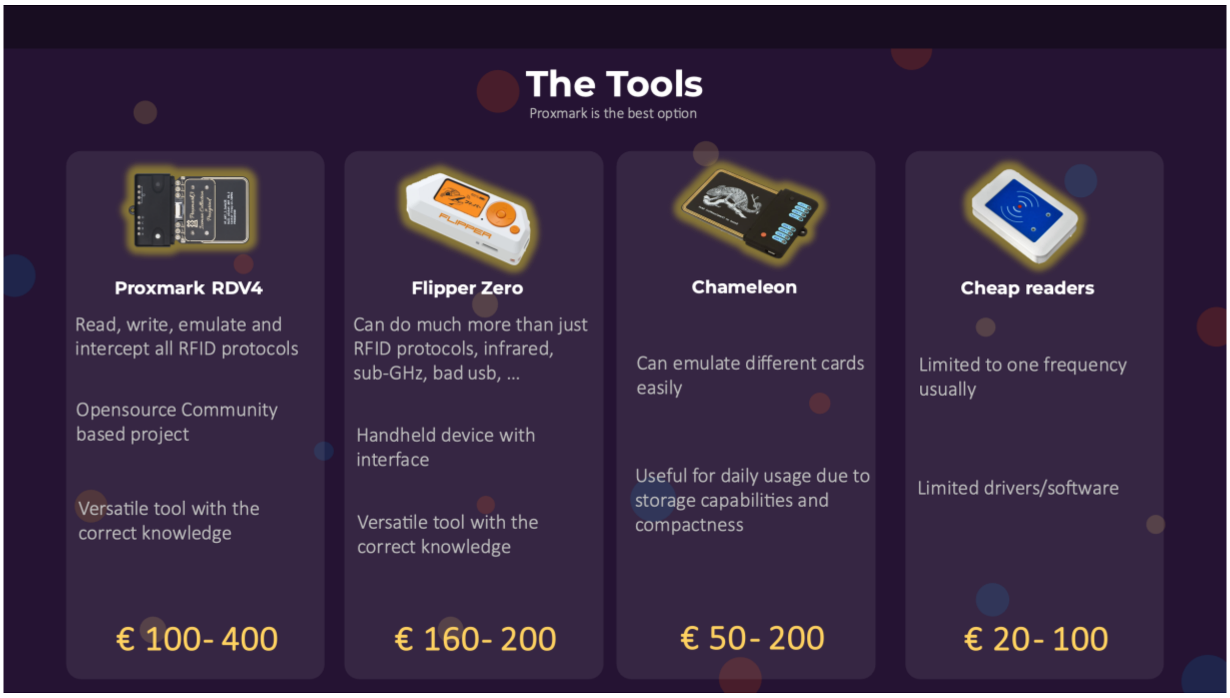
5.6. Secure?

So why do RFID systems need to be secure? What are we protecting?

The most common use case will be physical access control where each registered RFID card can be used to identify a person securely and allow access to certain areas of a secured building. The most important piece of information here is the data stored on the chip that identifies the user. To access it, a reader will typically send a password to access that piece of information within the RFID card. Thus, if an attacker is able to read that data, it would allow to clone the card and obtain unauthorized access.

One of the strongest RFID Protocols out there that has secure key exchange and encryption enabled for the data is currently Mifare DESFire EV3. When used correctly these should provide the strongest protection against cloning of the cards and unauthorized access. In addition to using a secure RFID protocol, what can aid the most in protecting physical access controls systems is adding detection when unauthorized access was granted. This can be done by adding a random number each time a user interacts with a reader and storing that number on the server and the card. Each time a new number is issued, the two values are cross examined to detect any duplicated cards going around.

5.7. Tools



The Tools

Proxmark is the best option

Proxmark RDV4

Read, write, emulate and intercept all RFID protocols

Opensource Community based project

Versatile tool with the correct knowledge

€ 100- 400

Flipper Zero

Can do much more than just RFID protocols, infrared, sub-GHz, bad usb, ...

Handheld device with interface

Versatile tool with the correct knowledge

€ 160- 200

Chameleon

Can emulate different cards easily

Useful for daily usage due to storage capabilities and compactness

€ 50- 200

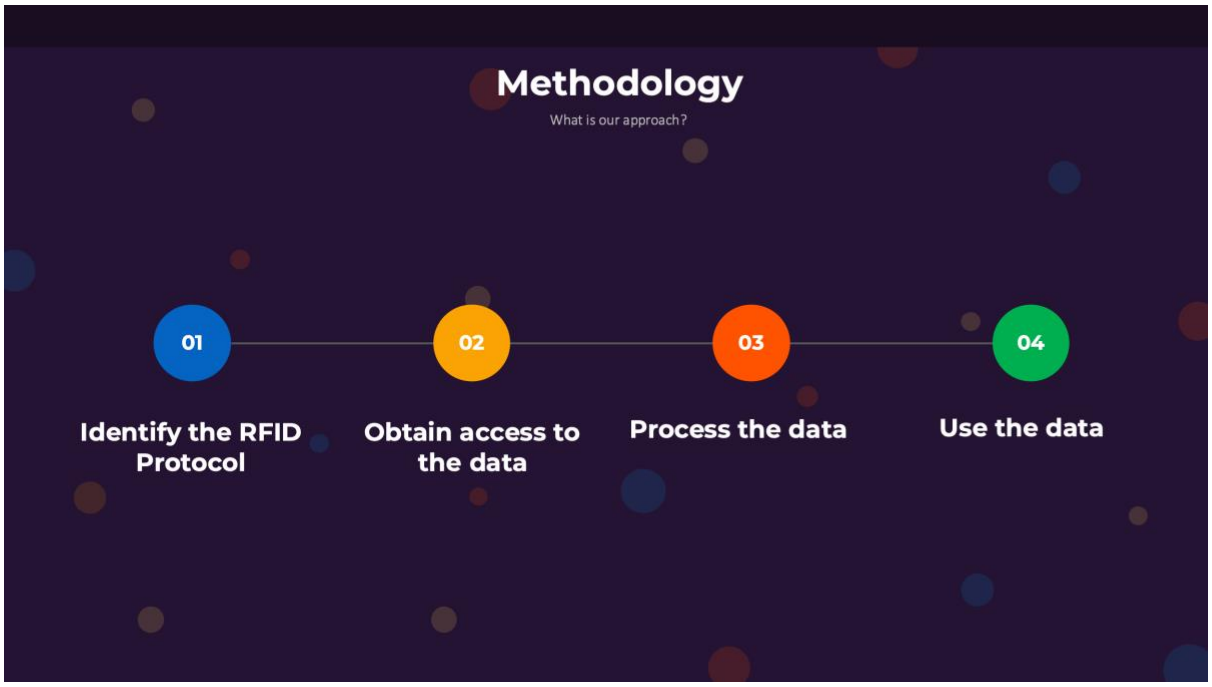
Cheap readers

Limited to one frequency usually

Limited drivers/software

€ 20- 100

5.8. Methodology



Methodology

What is our approach?

01 Identify the RFID Protocol

02 Obtain access to the data

03 Process the data

04 Use the data

6. HID Prox ID

HID Prox ID is a proximity card technology developed by HID Global, a company specializing in secure identity solutions. The HID Prox technology was developed in the 1990s.

HID Prox cards operate on a 125 kHz frequency and contain a unique identifier (ID) that is read by proximity card readers. The identifier is often a numeric code, and the card itself may have additional memory for storing user-specific data or access credentials.

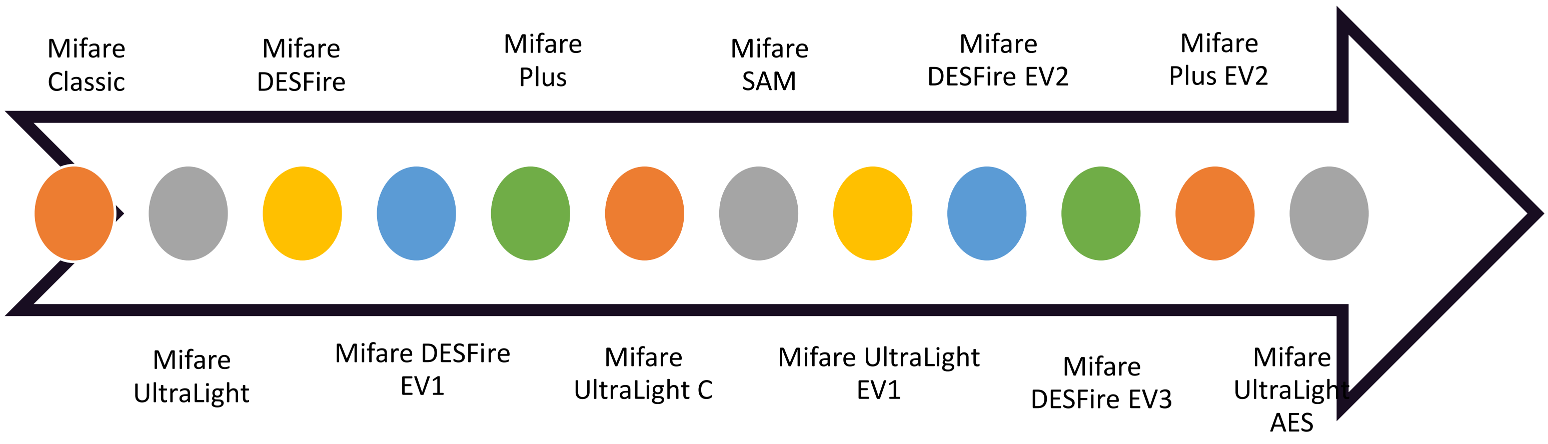
Unfortunately, the card does not protect the identifier and the sole protection implemented for this type of card is the fact that the cards are supposed to be read-only. Meaning that nobody would be able to clone a badge if you can’t write to any badge. However, there are special cheap cards on the market where this number is modifiable, enabling the possibility to clone these cards and unlike other systems that require pass keys to read the data, this is just a readable number through RFID which goes really quick.

HID Prox technology has been widely adopted globally for access control and identification purposes. It is commonly used in various industries, including corporate offices, government facilities, educational institutions, and more. This might have been because it was most likely a cheaper option and because changing such a system in a building placed before the security implication were known is expensive.

7. MIFARE

MIFARE stands for “MIkron FARE Collection System” and it covers several different kinds of contactless cards. Its created by NXP Semiconductors company. It uses ISO/IEC 14443 Type A 13.56 MHz. The technology should be embodied in the cards and readers. MIFARE provides different kinds of security mechanisms. MIFARE also have the most secure cards on the market.

MIFARE Timeline 1994-2023



7.1. MIFARE Classic

The MIFARE Classic cards are fundamentally just a memory storage device, where the memory is divided into segments and blocks with simple security mechanisms for access control. Those cards are ASIC-based and have limited computational power. They are still heavily used today for access control systems in hotels and companies, for transportation and ticketing systems.

The MIFARE Classic Mini version offers 320 bytes of data storage divided into 5 sectors. It uses 16 bytes per sector for the keys and access conditions and cannot be used for user data.

MIFARE Classic 1K offers 1024 bytes of data storage divided in 16 sectors. Every sector is protected by two different keys, called A and B. Each key can be programmed to allow operations such as reading, writing, increasing value blocks, etc.

MIFARE Classic 4K offers 4096 bytes split into 40 sectors. 32 of those sectors are the same size as the 1K sectors and 8 sectors are quadruple size sectors.

The MIFARE Classic uses Crypto-1 protocol for authentication and ciphering.

	Bruto Data storage	**Sectors**	**Netto Data Storage**
MIFARE Classic Mini	320 bytes	5	224 bytes
MIFARE Classic 1K	1024 bytes	16	752 bytes
MIFARE Classic 4K	4096 bytes	40	3440 bytes

MIFARE Memory Capacities

The MIFARE card series, including the S50 and S70, are products of NXP Semiconductors. They are a part of the MIFARE family, which is widely used in various contactless smart cards and proximity cards. Here's a brief overview of the differences between the MIFARE S50 and S70 cards, and a general description of NXP cards:

MIFARE S50 (also known as MIFARE Classic 1K):

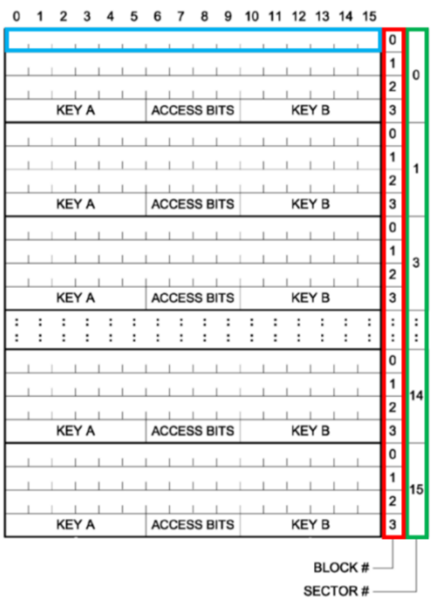
- Memory: The S50 has 1KB of memory.
- Security: Features basic security mechanisms primarily for simple applications like public transport, loyalty cards, etc.
- Structure: Divided into 16 sectors, each with 4 blocks.
- Application: Commonly used for transport tickets, simple access control, and low-security applications.

MIFARE S70 (also known as MIFARE Classic 4K):

- Memory: The S70 has 4KB of memory, offering more storage capacity compared to the S50.
- Security: Similar security features as the S50, but the increased memory allows for more complex applications.
- Structure: Divided into 40 sectors, with the first 32 sectors containing 4 blocks each and the last 8 sectors containing 16 blocks each.
- Application: Suitable for applications requiring more data storage like multi-application smart cards, more complex access control systems, and multi-use systems

7.2. MIFARE Classic Memory Layout

A MIFARE Classic 1K card has 16 sectors containing each 4 blocks and a block contains 16 bytes. In the fourth block of **each** sector you can find key A, B and the access bits. The keys are used to read/write on that specific sector and its perfectly possible to have different keys for each sector. The access bits contain information about the access that can be achieved with key A or key B. This means that in a MIFARE Classic 1K memory there will be 756 free bytes.



```
16 sectors
3 blocks/sector
16 bytes/block

16 x 3 x 16 = 768 – 16(UID) = 756 bytes
```

UID

Then there is also the very first block of the first sector, block 0 which contains the UID of the card. The length of the UID of the card can vary, since MIFARE started with distributing 4-byte UID cards but noticed in 2010 that depletion of the range would occur. So, they increased the UID bytes to 7[1]. The UID bytes are originally not writeable since they are hardcoded by the card manufacturer. However, there are special MIFARE classic cards where the UID can be modified. These magic MIFARE cards have special APDU commands available to modify the UID portion. There are many versions of it using different techniques, since some of the RFID reader could check for magic Mifare cards

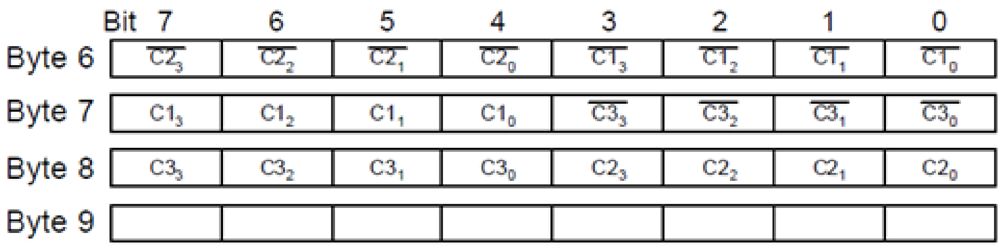
Access bits

Access bits are not difficult to understand but can be tedious to calculate manually. You can specify the access conditions for each block within that sector using the access bits:

$C1_0$	$C2_0$	$C3_0$	read	write	increment	decrement, transfer, restore
0	0	0	key A|B[1]	key A|B[1]	key A|B[1]	key A|B[1]
0	1	0	key A|B[1]	never	never	never
1	0	0	key A|B[1]	key B[1]	never	never
1	1	0	key A|B[1]	key B[1]	key B[1]	key A|B1
0	0	1	key A|B[1]	never	never	key A|B[1]
0	1	1	key B[1]	key B[1]	never	never
1	0	1	key B[1]	never	never	never
1	1	1	never	never	never	never

The $C1_x$-$C3_x$ of all the blocks are used to calculate the access bits as follows:

[1] https://www.cartaplus.be/4-byte-and-7-byte-uid-offering-of-mifare-classic-mifare-plus-smartmx-and-licensed-products/



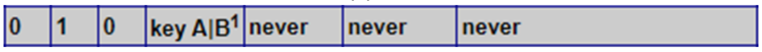
Bit	7	6	5	4	3	2	1	0
Byte 6	$\overline{C2_3}$	$\overline{C2_2}$	$\overline{C2_1}$	$\overline{C2_0}$	$\overline{C1_3}$	$\overline{C1_2}$	$\overline{C1_1}$	$\overline{C1_0}$
Byte 7	$C1_3$	$C1_2$	$C1_1$	$C1_0$	$\overline{C3_3}$	$\overline{C3_2}$	$\overline{C3_1}$	$\overline{C3_0}$
Byte 8	$C3_3$	$C3_2$	$C3_1$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

For example, let say for a given sector we only want the first block to be readable by key A or B, then it would result in the following:

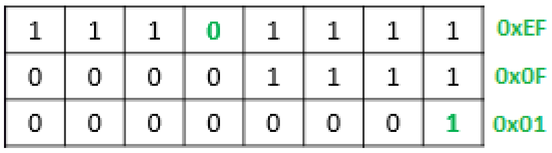
Access bits setup:

1	1	1	1	1	1	1	1
0	0	0	0	1	1	1	1
0	0	0	0	0	0	0	0

$C2_0$ will be flipped in this case:



0	1	0	key A|B[1]	never	never	never



1	1	1	0	1	1	1	1	0xEF
0	0	0	0	1	1	1	1	0x0F
0	0	0	0	0	0	0	1	0x01

Access bits: 0xEF 0x0F 0x01

Now don't use these access bits since it would lock out the sector for good, since no sector key can write anything anymore.

Usually, key A is used to read from the card and is configured to be used by the RFID readers that only need to read the data, such as a door lock. The second key will be used to configure and write the card which is usually used in more secure environment, such as the registration desk. At least, this was probably the idea until clever hackers figured out ways to hack these cards. and this access is defined in those access bits and key B is used to read/write to the card. This is useful if you want to store the read key in an exposed reader but the write key at badge maintainer/administrator.

7.3. Mifare classic vulnerabilities

Mifare classic implements the proprietary "crypto1" encryption, it was kept secret for a long time until German researchers (Henryk Plötz and Karsten Nohl[2]) investigated the card by analyzing the chip with a microscope and scraping the chip to its core. After reversing the chip they identified multiple flaws with the Crypto1 cryptographic function. They noticed that the key length was too small to be cryptographically strong enough. But more important while running some tests they discovered that the random number generator was not random enough which resulted in a couple of attacks to recover the access keys of the:

1. Nested attack

[2] https://fahrplan.events.ccc.de/congress/2007/Fahrplan/events/2378.en.html

2. Hardnested attack
   Mifare came with an upgraded version of the mifare classic card with a better RNG but it is still vulnerable.

3. Dark-side attack

Nested attack/Hardnested/Staticnested attack

- Requires a valid key
- Exploits the linear feedback shift register (LFSR)
- Timing based attack
- Collects many nonces/challenges to obtain the keys through some cryptographic magic

https://github.com/bennesp/hardnested

https://github.com/nfc-tools/miLazyCracker

https://www.cs.ru.nl/~rverdult/Ciphertext-only_Cryptanalysis_on_Hardened_Mifare_Classic_Cards-CCS_2015.pdf

https://github.com/aczid/crypto1_bs

https://github.com/nfc-tools/mfoc-hardnested

https://github.com/bennesp/hardnested

Darkside attack

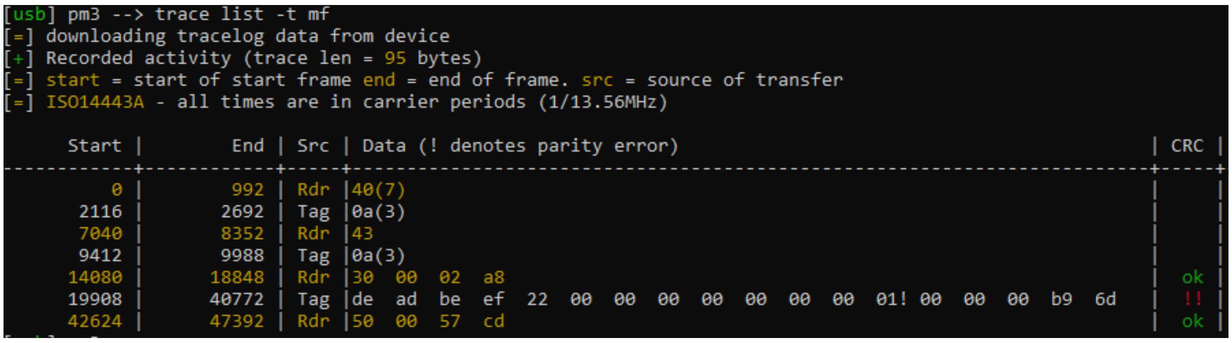
- During authentication the tag responds with some encrypted data
- The encrypted data is a static error code
- Can be used to recover the key

https://github.com/nfc-tools/mfcuk

https://eprint.iacr.org/2009/13 7.pdf

7.4. Magic Mifare

As mentioned earlier, the UID of the Mifare Classic Card should be readonly, and that's the case for all official NXP Mifare cards. However, there are modified cards available that enable special commands to overwrite the UID and contain functionality to ignore the access bits and write to the blocks without any restrictions. When doing a Mifare UID configuration for gen1 Magic Mifare cards the following APDU commands occur:



```
[usb] pm3 --> trace list -t mf
[=] downloading tracelog data from device
[+] Recorded activity (trace len = 95 bytes)
[=] start = start of start frame end = end of frame. src = source of transfer
[=] ISO14443A - all times are in carrier periods (1/13.56MHz)

      Start |        End | Src | Data (! denotes parity error)                                           | CRC |
------------+------------+-----+-------------------------------------------------------------------------+-----+
          0 |        992 | Rdr |40(7)                                                                    |     |
       2116 |       2692 | Tag |0a(3)                                                                    |     |
       7040 |       8352 | Rdr |43                                                                       |     |
       9412 |       9988 | Tag |0a(3)                                                                    |     |
      14080 |      18848 | Rdr |30  00  02  a8                                                           |  ok |
      19908 |      40772 | Tag |de  ad  be  ef  22  00  00  00  00  00  00  00  01! 00  00  00  b9  6d   | !!  |
      42624 |      47392 | Rdr |50  00  57  cd                                                           |  ok |
```

8. Challenges

Please refer to the Challenge bundle for the challenges and the Solutions bundle for the solution of the challenges for each supported device.