RFID Training
Theory Section

by Vanhoecke Vinnie @ Brucon 0x11

1. Table of content

1. Table of content 2
2. Introduction 5
3. OctoBox 6
   3.1. Profile 7
   3.2. OctoBox Standalone Setup 7
   3.3. Card Setup 8
4. Tool setup/installation 9
   4.1. Proxmark3 9
   Proxmark3 Linux Installation 10
   Proxmark3 macOS Installation 12
   Proxmark3 Windows Installation 14
   4.2. Flipper Zero 17
   Flipper Zero Installation 17
   4.3. ACR 122U 20
   ACR122U Setup for Windows and Linux 20
   ACR122U macOS Installation 23
5. General RFID Theory 24
   5.1. PICC vs PCD 24
   5.2. Active RFID vs Passive RFID 24
   5.3. Radio Signal Power 26
   5.4. RFID Ranges 26
   5.5. Modulation 27
   5.6. RFID Frequencies 25
   Low Frequency 25
   High frequency 25
   Ultra-high frequency 25
   5.7. RFID Layers 28
   5.8. Near Field Communication (NFC) 30
   5.9. RFID Protocols 29

5.10. APDU/Commands 31
5.11. Example data flow between reader and card 32
5.12. Protections 34
5.13. Tools 34
6. HID Proximity Cards 36
6.1. HID iClass 37
6.2. Modulation 37
6.3. Data Transmission 37
6.4. Data Formats 38
6.5. Wiegand 39
6.6. T55xx 40
7. MIFARE Classic 41
7.1. MIFARE Classic Memory Layout 42
UID 42
Access bits 42
MIFARE Application Directory (MAD) 44
7.2. MIFARE Classic APDU Commands 45
ATQA/SAK/NAK 45
CRC 46
Crypto1 46
MIFARE Authenticate Sequence 47
Reading/Writing to a card 48
7.3. MIFARE classic vulnerabilities 50
Darkside attack 50
Nested attack / Hardnested / Staticnested attack 50
FUDAN Backdoor 51
7.4. Magic MIFARE 51
8. MIFARE UltraLight 53
8.1. MIFARE Ultralight Memory Layout 53
8.2. MIFARE Ultralight Commands 53
8.3. MIFARE Ultralight Security Features 53

8.4. MIFARE Ultralight Vulnerabilities 54
8.5. Use Cases and Applications 54
8.6. Tools for Working with MIFARE Ultralight 54
9. Table of content 54
10. Introduction 55
10.1. Training RFID Cards 55
11. HID Proximity Cards 56
11.1. HID Proximity Cards Exercises 56
Exercise #0: LF Read 56
Exercise #1: T5577 57
11.2. HID Proximity Cards Challenges 58
Challenge #1: ATM 1 58
Challenge #2: ATM 2 61
12. MIFARE Classic 62
12.1. MIFARE Classic Exercises 62
Exercise #0: Identify 62
Exercise #1: PIN Code 67
Exercise #2: Username 72
Exercise #3: Access Bits 77
Exercise #4: Brute Force 82
Exercise #5: Key Recovery 87
Exercise #6: Key Recovery 2 93
Exercise #7: Magic MIFARE 96
12.2. MIFARE Classic Challenges 101
Challenge #1: Vault 101
Challenge #2: Employee Portal 105
Challenge #3: Vending Machine 110
Challenge #4: Hotel Rooms 115
Challenge #5: Speedrun 121

2. Introduction

Welcome to the theory section of the RFID Training Bundle, created by Vinnie Vanhoecke. This section provides an introduction to the fundamentals of RFID systems, detailed documentation of the RFID protocols covered in this training, and setup instructions for the various supported RFID devices and the OctoBox.

The challenges and exercises related to each chapter can be found in the second part of the RFID Training Bundle, designed to help you apply the knowledge gained here in simulated real-world scenarios.

This first section outlines how to install the required training tools and software (including the Proxmark3, Flipper Zero, ACR122U, and OctoBox) across different operating systems. Once your tools are set up, we'll dive into the core concepts of RFID systems with a focus on HID Proximity Cards and MIFARE Classic technology, giving you the theoretical knowledge needed for effective hands-on practice.

It's recommended to revisit key sections, such as the MIFARE Classic memory layout, while working through the exercises and challenges.

By the end of this training, you'll have a solid foundation to tackle real-world RFID hacking scenarios with confidence.

3. OctoBox

OctoBox will be the software/device used during the training to perform different exercises and simulate real-life challenges. You can either run it standalone using an installation file on your computer and connect it with a compatible USB RFID device or you could buy the actual OctoBox device itself.

Figure 1- OctoBox Main Menu

The OctoBox device features a Raspberry Pi 5 paired with a 4.3-inch touchscreen. On the left side, you'll find the low-frequency RFID reader, while the high-frequency reader is positioned on the right side of the OctoBox. The software powering the device is written in Python 3, utilizing the Kivy library for its user interface.

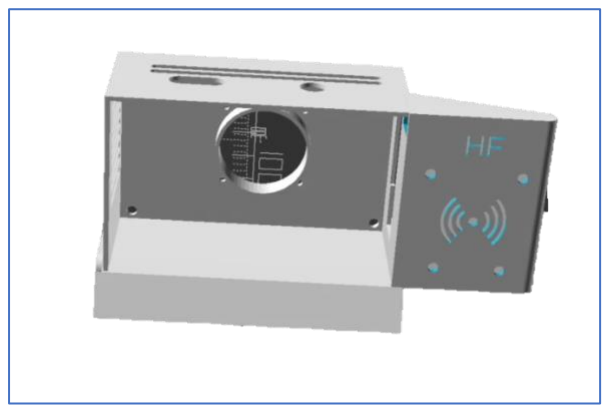


Figure 2- OctoBox Device

It's important to note that low-frequency exercises and challenges cannot be completed using the standalone version of the OctoBox, as the ACR122U is exclusively a high-frequency RFID reader. However, this limitation affects only about three challenges.

3.1. Profile

The OctoBox software tracks your progress throughout the training, awarding points as you complete exercises and challenges. More complex tasks yield higher point values. Once you reach the required number of points, you will receive the password needed to access your training certificate, serving as proof of completion.

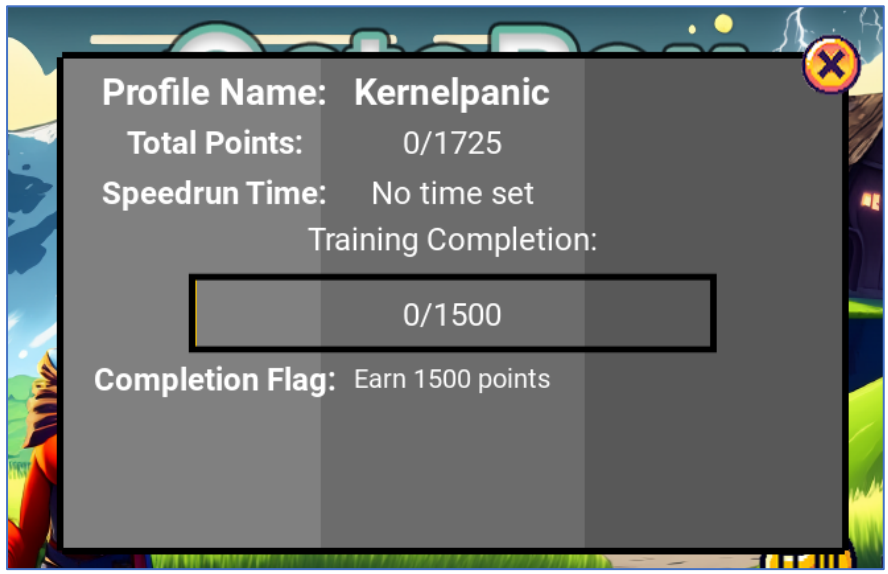


Figure 3- OctoBox Software Profile Screen

3.2. OctoBox Standalone Setup

The OctoBox software is available for Linux (.deb), macOS (.dmg), and Windows (.msi) and becomes accessible upon obtaining a license for the training. If any issues arise with the installation files, the code package can be executed directly as an alternative.

To install the application, simply use the appropriate installer for your operating system. After installation, connect a compatible NFC USB reader and select it from the list within the Settings page to ensure proper functionality.

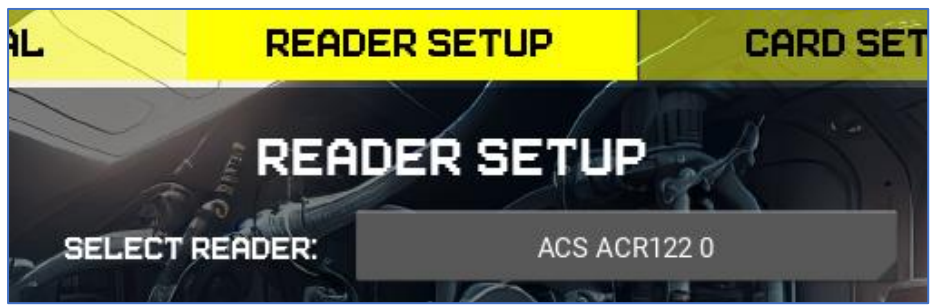


Figure 4- OctoBox Software Reader Settings Screen

If you have any troubles with the installing or using the OctoBox software, feel free to reach out to rfidtraining@dtrox.com

3.3. Card Setup

The OctoBox software also includes functionality to reconfigure the training cards. To do this, navigate to the Settings menu and select the Card Setup tab. From there, you can choose the card you wish to reconfigure. This feature is particularly useful for resetting a card's data back to its original training configuration, ensuring it's ready for repeated use in exercises and challenges.

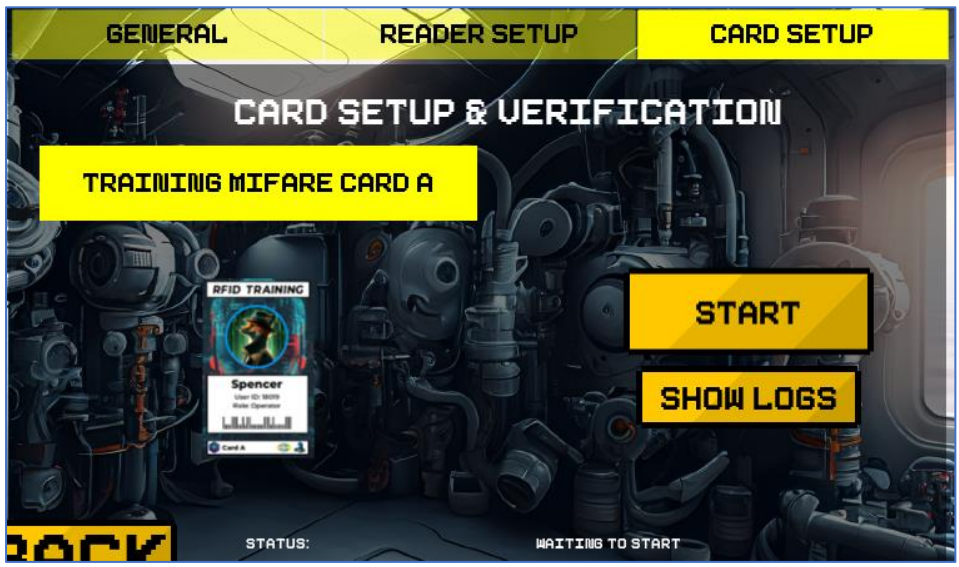


Figure 5- OctoBox Software Card Setup Screen

4. Tool setup/installation

The training includes detailed installation walkthroughs for three widely used RFID devices: the Proxmark3, Flipper Zero, and ACR122U. However, the exercises and challenges can also be completed using other RFID reader/writers, including mobile phones or alternative NFC-enabled USB devices.

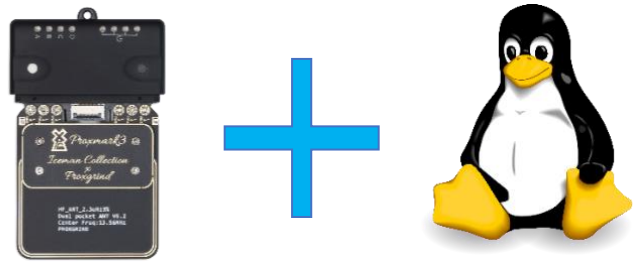
Please note that these instructions have not been tested within Virtual Machines (VMs), and using a VM is generally discouraged due to common issues with USB passthrough. The same applies to Windows Subsystem for Linux (WSL), where USB passthrough is not officially supported. That said, with some extra effort, it is still possible to make it work if you are determined to do so. Some users have found success using specific VM or USB passthrough configuration.

4.1. Proxmark3

The Proxmark3 is often referred to as the Swiss Army knife of RFID research. Its codebase is primarily written in C, is fully open source, and is relatively straightforward to use, especially for those with experience in embedded programming, electronics, or Linux. The tool was originally designed by Jonathan Westhues but now actively maintained by many volunteers with one of the key maintainers being legendary Iceman.

One of the Proxmark3's greatest strengths lies in its active community and ongoing maintenance. Frequent updates keep the tool current and reliable, which is a significant advantage for an open-source project. Its extensibility also makes it an excellent learning resource, allowing users to build upon and explore the work contributed by others.

Be sure to follow the installation instructions specific to your operating system (Windows, Linux, or macOS) to set up the device properly.



Proxmark3 Linux Installation

The documentation to build and flash the Proxmark3 firmware and build the Proxmark3 client for Linux is documented here:

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

On Debian / Ubuntu and derivatives, follow the steps below (on Arch, Fedora or openSUSE, check the link above):

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 libgd-dev
```

3. **Download latest release from the Proxmark3 repository**
   https://github.com/RfidResearchGroup/proxmark3/releases
   Select the latest version and download the Source code zip or tar.gz file.
   Example for fetching a tar.gz version released on 22 November 2024:

```
wget
https://github.com/RfidResearchGroup/proxmark3/archive/refs/tags/v4.19552.tar.gz -
O proxmark3.tar.gz
tar -zxvf proxmark3.tar.gz
cd proxmark3-4.19552
```

4. **Build the code:**

```
make clean && make -j
```

Building might take some time and hopefully no errors occurred that require additional troubleshooting. Once successful, it should have created the firmware binaries and the Proxmark3 Client. The next step would be to flash the firmware on the Proxmark, by following these steps:

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

5. **Connect your Proxmark3 device to your computer.**

⚠ **Important!** Make sure ModemManager does not interfere during the flashing process, make sure to remove it from your system or reference the other solutions within the following link: https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/ModemManager-Must-Be-Discarded.md

```
sudo apt remove modemmanager
```

6. **Flash the firmware**
   Once connected, you can run the flash all binary to install the bootloader and firmware on the Proxmark3 (Sometimes its required to have the battery and/or Bluetooth switch turned off):

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

If you recently flashed the bootloader, you could just flash the firmware image by issuing:

```
./pm3-flash-fullimage
```

7. **Test the Proxmark3 shell**
   Once that's completed you should be good to go to enter the Proxmark3 shell by typing in:

```
./pm3
```

The Proxmark3 shell provides all the functionality needed to complete the training

Proxmark3 macOS Installation

The documentation to build and flash the Proxmark3 firmware and build the Proxmark3 client on macOS is documented here:

https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/macOS-Homebrew-Installation-Instructions.md#homebrew-macos-developer-installation

1. **Install required packages using Homebrew:**

```
brew install readline qt5 gd pkgconfig coreutils
brew install RfidResearchGroup/proxmark3/arm-none-eabi-gcc
```

2. **Download latest release from the Proxmark3 repository**
   https://github.com/RfidResearchGroup/proxmark3/releases
   Select the latest version and download the Source code zip or tar.gz file.
   Example for fetching a tar.gz version released on 22 November 2024:

```
wget
https://github.com/RfidResearchGroup/proxmark3/archive/refs/tags/v4.19552.tar.gz -
O proxmark3.tar.gz
tar -zxvf proxmark3.tar.gz
cd proxmark3-4.19552
```

3. **Build the code:**

```
make clean && make -j
```

Building might take some time and hopefully no errors occurred that require additional troubleshooting. Once successful, it created the firmware binaries, ready to be flashed on the Proxmark3, by following these steps:

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

4. **Connect your Proxmark3 device to your computer.**
5. **Flash the firmware**
   Once connected, you can run the flash binary to install the compiled bootloader and firmware on the Proxmark3 (Sometimes its required to have the battery and/or Bluetooth switch turned on)::

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

If you recently flashed the bootloader, you could just flash the firmware image by issuing:

```
./pm3-flash-fullimage
```

6. **Test the Proxmark3 shell**
   Once that's completed you should be good to go to enter the Proxmark3 shell by typing in:

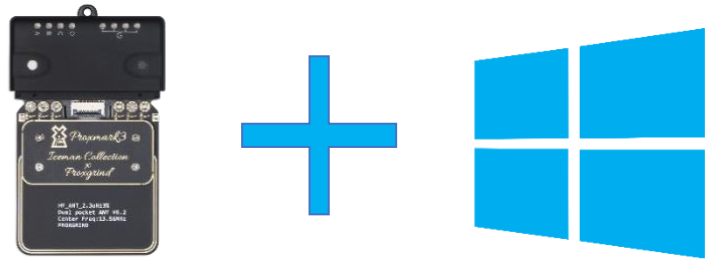
```
./pm3
```

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

Debugging

- If you are having issues with pm3-flash-all to get the Proxmark3 in bootloader mode:
  With your Proxmark3 unplugged from your machine, press and hold the button on your Proxmark3 as you plug it into a USB port. You can release the button, two of the four LEDs should stay on and you should be in bootloader mode, reader to flash.[1]

[1] https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/macOS-Homebrew-Installation-Instructions.md#the-button-trick



Proxmark3 Windows Installation

The documentation to build and flash the Proxmark3 firmware and build the Proxmark3 client on Windows can be found here:

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

While there are also installation instructions available for WSL, this guide focuses on the ProxSpace setup. However, if you prefer to explore the WSL method consider using the WSL1 installation guide instead of the WSL2:

https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/md/Installation_Instructions/Windows-Installation-Instructions.md#installing-dev-environment-with-wsl-1

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 to install everything, but it should open a new terminal with a prompt starting with pm3:

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

This environment provides you with a building environment to compile and flash the Proxmark3 repository on Windows. To perform this, follow these steps on the ProxSpace terminal:

4. **Go to the release tab on the Proxmark3 repository**
   https://github.com/RfidResearchGroup/proxmark3/releases
   Select the latest version and download the Source code zip or tar.gz file.
   Example for fetching a tar.gz version released on 22 November 2024:

```
wget https://github.com/RfidResearchGroup/proxmark3/archive/refs/tags/v4.19552.tar.gz -O
proxmark3.tar.gz
```

```
tar -zxvf proxmark3.tar.gz
cd proxmark3-4.19552
```

5. **Then build the code:**

```
make clean && make -j
```

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

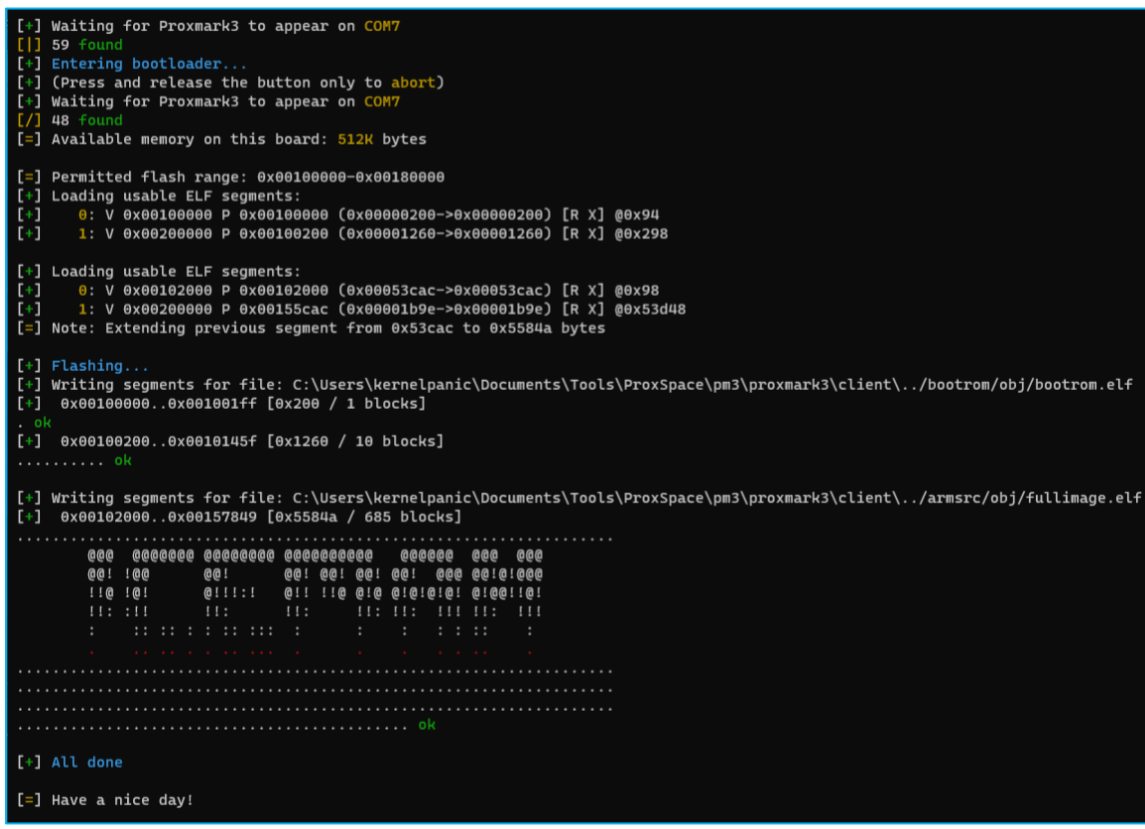
⚠ Debugging Tip: Check if the USB cable is not charge-only type cable.

6. **Connect your Proxmark3 device to your Windows computer.**
7. **Flash firmware**
   Once connected, you can run the flash binary to install the bootloader and firmware on the Proxmark3 (Sometimes its required to have the battery and/or Bluetooth switch turned off):

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

If you recently flashed the bootloader, you could just flash the firmware image by issuing:

```
./pm3-flash-fullimage
```



8. **Test the Proxmark3 shell**
   Once that's completed you should be good to go to enter the Proxmark3 shell by typing in:

```
./pm3
```

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

4.2. Flipper Zero

Introduced in 2020, the Flipper Zero quickly gained attention as a versatile, pocket-sized device. It features both Low Frequency and High Frequency RFID readers, backed by strong community support. While it doesn't match the Proxmark3 in terms of RFID-specific capabilities, the Flipper Zero offers a broader range of features, supporting protocols like Infrared, Wi-Fi, and Sub-GHz, which is impressive for such a compact, battery-powered tool.

When it comes to RFID research, the Proxmark3 still significantly outperforms the Flipper Zero. However, the Flipper remains a valuable addition to any toolkit, especially given its multifunctionality. With continued development and community involvement, its RFID support may improve over time, potentially narrowing the gap.

It's worth noting that the Flipper Zero can be less user-friendly for certain tasks and challenges. Interacting with RFID cards using the Flipper differs from the approach used with other devices, which may require some adjustment. While the Proxmark3 offers granular control for advanced RFID tasks, the Flipper Zero provides simplicity and portability for general use, making it a great tool for fieldwork and multi-protocol environments.

Flipper Zero Installation

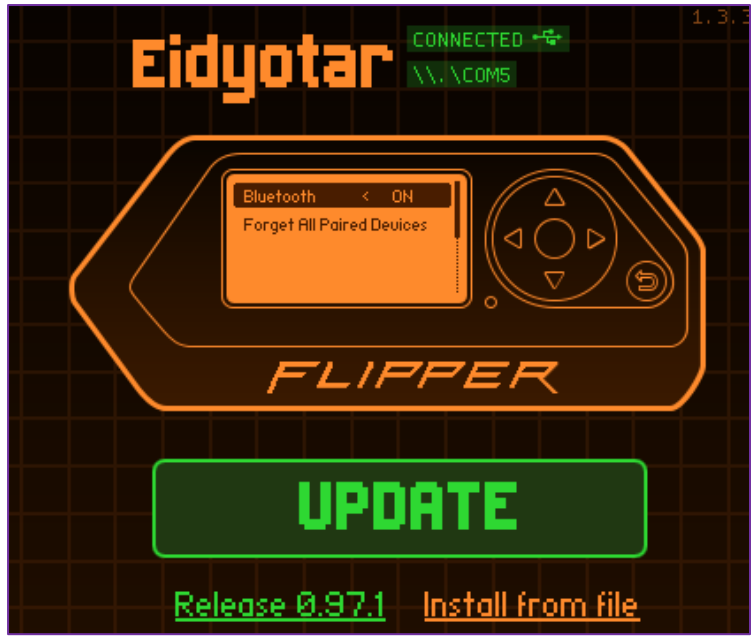
The documentation to flash the firmware of the Flipper Zero is documented here:

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

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

qFlipper Application

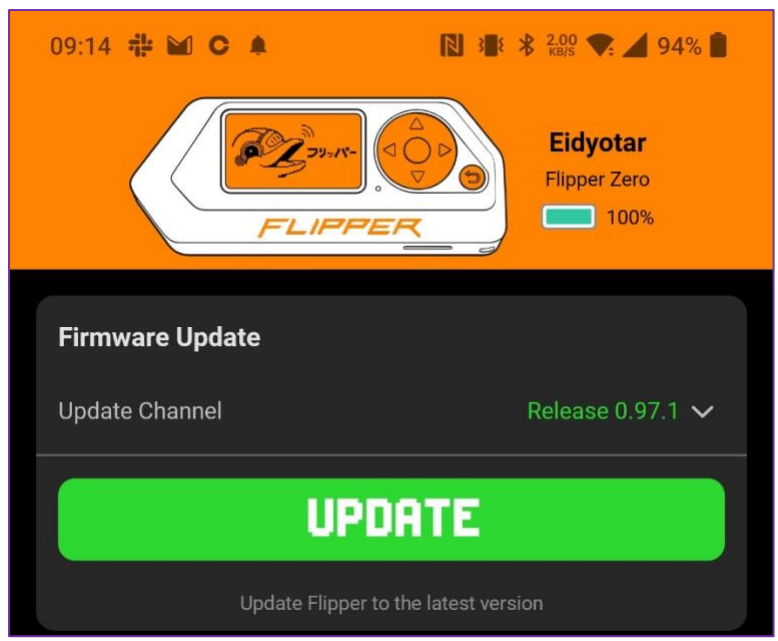
1. Install qFlipper tool from here: https://flipperzero.one/update
2. Connect your Flipper Zero to your computer with USB-C cable.
3. Open the qFlipper tool and Update the software



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.



Required Flipper Apps for training

In order to complete the training a few apps on the Flipper App store will be used. The following apps will be used:

- https://lab.flipper.net/apps/nfc_magic
- https://lab.flipper.net/apps/mfkey

Follow these instructions to install the required applications:

1. Go to mobile application
2. Make sure Flipper is connected and synchronized
3. Through app store install required applications

4.3. ACR 122U

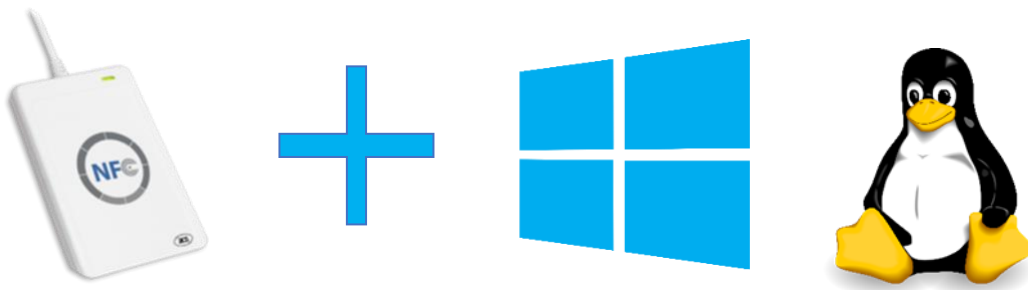
The ACR122U is an affordable NFC reader and writer, typically priced between 20 and 50 euros, and is compatible with several NFC libraries. While it does not offer the same range of capabilities as the Proxmark3 or Flipper Zero, it remains a powerful tool for interacting with numerous RFID systems worldwide. Notably, the MIFARE® standard, which the ACR122U supports, holds a significant presence in the RFID market. According to Netronix, MIFARE® products comply with the international ISO/IEC 14443 standard, which is used in over 80% of all contactless cards globally.[2]

For this reader, I have developed a custom Python tool designed specifically to interact with RFID cards. Although you are free to use alternative tools, the walkthroughs in this training are based on this tool:

https://github.com/VinnieV/crid

In addition to this tool, a few other utilities are required for specific tasks, including mfoc, mfcuk, and milazycracker, but not all tools can be installed on each operating system:

	Windows	Linux	macOS
CRID	✓	✓	✓
mfoc & mfcuk	✗	✓	✓
milazycracker	✗	✓	✗



ACR122U Setup for Windows and Linux

These installation guidelines will be similar for Windows and Linux. For the ACR122U we will install the CRID client to interact with the device, which is a Python CLI package created for the training. On Linux you will also be able to install and use the MIFARE key recovery tools.

[2] https://netronix.pl/en/mifare-the-most-popular-rfid-standard-in-the-world?cms_rewrite=mifare-the-most-popular-rfid-standard-in-the-world

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/installation/

Additionally, on Linux you need to install the following packages from your OS package manager:

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

CRID

https://github.com/VinnieV/crid

You can use pip to install the CRID by running the following commands:

```
mkdir crid
cd crid
python3 -m venv venv
source venv/bin/activate
python3 -m pip install swig crid
```

You can also do pip install from the source directly:

```
git clone https://github.com/VinnieV/crid.git
cd crid
python3 -m venv venv
source venv/bin/activate
python3 -m pip install .
```

Sometimes the crid command will not be available to you within your terminal. If this happens, you can just run the python3 main.py file directly.

MiLazyCracker (Only Linux)

For some exercises and challenges, you will need to use some additional tools. For Linux you can install MiLazyCracker, 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 place 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 macOS Installation

```
brew install swig
mkdir crid
cd crid
python3 -m venv venv
source venv/bin/activate
python3 -m pip install swig crid
```

if you are running macOS version == 14 (Sonoma), Apple added their own CCID driver in version 14, this caused so many issues with smartcard libraries that they actually reverted the change back in version 14.1:

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

If you are running macOS version 14, enable the correct driver by issuing the following command:

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

Requires reboot to work.

For some exercises and challenges you will need some additional tools, but unfortunately the miLazyCracker is not easy to install on macOS. However, with homebrew you are able to install the tools that miLazyCracker will use separately, by just executing the following command:

```
brew install libnfc mfoc mfcuk
```

macOS Debugging

- Unplug and plug the device back in, since sometimes macOS keeps the device busy with another process.
- Might need to install the following USB driver: https://www.wch-ic.com/downloads/CH341SER_MAC_ZIP.html
- Use nfc-list command to see if your reader is detected on your computer.

5. General RFID Theory

RFID (Radio Frequency Identification) uses radio signals to send data wirelessly. 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 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. PICC vs PCD

Within the ISO standards for RFID systems, two key acronyms are used to define the roles of devices involved in RFID communication:

- **PICC (Proximity Integrated Circuit Card):** This refers to the RFID card or tag that responds to the reader's signal. It is the passive element in the communication process, typically powered by the electromagnetic field generated by the reader.
- **PCD (Proximity Coupling Device):** This represents the RFID reader or writer that initiates communication with the PICC. It generates the electromagnetic field required to power and interact with the card.

These definitions are standardized under ISO/IEC 14443, which is commonly used in contactless cards and secure identification systems.


This document describes the following:

— polling for proximity cards or objects (PICCs) entering the field of a proximity coupling device (PCD);

Figure 6 - 14443-3:2018 page 1

Real-World Example: In everyday use, a contactless payment terminal acts as the PCD, while your bank card functions as the PICC.

Understanding these terms is helpful when reviewing documentation about RFID systems, as they help differentiate between the active (PCD) and passive (PICC) components involved in data exchange.

5.2. Active RFID vs Passive RFID

We refer to Active RFID systems when the tags have a power source to generate radio signals, passive means its powered through another medium such as magnetic waves originating from a reader for example. Active RFID operates usually on frequencies like 433 MHz or 2,4 GHz and are usually referred to as beacons and transponders. Beacons will consistently send out a signal on a configured interval where transponders will only respond when it detected a signal from a reader.

5.3. RFID Frequencies

Each RFID card operates on a specific frequency, typically falling into one of three common categories: Low Frequency (LF), High Frequency (HF), and Ultra-High Frequency (UHF). The frequency range determines the card's read range, data transfer rate, and typical use cases.

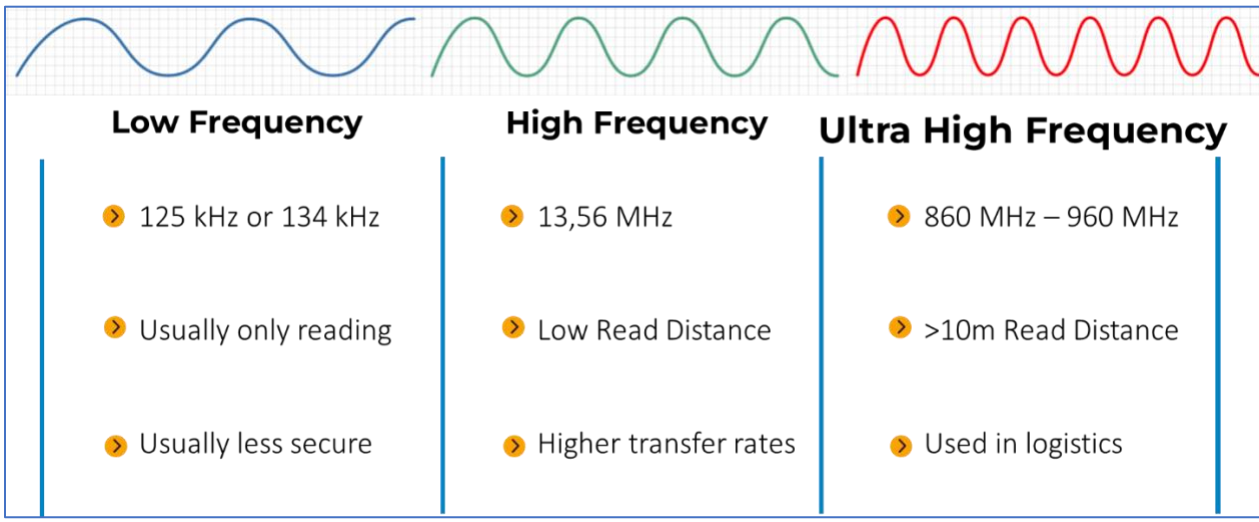


Figure 7 – Different RFID Frequencies

Low Frequency

The Low Frequency (LF) band spans from 30 kHz to 300 kHz, but most LF RFID systems operate specifically at 125 kHz, with some variants using 134 kHz. One of the key advantages of LF systems is their low power requirement, since the protocols are usually really simple and fast, making them ideal for applications where minimal energy consumption is essential. The tags interact with the magnetic field created by a reader.

High frequency

The High frequency (HF) RFID cards almost always operates on the 13,56 MHz frequency with the biggest advantage of having a bigger data transmission speed. The tags interact with the magnetic field created by a reader.

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) since instead of the magnetic field being used, actual electromagnetic radio waves are used to communicate. These read distances are useful for usage at warehouse management for example.

	Frequency	**Typical Read Range**
Low frequency	125kHz 134.2kHz	8 cm

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

5.4. Radio signals/Magnetic Field

For low frequency and high frequency, the distance will be so small that no radio waves are being formed and only the magnetic field will be what matters. Different from Ultra-high frequency where the radio waves are created, and an electromagnetic field is used instead of only the magnetic field. For near field communications in LF and HF, the reader will create a magnetic field with its coil where the tag will obtain that power through its own coil. Similar on how electric transformers work.

A chip powered through an RFID reader can do much more complex calculations than expected, most secure cards use secure cryptographic components with AES encryption that protects the data. Additionally, impressive is WISP, this UHF RFID tag where the microcontroller can be programmed and is solely powered by radio signals reaching distances of around three meters:

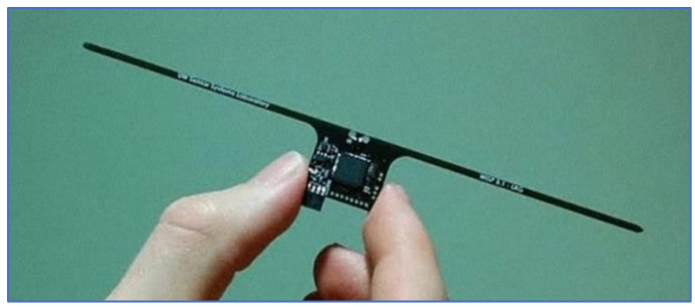


Figure 8 – WISP Radio powered device[3]

5.5. RFID Ranges

With most RFID devices you will see ranges of around 1-10cm, in rare occasions it goes further, for example the following reader, will probably yield around 10-20cm:

Figure 9 - HID MaxiProx Reader

[3] https://clara.nz/docs/research/JVS/Ultralow-noise%20programmable%20voltage%20source/most_viewed_papers_similar_to_this_one/WISP_A_passively_powered_UHF_RFID_tag_wi.pdf

This goes for ranges when presenting a card to a reader and providing it enough power to enable the chip in the card. But you could also attempt to eavesdrop the radio signals from communication happening between a card and a reader. In a paper created by Maximilian Engelhardt, Florian Pfeiffer, Klaus Finkenzeller and Erwin Biebl they showed research about an interesting technique to obtain further eavesdrop ranges by also processing the harmonic frequencies of communication.[4] They managed to obtain ranges of 10 meters.

5.6. Modulation

For high and low frequency RFID (near field), the magnetic field is modulated to transmit data. The type of modulation also depends on whether the PCD or PICC is sending the data. The modulation is different since the PICC cannot create its own magnetic field but changes the consumption of the magnetic field, also called load modulation. This change within the magnetic field usage can be measured by the PCD, allowing the PICC to transfer data to the PCD.

Before the data is modulated, the data will be encoded, encoding can help with error detection and correction, ensuring that the transmitted information remains reliable even in the presence of noise or interference.

Modulation is almost always Amplitude Shift Keying (ASK) but with different variations, for example ISO/IEC 14443 defines two types: Type A and Type B.

Type A:

- Modulation used by PCD: 100% Amplitude Shift Keying (ASK) with Modified Miller encoding.
- Modulation used by the PICC: 847 kHz subcarrier load modulation with OOK (on-of keying) and using Manchester encoding.

In 100% ASK, the carrier signal is completely turned off to represent a binary zero. While this method is simpler and more energy-efficient, it is more susceptible to disruptions in noisy environments due to its sensitivity to electromagnetic interference (EMI).

Type B:

- Modulation used by PCD: 10% Amplitude Shift Keying (ASK) with NRZ-L (Non-Return-to-Zero) encoding.
- Modulation used by the PICC: 847 kHz subcarrier load modulation with BPSK (Binary Phase Shift Keying) with NRZ-L encoding.

In contrast, 10% ASK reduces the depth of modulation, meaning the carrier signal is only slightly altered rather than fully turned off. This approach makes Type B more resistant to electromagnetic interference and ensures continuous power transfer to the RFID tag, which is crucial for stable communication in environments with high interference.

[4] Extending ISO/IEC 14443 Type A Eavesdropping Range using Higher Harmonics
http://rfid-handbook.de/downloads/Systech2013_eavesdropping_Engelhardt_Pfeiffer_Finkenzeller.pdf

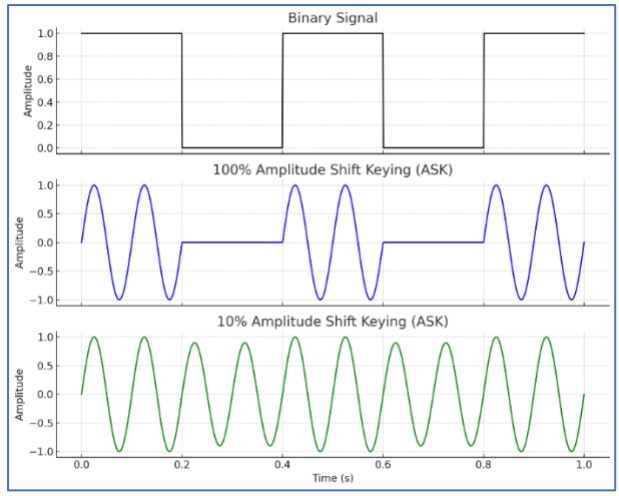


Figure 10 – ASK 100% and ASK 10% graph representation

5.7. RFID Layers

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

- ISO 14443 type A and B
  For contactless smart cards
- ISO 15693
  For vicinity cards with longer ranges
- ISO 18000
  For Air Interface Communications
- ISO 19794
  Biometric

Most common RFID protocol is ISO 14443 which can be dissected into different layers, similar on how the OSI layer exist to help us understand the different layers of networking.:

- **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.

5.8. RFID Protocols

Many different RFID standards and protocols came to existence since the technology became more common within 1990's. Many protocols were created within a time where security issues were less prevalent, and are still used frequently.

RFID Protocol/standard	Date of Invention	Frequency
Low Frequency		
Indala	1980s	125 kHz
HID Prox	1991	125 kHz
Hitag1, Hitag2	1994	125 kHz
Temic T55x7	1998	125 kHz
High Frequency		
FeliCa	1988	13.56 MHz
Legic Prime	1992	13.56 MHz
MIFARE Classic	1994	13.56 MHz
ISO 15693	2000	13.56 MHz
NFC (Near Field Communication)	2003	13.56 MHz
Legic Advant	2003	13.56 MHz
MIFARE DESFire EV2	2016	13.56 MHz
Ultra-High Frequency		
EPCglobal UHF Class 1 Gen 2	2004	860-960 MHz
ISO 18000-6	2004	860-960 MHz

Figure 11 – RFID protocols history table (not exhaustive at all)

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 Proxcard
- Indala

There are also other RFID protocols that have known vulnerabilities:

- MIFARE Classic
    - Nested attack
    - Hardnested attack
    - Darkside attack
- Legic Prime

Security is a crucial element in RFID systems to protect sensitive data, ensure data integrity, and prevent unauthorized access. Weak or outdated protocols not only expose systems to cloning and spoofing but can also serve as entry points for broader security breaches.

When selecting or working with RFID systems, it is essential to evaluate the security measures in place and stay informed about known vulnerabilities.

5.9. Near Field Communication (NFC)

Since many RFID protocols were being developed, a standard was required for the most common use cases to provide consistency between different applications. This is where the Near Field Communication (NFC) standard comes in play, which is essentially a subset of different RFID protocols, 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.

As you can see in the picture below, the NFC standard defines 5 different types:

- NFC A
- NFC B
- NFC F
- P2P (ISO 18092
- NFC V (ISO 15693)

Each type supports different RFID Protocols and standards, making it easy to integrate your use cases and ensure compatibility with other devices.

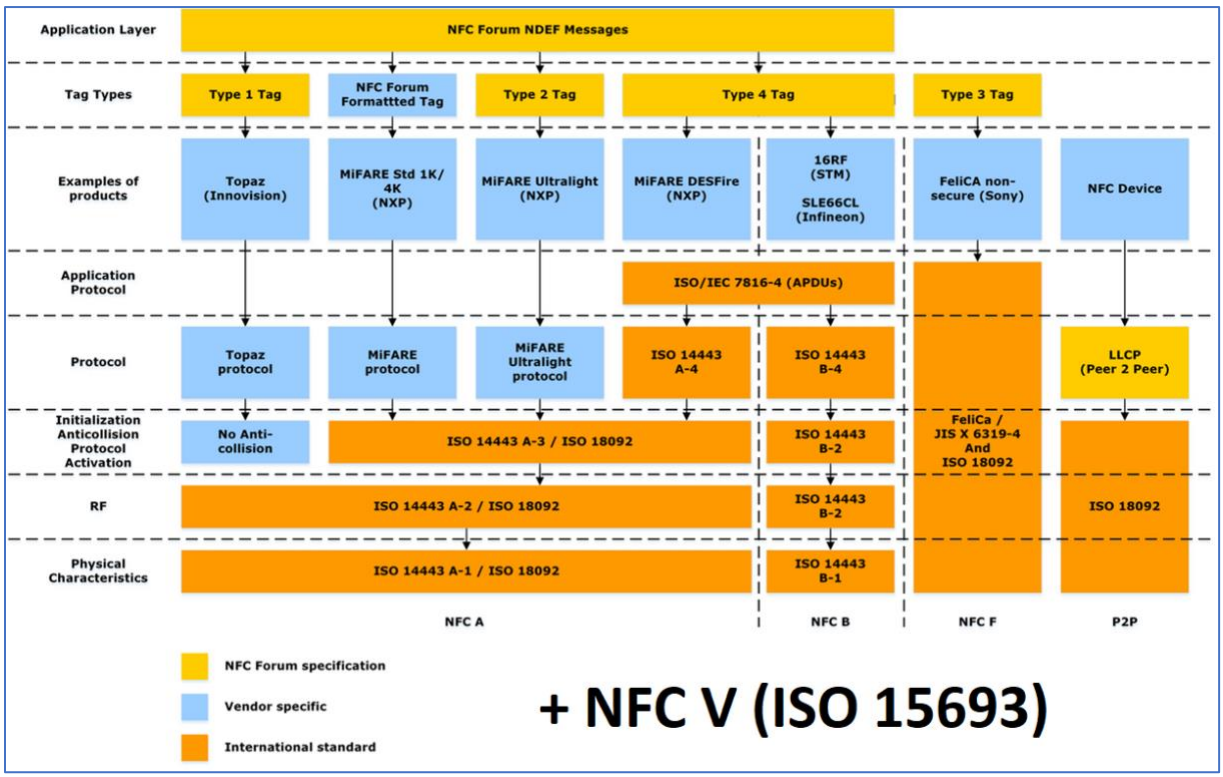


Figure 12 – NFC Standard (Wikipedia)

The NFC standard does not only specify data transfer protocols, but also wireless charging capabilities.

5.10. APDU/Commands

RFID systems operate across multiple layers—from the physical layer to timing protocols, session management, and anti-collision mechanisms. However, for the purpose of this training, our primary focus is understanding the data exchanges required to trigger specific functions within the RFID chip. These exchanges are done through APDU Commands:

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

These are structured, binary-formatted commands exchanged between the RFID reader and the RFID chip. APDUs enable a wide range of functionalities, including, reading and writing to the memory sections, authenticating and executing specific operations supported by the RFID protocol.

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

Load Authentication Keys APDU Command										
Example: FF82000006FFFFFFFFFF										
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

Fortunately, there is comprehensive documentation available regarding APDU commands within official RFID standards. These resources provide detailed insights into how APDU structures function and how they can be used to interact with both the RFID reader and the RFID card.

However, it is important to make a clear distinction between two types of APDU commands found in the documentation:

- **Reader-Software APDUs:**
  These commands are exchanged between your RFID software and the RFID reader. They control the reader's behavior, such as loading authentication keys, configuring settings, or managing communication protocols.
- **Reader-Card APDUs:**
  These commands are sent from the reader to the RFID card to perform actions like reading data blocks, writing to memory, or initiating authentication processes.

Understanding this distinction is crucial. Misinterpreting an APDU intended for the reader-software interface as one meant for the reader-card communication (or vice versa) can result in unexpected behavior or failed operations.

When working with RFID systems, always verify whether an APDU command applies to the communication between your software and the reader or between the reader and the RFID card.

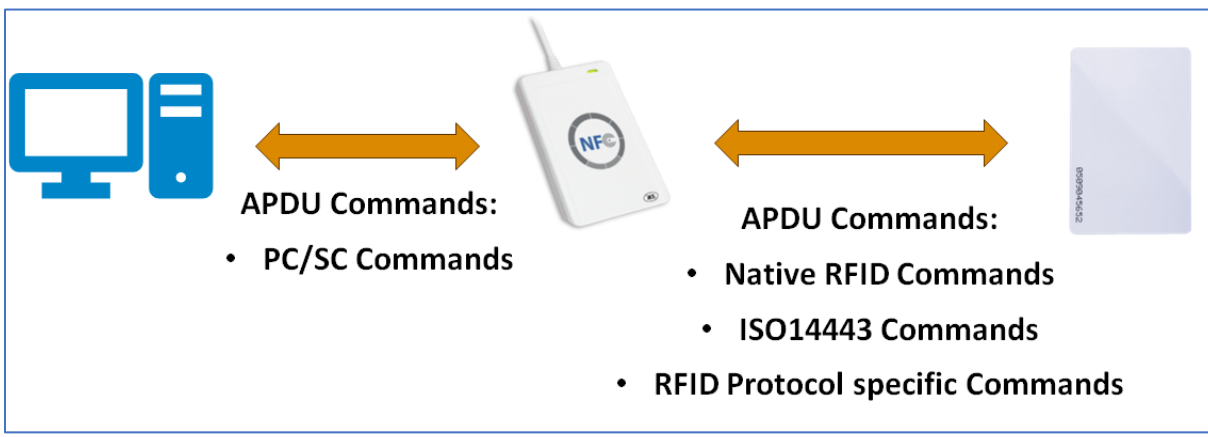


Figure 13 – Different APDU Commands

5.11. Example data flow between reader and card

Before diving into higher-level commands and data exchange, it's essential to understand the initial communication process between an RFID reader and a card. This process ensures that the reader can detect, identify, and establish a connection with the correct card especially in environments where

multiple cards might be present within the reader's field. The steps outlined below follow the guidelines set by ISO 14443 type A Part 3, which defines the card selection process

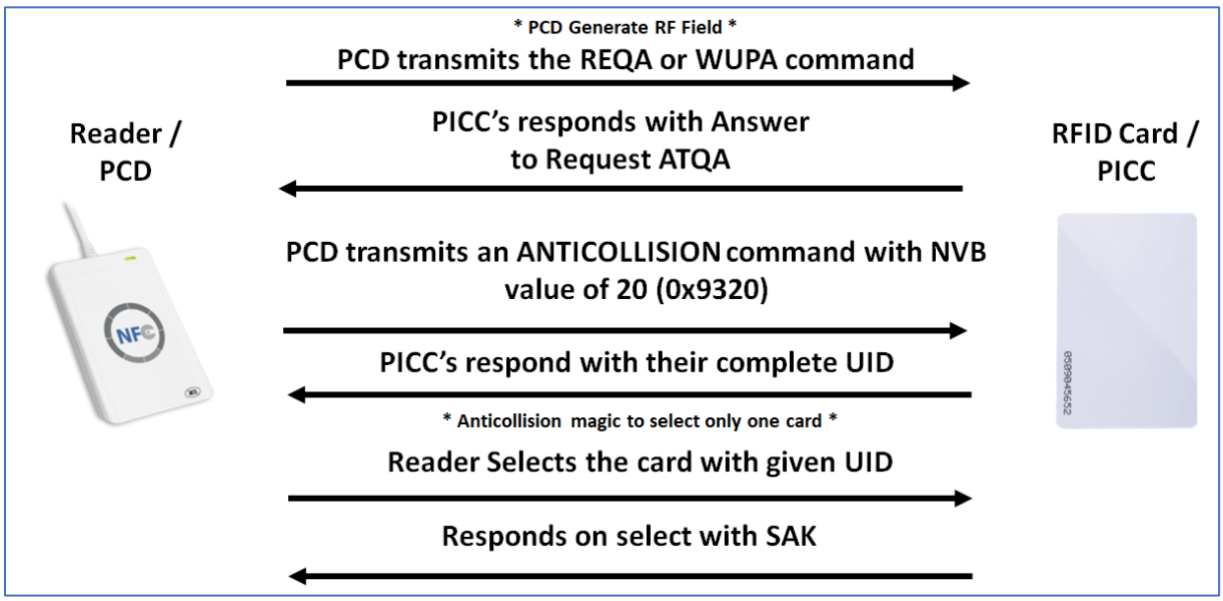


Figure 14 – ISO 14443 type A Part 3 Data Flow

1. **Radio Field Generation:**
   The reader generates a continuous radio frequency (RF) field, creating the electromagnetic energy necessary to power passive RFID cards within its range.
2. **Card Activation:**
   When an RFID card enters the RF field, it becomes powered and can begin processing incoming commands.
3. **Initiating Communication (REQA/WUPA):**
   The reader sends either a REQA (Request Command) or WUPA (Wake-Up Command).
   - REQA is used to check for cards in the idle state.
   - WUPA is used to wake up cards that may be in a halted state.
4. **Card Acknowledgment (ATQA):**
   In response, the card transmits an ATQA (Answer to Request) message, indicating its presence and providing basic information about its capabilities.
5. **Anti-Collision Process:**
   To manage multiple cards within range, the anti-collision protocol is triggered.
   - Each card sends its UID (Unique Identifier).
   - If multiple UIDs cause a collision, the reader can re-request specific parts of the UID until it isolates one card.
   - This process ensures that the reader can select a single card, even in crowded environments.
6. **Card Selection:**
   Once the reader has successfully identified a unique card, it sends a SELECT command to lock communication onto that specific card.

7. **Protocol Activation (SAK Response):**
   The card replies with a SAK (Select Acknowledge) response, confirming the selection. The card then transitions into an active protocol state, ready for further communication, such as authentication or data exchange.

5.12. Protections

RFID systems play a critical role in physical access control, where each registered RFID card serves as a unique identifier, granting individuals access to restricted areas within secured buildings. The most crucial asset in this process is the data stored on the RFID chip, which directly links the card to its authorized user.

To retrieve this data, the RFID reader typically sends an authentication request or password to the card. If an attacker intercepts or extracts this information, they can clone the RFID card and gain unauthorized access to secured areas.

One of the most secure RFID protocols currently available is MIFARE DESFire EV3, which offers secure key exchange and robust data encryption. When implemented correctly, it provides strong protection against common threats like card cloning and unauthorized access.

However, even with advanced encryption, relying solely on the protocol isn't enough. A key strategy for enhancing security is incorporating detection mechanisms that identify unauthorized access attempts. One effective method is using rolling codes or random number generation:

- Each time a user interacts with a reader, a unique random number is generated and stored on both the card and the server.
- During subsequent interactions, the system cross-examines these numbers. If a cloned card is used, the mismatched data will trigger an alert.

This approach not only strengthens protection against cloning but also adds an additional layer of access verification, making it significantly harder for attackers to bypass security.

5.13. Tools

If you are serious about diving into RFID research, the Proxmark3 is your best choice. Known for its versatility and deep protocol support, it offers a wide range of features for both low frequency and high frequency RFID systems. Considering its capabilities, it remains relatively affordable for the depth of analysis it enables.

Another valuable tool is the Flipper Zero, a pocket-sized multi-tool that also supports both low frequency and high frequency RFID protocols. While it may not offer the same advanced features as the Proxmark3, its ease of use and portability make it a great companion for fieldwork and basic RFID tasks.

For more specialized tasks like card emulation, devices such as the Chameleon Mini are highly effective. The Chameleon Mini can store and emulate multiple cards, making it useful for scenarios where you need to mimic specific RFID tags without carrying physical cards.

Lastly, simpler USB RFID readers like the ACR122U, which is used in this training, offer a straightforward way to interact with many high frequency systems. While the ACR122U focuses on

13.56 MHz protocols, other variants exist that support additional frequencies and standards, giving you flexibility depending on your project needs.

6. HID Proximity Cards

HID Proximity Cards are a technology developed by HID Global, a company with a wide range of access control solutions. These cards operate at 125 kHz (Low Frequency) and are widely used for physical access control systems in corporate, government and institutional environments.

The HID ProxCard was the first version released in 1991, with some different variations released in the upcoming decades. Throughout the 1990s and early 2000s, HID expanded its product line with models like the ProxCard II, ISOProx II, and DuoProx II, offering enhanced durability and form factors. As security concerns grew, HID introduced iCLASS smart cards in 2002, operating at 13.56 MHz with encrypted communication to improve security over the original 125 kHz Prox technology. In 2011, HID launched iCLASS SE and Seos smart cards, designed to provide even stronger encryption, mobile credential compatibility, and cloud-based access control integration

The ProxCard II is a clamshell-style card with a thickness of approximately 1.8 mm, making it thicker than standard credit-card-sized RFID cards. In contrast, the ISOProx II and DuoProx II cards conform to the ISO 7810 standard, with a nominal thickness of 0.76 mm, similar to standard credit cards.

Figure 15 – HID Proxcard vs ISOProx

HID cards and readers commonly display the logo, which can provide valuable information during reconnaissance efforts:

Figure 16 – HID Readers and Tags

6.1. HID iClass

HID iCLASS is a high-frequency (13.56 MHz) RFID smart card technology and features mutual authentication, encrypted communication, and secure data storage, using proprietary encryption based on a diversified key system. However, security vulnerabilities have been discovered, with researchers demonstrating that older iCLASS (non-SE) cards can be cloned or cracked, leading HID to introduce iCLASS SE and Seos, which offer stronger encryption and enhanced security features.

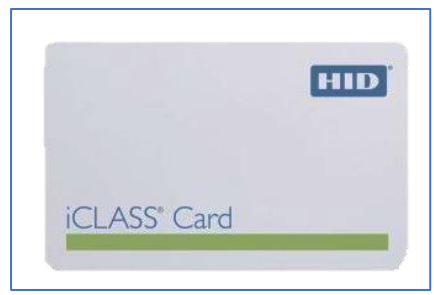


Figure 17 – iCLASS Card

6.2. Modulation

HID Proximity Cards operate at a 125 kHz frequency and contain a unique identifier (ID) that is read by proximity card readers. These cards use On-Off Keying (OOK), a specific form of Amplitude Shift Keying (ASK) modulation.

6.3. Data Transmission

When powered by the electromagnetic field generated by the reader, the HID Proximity card responds by continuously transmitting its unique ID. The simplicity of this process makes these cards highly reliable, but it also introduces a significant security weakness because the data is transmitted without encryption, leaving it vulnerable to cloning and unauthorized access.

The only inherent protection for HID Proximity cards is that they are read only. This means that, under normal circumstances, an attacker cannot directly overwrite or modify the card's data. However, this defense is easily bypassed using programmable cards like the T5577, which can emulate HID Proximity cards by copying their unique identifiers. These rewritable cards allow the cloning of HID cards, as they can be configured to transmit the same ID as the original.

Unlike more secure RFID systems that require authentication or passwords to read stored data, low frequency cards like HID Proximity cards simply transmit a number when in range. This makes it relatively easy for an attacker with a basic RFID reader to intercept the ID and create a functional clone, especially when in close proximity to the original card.

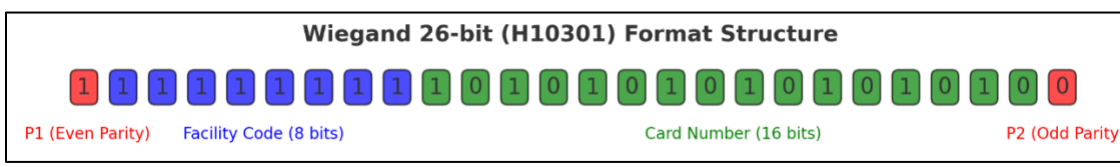
6.4. Data Formats

The data read from HID Proximity cards can be interpreted into various formats, depending on the configuration. These formats represent different ways of structuring the binary data from the card and usually have a facility code assigned and a card number with some parity bits, a brief list of available formats:

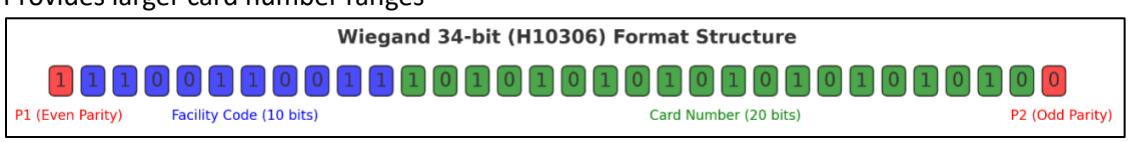
Format	Bit Length	Facility Code	Card Number	Parity Bits
H10301	26-bit	8 bits	16 bits	2 bits
H10302	37-bit	16 bits	19 bits	2 bits
H10304	37-bit	16 bits	19 bits	2 bits
H10306	34-bit	10 bits	20 bits	2 bits
H10320	36-bit	4 bits	31 bits	1 bit
H10323	37-bit	8 bits	24 bits	5 bits
H10325	35-bit	15 bits	19 bits	1 bit
H10328	48-bit	16 bits	32 bits	N/A

Common formats are:

- 26-bit Wiegand (H10301)
  Most common format, has a facility code range of 0-255 and sequence number range of 0-65535.



- 34-bit Wiegand (H10306):
  Provides larger card number ranges



Knowing how the RFID system processes the number is not required. Knowledge of the binary number transmitted by the card is enough to clone the card, but knowing the pattern might help in enumerating other cards within the system.

6.5. Wiegand

"Wiegand" refers to specific topics in low-frequency RFID. It involves both data formats, where card numbers decode into Wiegand codes, and the protocol used from the reader to the access control system.

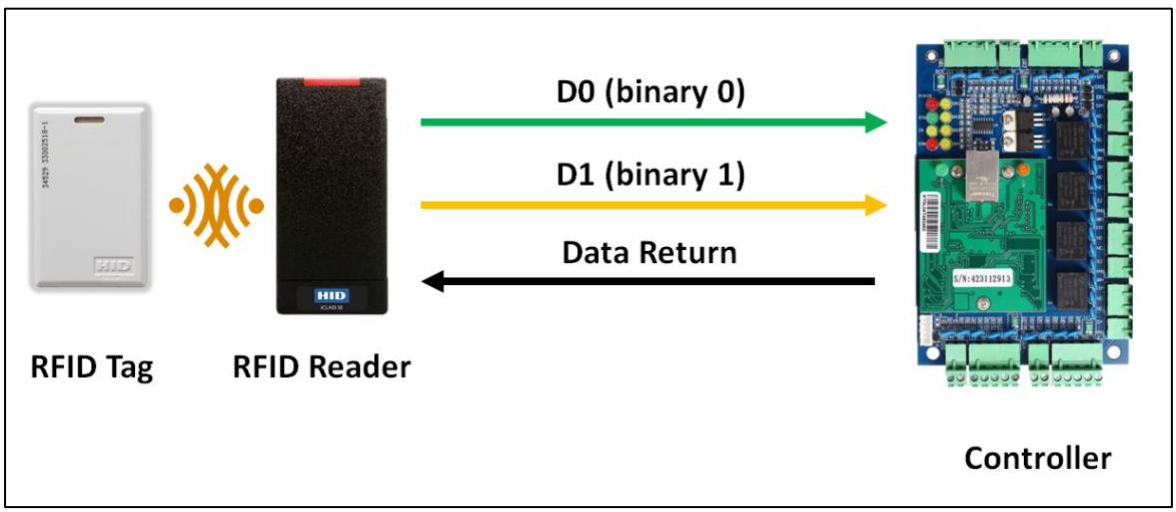


Figure 18 – RFID Reader & Controller Wiegand Setup

Wiegand uses two wires for data communication:

- Data0 (D0)
- Data1 (D1).

Data is transmitted as a sequence of pulses:

- A pulse on the D0 line represents a binary 0.
- A pulse on the D1 line represents a binary 1.

The absence of a pulse on either line is interpreted as no data being transmitted at that moment.

There are tools that can tap the lines between an RFID reader and controller to intercept the Wiegand traffic. A popular tool for this is the ESPKey:

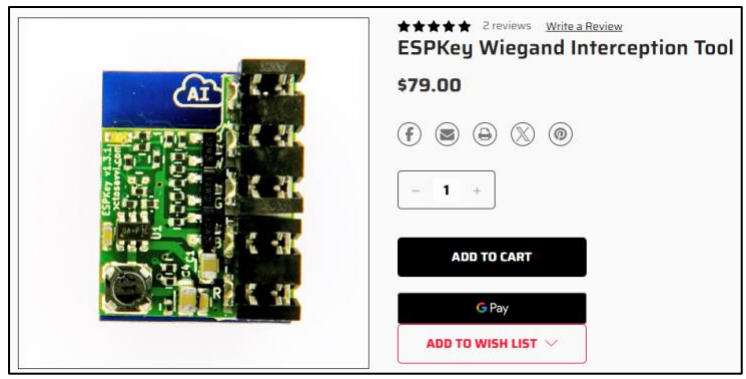


Figure 19 – ESPKey Wiegand

It has a built-in Wi-Fi controller to interact with the device and has various functionalities to help you extract credentials and obtain physical access.

6.6. T55xx

The T55xx series of RFID chips, including models like the T5557, T5567, and ATA5577, were originally developed by Atmel Corporation, a company known for its semiconductor products. In 2016, Atmel was acquired by Microchip Technology Inc., which now oversees the production and distribution of these chips. These cards are highly versatile and programmable and are often used to clone HID Proximity cards because they can be programmed to mimic the exact modulation, encoding, and ID of an original card.

7. MIFARE Classic

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.

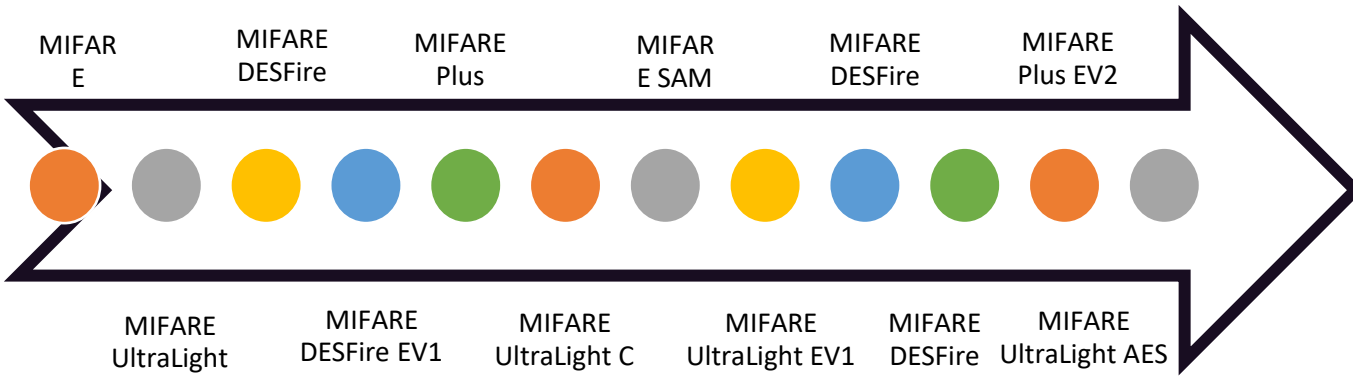


Figure 20 – MIFARE Timeline 1994-2023

The MIFARE Classic cards are fundamentally just a memory storage device, where the memory is divided into segments and blocks with and protected with 6-byte keys. 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.

MIFARE Classic 1K offers 1024 bytes of data storage divided in 16 sectors.

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.

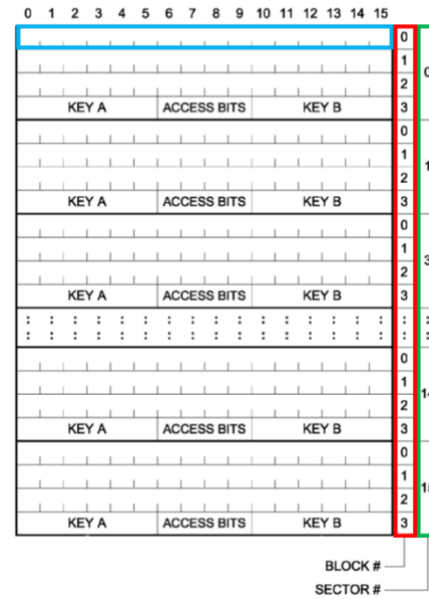
Every sector is protected by two different keys, called A and B. Each key can be programmed to allow operations such as reading, writing, increasing value blocks, etc. The MIFARE Classic uses Crypto-1 protocol for authentication and ciphering. MIFARE Classic 1K and MIFARE Classic 4K are the most common type of cards being used, and are also commonly referred to S50 or S70 respectively:

	Bruto Data storage	Sectors	Netto Data Storage
MIFARE Classic Mini	320 bytes	5	224 bytes
MIFARE Classic 1K (S50)	1024 bytes	16	752 bytes
MIFARE Classic 4K (S70)	4096 bytes	40	3440 bytes

Figure 21 – MIFARE Memory Capacities

7.1. 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 also called the manufacturer block, which contains the UID of the card and other card configuration settings. 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[5]. 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 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 and dangerous to calculate manually, one miscalculation and you can lock out the sector. Use the Proxmark3 command hf mf acl or online MIFARE Access bits calculators to obtain the correct access bits.

Using the following table you can decide the permission row that you would want to use for the sector. Then a formula can be used to convert these permissions into a 3 byte value called the Access Bits:

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

$C1_0$	$C2_0$	$C3_0$	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, the small x indicated block number:

	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, but this was just a demonstration.

Usually, key A has read access on certain blocks and is typically used by the RFID readers exposed to the users, such as a door lock or subway ticket door. Key B will be the administrator key with full read and write permissions, typically used in a more secure environment such as the registration desk. This is useful if you want to store the read key in an exposed reader but the write key at the badge maintainer/administrator.

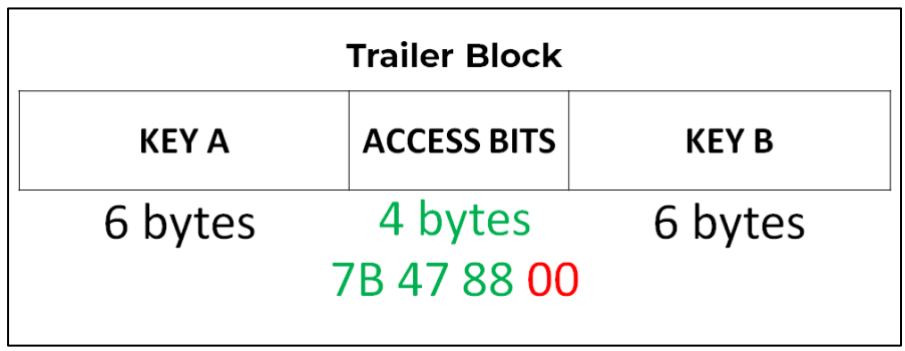
MIFARE Application Directory (MAD)

The MAD is a structure which can be used to allocate different applications to the different sectors of MIFARE cards. You can compare it with a partition table for a hard drive. You can find the MAD structure documented within the documentation provided from NXP called: AN10787, if you google this you will be able to download the documentation.

There are three versions

- MAD1
  Limited to 16 sectors
- MAD2
  For MIFARE Classic 4K and MIFARE Plus
- MAD3
  For MIFARE DESFire

The version of the MAD is defined within the General Purpose Bit within the access bits. Which is the fourth byte of the access bits:

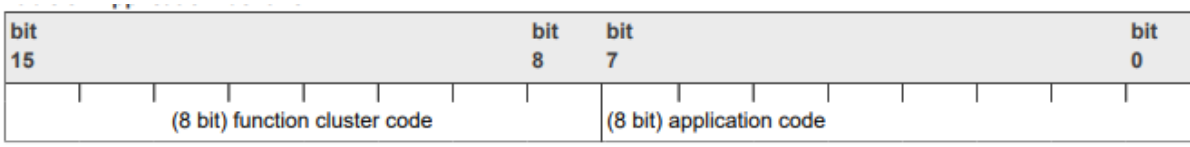


It is important to note that key A should be A5A4A3A2A1A0 with read-only access so that any system can process the MAD and key B should be the key that has write permissions to allocate new applications.

MAD1 would look like the following for Sector 0:

byte 15	byte 14	byte 13	byte 12	byte 11	byte 10	byte 9	byte 8	byte 7	byte 6	byte 5	byte 4	byte 3	byte 2	byte1	byte 0
m	a	n	u	f	a	c	t	u	r	e	r	c	o	d	e
AID for sector 0x07		AID for sector 0x06		AID for sector 0x05		AID for sector 0x04		AID for sector 0x03		AID for sector 0x02		AID for sector 0x01		info	CRC
AID for sector 0x$F		AID for sector 0x$E		AID for sector 0x$D		AID for sector 0x$C		AID for sector 0x$B		AID for sector 0x$A		AID for sector 0x09		AID for sector 0x08	
s	e	c	t	o	r	t	r	a	i	l	e	r	0x	0	0

There are two bytes allocated for sectors 1-15, the two bytes represent an AID or Application Identifier. The application Identifier is a two-byte code consisting of a function cluster code of one byte and an application ID of one byte:



The function cluster code contains some categorization of the application:

Table 16. Function cluster codes

cluster code (hex)	function
00	card administration
01-07	miscellaneous applications
08	airlines
09	ferry traffic
10	railway services
11	miscellaneous applications
12	transport
14	security solutions
18	city traffic

7.2. MIFARE Classic Commands

MIFARE Classic adds a few specific commands that are available for the reader to use to authenticate, read and write to the cards.

ATQA/SAK/NAK

ATQA and SAK is also within the ISO 144443 standard, but MIFARE extends it a little.

The ATQA, which is a response from the card when the reader sends a REQA or WUPA command, contains which type of MIFARE RFID chip is used:

Chip Type	Common Name	ATQA
MF1S500yX	MIFARE Classic 1K	**0x0044**
MF1S503yX	MIFARE Classic 1K EV1	**0x0004**
MF1S700yX	MIFARE Classic 4K	**0x0042**
MF1S703yX	MIFARE Classic 4K EV1	**0x0002**
MF1ICS50	MIFARE Classic 1K	**0x0044**
MF1ICS70	MIFARE Classic 4K	**0x0042**
MF1ICS50 EV1	MIFARE Classic 1K EV1	**0x0004**

MF1ICS70 EV1	MIFARE Classic 4K EV1	**0x0002**

CRC

The CRC is a two-byte value that is used for multiple commands where a checksum is created based on the command transmitted. The MIFARE documentation references the ISO 14443-3 Standard as to how to calculate the CRC. This CRC uses an initial value of 0x6363 and uses the polynomial 0x1021. Example code for Python3 presented below:

```
def iso14443a_crc(data: bytes) -> bytes:
    """
    Calculate the CRC_A as specified in the ISO 14443-3 standard.

    :param data: Input data as bytes
    :return: CRC as two bytes (little-endian)
    """
    crc = 0x6363  # Initial value

    for byte in data:
        # XOR-in next input byte
        crc ^= byte
        # Process each bit in the byte
        for _ in range(8):
            if crc & 0x0001:  # If LSB is set
                crc = (crc >> 1) ^ 0x8408  # Polynomial reflected
            else:
                crc >>= 1

    # Return the CRC in little-endian format
    return bytes([crc & 0xFF, (crc >> 8) & 0xFF])
```

Not all commands require sending a CRC sequence at the end, but many require it.

Crypto1

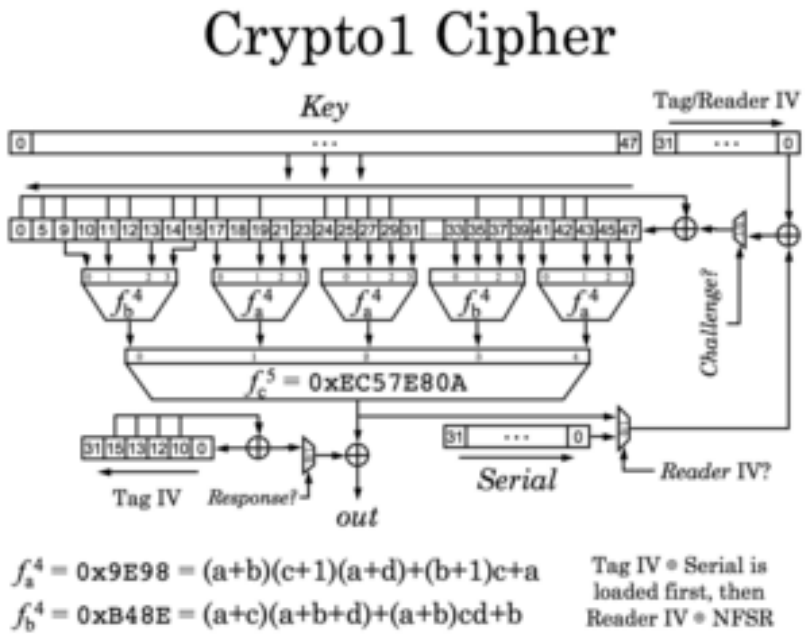


Figure 22 – Crypto1 Cipher illustration

MIFARE classic implements the proprietary Crypto1 encryption, it was kept secret for a long time until German researchers (Henryk Plötz and Karsten Nohl[6]) delayered the chip and reversed the protocol. It was discovered that the encryption cipher uses 6-byte keys and creates a stream cipher originating from a 48-bit Linear Feedback Shift Register (LFSR) that is used to XOR the plaintext value with. The encryption is used within the authentication sequence to verify if both parties are using the correct key and used to encrypt subsequent traffic. It's currently implemented within various libraries and tools.

MIFARE Authenticate Sequence

In order to setup an encrypted channel and verify if both parties are using the same access key, the protocol perform an authentication sequence where the Crypto1 cipher is used to encrypt a random nonce. The sequence involves four steps:

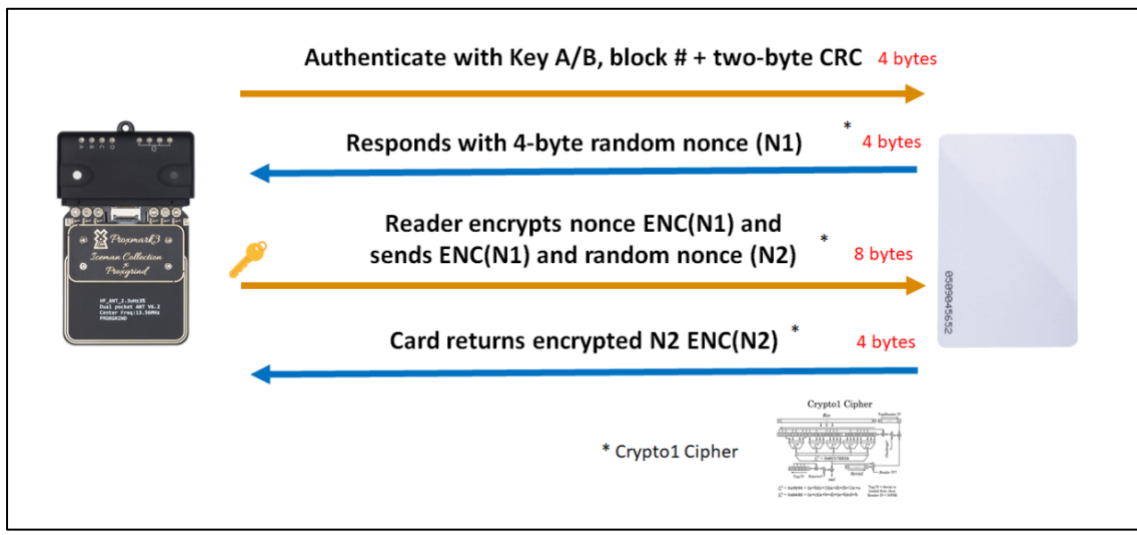


Figure 23 – MIFARE Authentication Sequence

1. The first step is performed by the reader where it send the authenticate command towards the card, specifying the key type (A/B) and the block number with a two-byte CRC:

Authenticate Command			
1	2	3	4
Key Type	Block Number	CRC	
60 (Key A) 61 (Key B)	00-FF	XX XX	

2. The card responds with a 4-byte random nonce

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

3. The reader uses the nonce and Crypto1 cipher to derive a similar nonce and encrypt it using the correct key
4. The card verifies if everything is encrypted successfully and sends back the encrypted nonce from the reader, which the reader can use to verify the card

Once the authentication is successful the Crypto1 states of both the reader and the card will be the same which can be used to encrypt all subsequent transactions, such as reading and writing to the card.

Reading/Writing to a card

Before reading can occur, the reader needs to perform the authentication sequence listed above. Once this has been completed, the reader can send the 4-byte read command to the card, specifying the block number:

Read Command			
1	2	3	4
Command	Block Number	CRC	
30	00-FF	XX XX	

If the authentication allowed access to the block, the card would respond with 16 bytes of data and a 2-byte CRC.

Similar for writing to card, although writing happens in two steps. The first command for initiating the write process is to send the write command, specifying the block number:

Write Command (part 1)			
1	2	3	4
Command	Block Number	CRC	
A0	00-FF	XX XX	

The card will respond with an acknowledgement if the card permits to write to the specified block. The reader can then initiate the second part of the write command where it sends the 16 bytes with a 2-byte CRC at the end:

Write Command (part 2)																	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Data																CRC	
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX																XXXX	

If writing was successful the card will responds with an acknowledgement.

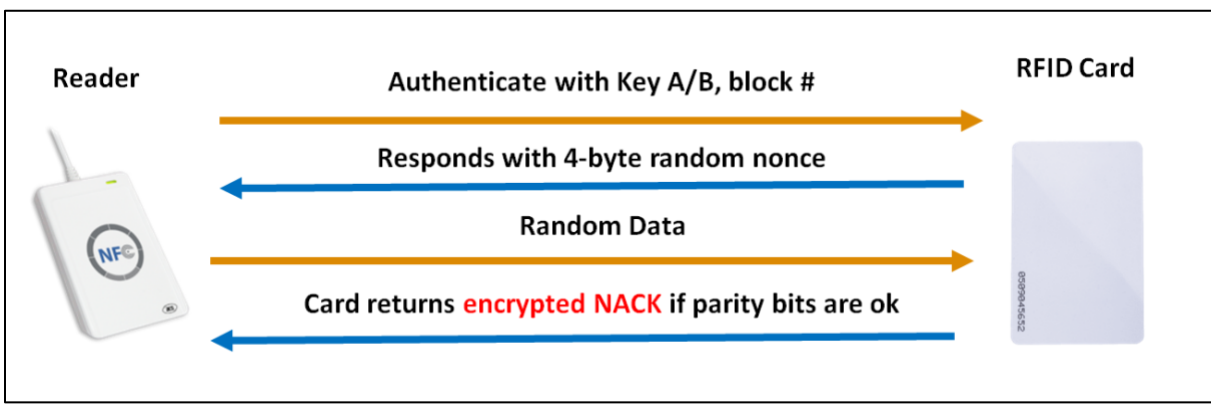
7.3. MIFARE classic vulnerabilities

During the research of Henryk Plötz and Karsten Nohl, they identified multiple flaws with the Crypto1 cipher. They noticed that the key length was too small to be cryptographically strong enough. But more importantly while running some tests they discovered that the random number generator was not random enough. This research and that of others have led to a few attacks to recover the access keys of the card:

1. Darkside attack
2. Nested attack
3. Hardnested attack
4. Staticnested attack

Darkside attack

The Darkside attack was the first attack discovered for MIFARE Classic. It will only work on older cards since the issue has been solved with newer cards. During the authenticate sequence, the card responds with a 4-byte nonce and expects some data back, with this attack, the reader will send random data back after receiving the nonce. When the parity bits happen to align when the card decrypts the random data, the card will return with a 4-bit encrypted NACK response. When collecting 8 or more of these encrypted NACK messages, you can recover the key that was used:

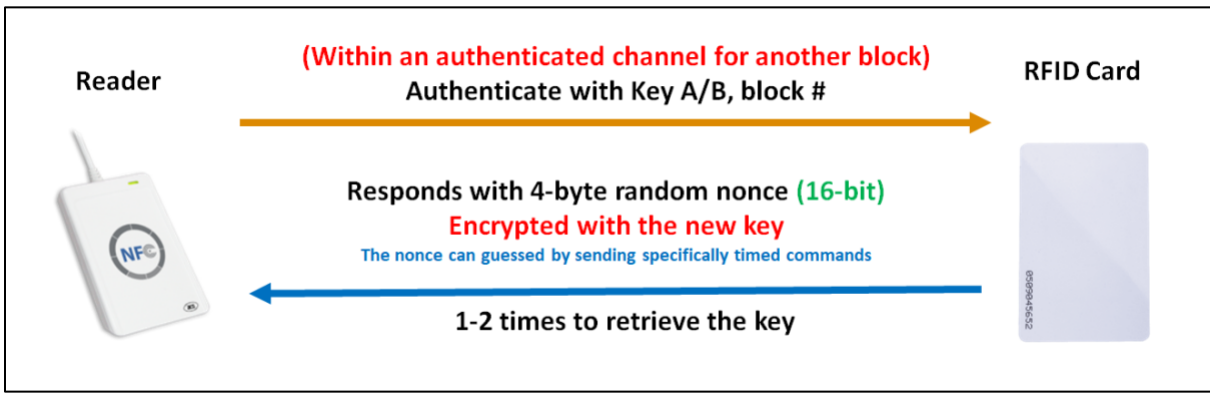


This issue exists because the parity bits are included within the plaintext, meaning there is 1/256 chance that the parity bits will be correct, triggering this issue. With some cryptographic calculations you can increase the chances significantly.

Nested attack / Hardnested / Staticnested attack

The nested attack requires one valid key for any sector, since the attack exploits a weakness when inside an already established authentication channel a new authentication occurs. The attack works by sending an authentication command for your target block inside an authenticated channel for a block you have the key for. The card will respond with a 4-byte random nonce encrypted with the key used for the sector. Due to weaknesses in the card where the 16-bit LFSR halved the randomness of the nonce and due to the weak randomness when timing the attack, it's possible to predict what the

nonce value will be. Resulting in you having an encrypted 4-byte ciphertext, where you know the plaintext of which can be used to recover parts of the key.



Hardnested is a similar attack that can be done on cards where the nested attack is fixed. It exploits a similar issue, however it is much slower since it requires you to send 1000-3000 commands to the card.

Staticnested attack is specifically for a type of card where all other attacks are remediated. The nonce is statically generated using the UID of the card, the sector you are targeting, and access key being used. Meaning if the same access key is used within other sectors on the same, it could be possible to recover the key.

FUDAN Backdoor

The FM11RF08S RFID card from Fudan was considered one of the most secure options in its class, with all known vulnerabilities addressed except for the staticnested attack. However, security researcher Philippe Teuwen discovered a backdoor key embedded within the chip that allows unauthorized access to the card's memory, bypassing the need for any specific sector access keys.

This discovery revealed a significant flaw in the card's security model, as the backdoor key enables full read access to the card's contents without triggering standard security mechanisms. Teuwen's findings highlight the importance of thoroughly vetting security features, even in systems believed to be hardened against known attacks.

For a deeper understanding of the vulnerability and the research behind its discovery, the full paper is an insightful read:

https://eprint.iacr.org/2024/1275.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 |
```

Figure 24 – Proxmark3 Traffic from gen1 Magic MIFARE

RFID Training Challenge Bundle

by Vanhoecke Vinnie @ Brucon 0x11

8. Table of content

9. Table of content 54

10. Introduction 55

10.1. Training RFID Cards 55

11. HID Proximity Cards 56

11.1. HID Proximity Cards Exercises 56

Exercise #0: LF Read 56

Exercise #1: T5577 57

11.2. HID Proximity Cards Challenges 58

Challenge #1: ATM 1 58

Challenge #2: ATM 2 61

12. MIFARE Classic 62

12.1. MIFARE Classic Exercises 62

Exercise #0: Identify 62

Exercise 1: PIN Code 67

Exercise 2: Username 72

Exercise 3: Access Bits 77

Exercise 4: Brute Force 82

Exercise 5: Key Recovery 87

Exercise 6: Key Recovery 2 93

Exercise 7: Magic MIFARE 96

12.2. MIFARE Classic Challenges 101

Challenge 1: Vault 101

Challenge 2: Employee Portal 105

Challenge 3: Vending Machine 110

Challenge 4: Hotel Rooms 115

Challenge 5: Speedrun 121

9. Introduction

Welcome to the Solutions bundle of the Ethical Hacking RFID Training. This document describes the exercises and challenges that need to be performed during the training and includes solution walkthroughs for the Proxmark3, Flipper Zero & ACR122U. Use it wisely, avoid just copying commands from the solutions bundle.

9.1. Training RFID Cards

You have received a couple of RFID cards:

- Card A: Spencer (Magic MIFARE Classic 1K)
- Card B: Harper (Magic MIFARE Classic 1K)
- Card C: RFID Speedrun Challenge Card Yu-Gi-Oh Version (Magic MIFARE Classic 1K)
- Card C: RFID Speedrun Challenge Card Pokemon Version (Magic MIFARE Classic 1K)
- Card D: Banker card (T5577)
- Card E: Secure Card (Magic MIFARE Classic 1K)

10. HID Proximity Cards

10.1. HID Proximity Cards Exercises

Easy	**Exercise #0: LF Read**
Devices	
Training Cards	**Card D**
Description	
For this exercise, perform a simple read of card D and review its Wiegand number format. **Proxmark** When running the Proxmark3 CLI you can initiate the **lf hid reader** command to get the data stored on the card. Run this command on Card D. **Flipper Zero** Make use of the 125 kHz RFID Module on the Flipper 	
References	
Proxmark	Make use of the lf hid reader command
Flipper Zero	Make use of the 125 kHz RFID module

Easy	Exercise #1: T5577	50 POINTS
Devices		
Training Cards	**Card F**	

Description

For this first exercise on HID Cloning, we are going to clone a card from a given 26-bit Wiegand number:

Facility Code: 20
Card Number: 1337

Program this 26-bit Wiegand number into Card F (T5577) and validate your steps on the OctoBox.

Proxmark

To write the number on Card F, you can use the command called **lf hid clone**, review the help menu to understand which parameter you would not to provide to clone the number in the correct format:

```
usage:
    lf hid clone [-h] [-w <format>] [--fc <dec>] [--cn <dec>] [-i <dec>] [-o <dec>] [-r <hex>] [--q5] [--em]
                 [--bin <bin>]

options:
    -h, --help                  This help
    -w, --wiegand <format>      see `wiegand list` for available formats
    --fc <dec>                  facility code
    --cn <dec>                  card number
    -i <dec>                    issue level
    -o, --oem <dec>             OEM code
    -r, --raw <hex>             raw bytes
    --q5                        optional - specify writing to Q5/T5555 tag
    --em                        optional - specify writing to EM4305/4469 tag
    --bin <bin>                 Binary string i.e 0001001001
```

The help command also lists examples that can aid you with configuring the correct number and format on the T5577 card.

Flipper Zero

On the Flipper Zero you can emulate the card instead of modifying the T5577 card.

References

Proxmark	Make use of the lf hid commands
Flipper Zero	Make use of the 125 kHz RFID module

10.2. HID Proximity Cards Challenges

Easy	Challenge #1: ATM 1	50 POINTS
Devices		
Training Cards	**Card D and F**	
Description		
Can you gain access to the ATM by cloning the Badge of the bank employee? Thus clone Card D onto Card F.		

Walkthrough: HID Clone

Proxmark3

You can read the HID card by performing the lf hid reader command:

```
lf hid reader
```

This will provide you with the following output:

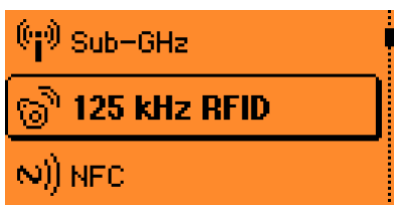
```
[usb] pm3 --> lf hid reader
[+] [H10301   ] HID H10301 26-bit
[+] [ind26    ] Indala 26-bit
[=] found 2 matching formats
[+] DemodBuffer:
[+] 1D5559555566A69999655566

[=] raw: 00000000000002005daa405
```

The raw field represented there can be used to create a cloned card on a T5577. Luckily card C is available as this type of card.

```
lf hid clone -r 2006ec0c86
```

Flipper Zero

Then one card will be left and try reading it the 125kHz RFID module:



This should give you the following result:

Within the More options you can write the data to Card C.

5YOA

This is a cheap Low Frequency RFID cloner from aliexpress for around 10 euro.

How to clone a Prox HID card:

1) Make sure two AAA batteries are inserted correctly in the back of the device
2) Power on the device with the switch on the side
3) Hold the card you want to scan against the device
4) Press the read button
5) A beep should occur
   and PASS light should be lighting green now.
6) Hold the device close to the target card you want to clone to
7) Press the write button
8) Beep should occur and the card should be cloned.

Hard	Challenge #2: ATM 2	200 POINTS
Devices		
Training Cards	Card F	

Description

Through some OSINT we identified an old backup file containing the following information:

Name	Role	Facility Code	Card Number
John Smith	Manager	110	100245
Emily Davis	Supervisor	120	100246
Michael Brown	Engineer	100	100247
Sophia Wilson	CFO	150	100248
David Martinez	Analyst	100	100249
James Anderson	Technician	100	100250
Olivia Taylor	VP Operations	130	100253
Liam Johnson	Security Officer	100	100254
Ava White	IT Admin	140	100255
Ethan Thomas	CEO	XXX	XXXXXX

Can you figure out, how to get access to the account of the CEO?

11. MIFARE Classic

11.1. MIFARE Classic Exercises

Exercise #0: Identify

Difficulty: Easy

Devices: 

Training Cards All training cards can be used for this challenge

Description

When assessing and testing RFID systems, the first thing you do is identify what RFID protocol the card operates on. Usually this requires a low and high frequency antenna to identify all types of RFID protocols. Identify the RFID types of all training cards.

Proxmark

Make sure you have the Proxmark3 client running, and you flashed the same version as the client you are using. Run the Proxmark3 CLI on your laptop by executing the **./pm3** file in your local Proxmark3 source code folder.

For high frequency cards you can use the command called **hf search**:

```
[usb] pm3 --> hf search
```

For low frequency cards you can use the command called **lf search**:

```
[usb] pm3 --> lf search
```

Flipper Zero

Flipper Zero can read both low and high frequency tags.
Use the NFC module on your flipper to detect high frequency cards:

Use the 125 kHz RFID Module for low frequency cards:

ACR122U

The ACR122U is an NFC compliant device and has a high frequency antenna ($13{,}56_{Mhz}$) this makes it possible to identify all kinds of RFID protocols included in the NFC bundle with the ACR122U. In order to use the CRID tool to identify cards with the ACR122U, you can issue the following command:

```
crid --identify
```

Keep in mind that not all cards can be identified with this reader, the card that can't be identified will only be used for one challenge. so it does not impact your experience much. The instructor also has some simple LF readers for that challenge.

Bonus:
Test any other personal cards you may have laying around.

References

Proxmark3 Command Cheat Sheet	https://github.com/RfidResearchGroup/proxmark3/blob/master/doc/cheatsheet.md

Walkthrough: Identify

Proxmark3

The first two commands you use with the Proxmark3 to identify the type of card are:

- lf search
- hf search

Then perform the following steps to obtain

1. Place one of the RFID cards on the Proxmark
2. Make sure you run the commands within the pm3 terminal client when connected to your computer.
3. Execute the following command to scan for High Frequency Cards:
   hf search

```
[usb] pm3 --> hf search
[/] Searching for ISO14443-A tag...
[+]  UID: D9 7E 73 4F
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
```

4. If no High Frequency cards were found, execute the following command:
   lf search

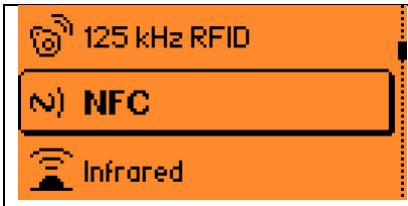
```
[usb] pm3 --> lf search

[+] Valid HID Prox ID found!
```

These commands will enumerate over various RFID protocols for its given frequency and will try to identify any RFID cards and its protocol. The search command is only doing a limited enumeration to recognize the type of card. Usually if you want to explore more into the specifics of the card you need go into the protocol level commands.

You should have identified two high frequency cards, both MIFARE Classic 1K and one low frequency card of type HID Prox ID.

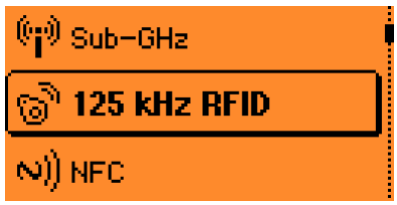
Flipper Zero

The Flipper Zero has two modules, NFC and 125kHz so we will have to run both separately. Lets start with trying out the NFC module of the flipper on all cards:

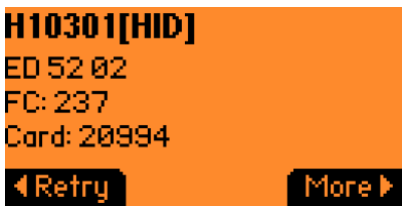


When choosing the read option within the NFC module and holding it against the training cards you will find that both Card A and B identify as MIFARE Classic 1K and you would receive a screen similar as these:

Then one card will be left and try reading it the 125kHz RFID module:



This should give you the following result:



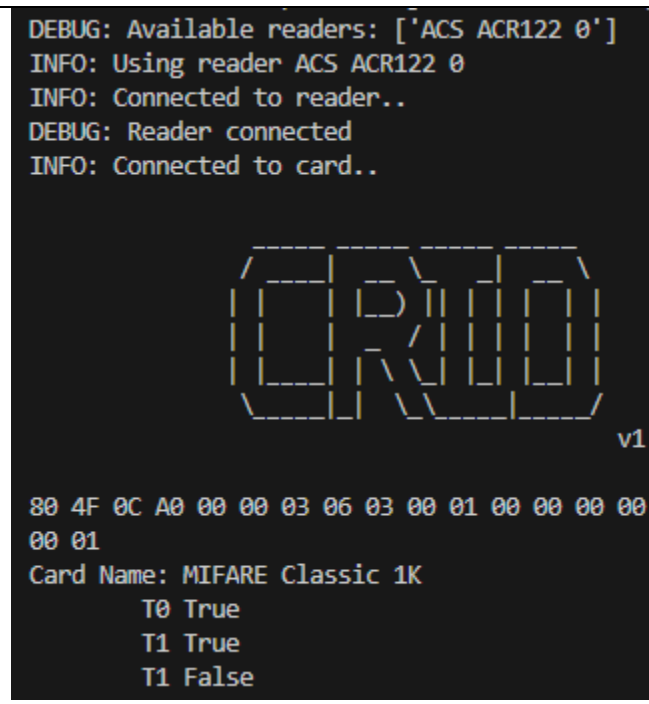
Indicating that an HID card has been identified with format H10301.

ACR122U

Execute the following command to identify high frequency cards with the ACR122U:

```
crid --identify
```

Which should output the following information:



You should have identified two high frequency cards, both MIFARE Classic 1K and one low frequency card of type HID Prox ID.

Easy	Exercise #1: PIN Code	50 POINTS
Devices		
Training Cards	**Card A or B, Sector 1**	

Description

As a first real exercise we will attempt to read from a MIFARE Classic card. Specifically, a PIN code required to get access to the points for this challenge. Figure out how it's stored within Sector 1 on Card A or B. For this exercise the access keys will be provided:

Access Key A: FFFFFFFFFFFF
Access Key B: FFFFFFFFFFFF

Proxmark

Reading MIFARE Classic card with the Proxmark3 can be done using the hf mf commands, in particular the **hf mf rdsc** can be used to read one of the sixteen sectors of a MIFARE Classic 1K card. To explore all MIFARE commands implemented in the Proxmark, please use **hf mf** command to list all subcommands for MIFARE Classic.

The command **hf mf rdbl** can be used to read one block from the MIFARE Classic Card. Please make use of the help menu to understand how the Proxmark3 requires you to specify any arguments to the command, such as which block and keys.

Important to note that normally you need specify either Key A or Key B to read data from to a MIFARE Classic card. But for this challenge, you can omit the keys from your command and let Proxmark3 use the default FF FF FF FF FF FF key.

Flipper Zero

Flipper Zero has an option within the NFC module to read a card, this command can take some time since it's also trying to brute force and find any other keys on the card. (Explained later in the training). You could

cancel the read action since the data for this Sector will be retrieved at the start since its using the default key and its mostly spending time on the other sectors where no default keys are configured.

ACR122U

Use the --read_sector or --read_block command within the CRID tool to initiate a read on the MIFARE Classic Card. The help menu can show how to provide the options for the read operation, take note of the keys and data format flags.

References

Proxmark3	Use the Proxmark3 CLI to find which hf mf command you need to use.
ACR122U (CRID)	use the crid tool with read_sector
Data Conversion	Use CyberChef (https://cyberchef.org/) to convert from Hex to ASCII.

Walkthrough: PIN Code

Proxmark3

The help menu of the mifare commands provided you with two useful commands to read some specific data from a MIFARE Classic cards: The hf mf rdsc and hf mf rdbl which both stand for read sector and read block.

```
usage:
    hf mf rdsc [-habv] [-c <dec>] [-k <hex>] -s <dec>

options:
    -h, --help                  This help
    -a                          input key specified is A key (def)
    -b                          input key specified is B key
    -c <dec>                    input key type is key A + offset
    -k, --key <hex>             key specified as 6 hex bytes
    -s, --sec <dec>             sector number
    -v, --verbose               verbose output
```

Read the first sector using the command:

```
[usb] pm3 --> hf mf rdsc -s 1

[=]   # | sector 01 / 0x01                                | ascii
[=] ----+-------------------------------------------------+----------------
[=]   4 | 57 65 6C 63 6F 6D 65 20 74 6F 20 52 46 49 44 21 | Welcome to RFID
[=]   5 | 54 68 65 20 50 49 4E 20 43 6F 64 65 00 00 00 00 | The PIN Code...
[=]   6 | 31 33 33 37 00 00 00 00 00 00 00 00 00 00 00 00 | 1337............
[=]   7 | 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF | .........i.....
```

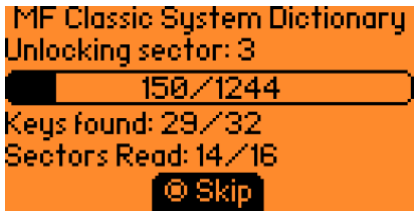
As you can see the PIN Code is listed there and is **1337**. Enter the PIN Code in the OctoBox to obtain the points.

Flipper Zero

Under the NFC module of the Flipper Zero you can find the Read function. That will essentially brute force most common keys and then output the data it identified, however it rather difficult to read the data with the Flipper, so I recommend using the ACR122U. In any, case here is how you can get the flag with the Flipper Zero:

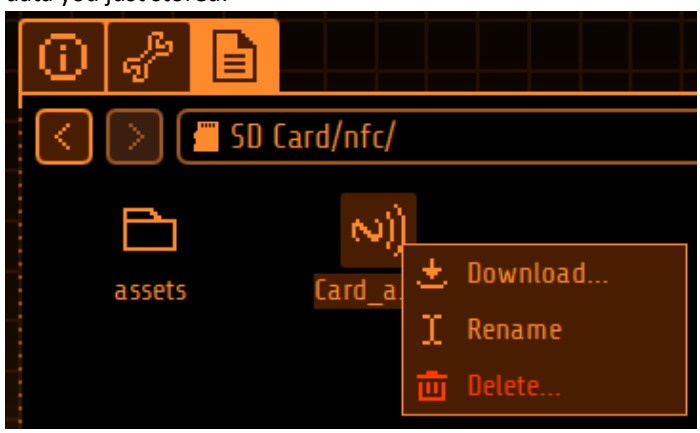
Under **NFC Screen**, select the **Read** function.

It will perform a brute force attack of a dictionary of keys for MIFARE Classic, if it found the first sectors feel free to skip the rest.



Then go to More and save the card to a file.

On the qFlipper application, open the file explorer and follow the path **SD Card/nfc/** and download the card data you just stored:



Once downloaded open the file in a text editor and observe sector 1:

```
Block 4: 57 65 6C 63 6F 6D 65 20 74 6F 20 52 46 49 44 21
Block 5: 54 68 65 20 50 49 4E 20 43 6F 64 65 00 00 00 00
Block 6: 31 33 33 37 00 00 00 00 00 00 00 00 00 00 00 00
```

This will translate to:
Block 4: Welcome to RFID!
Block 5: The PIN Code
Block 6: 1337

As you can see the PIN Code is listed there and is **1337**. Enter the PIN Code in the OctoBox to obtain the points.

ACR122U

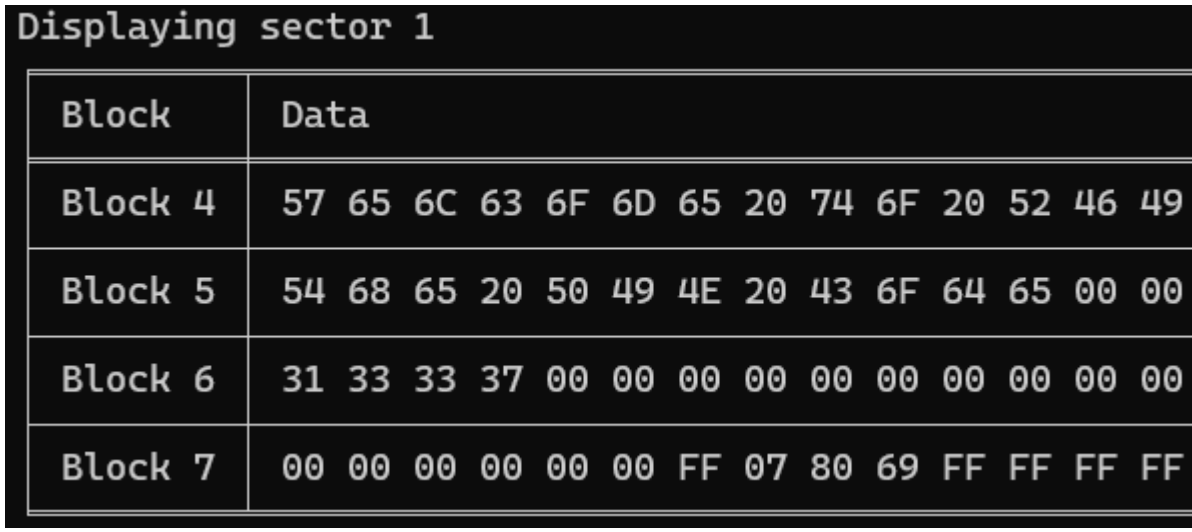
The tool CRID can be used during the exercise to work with the ACR122U device:

https://github.com/VinnieV/crid

Execute the following command to retrieve the sector data, by default CRID will try the default key "FFFFFFFFFFFF" to authenticate to the block.

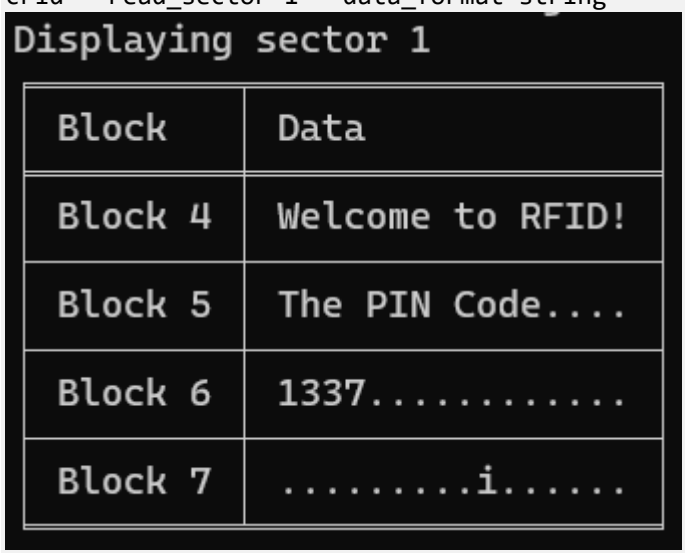
```
crid --read_sector 1
```

Which should output the following information:



You can also add data_format string to the command to get the ASCII representation immediately:

```
crid --read_sector 1 --data_format string
```



As you can see the PIN Code is listed there and is **1337**. Enter the PIN Code in the OctoBox to obtain the points.

Easy	Exercise #2: Username	75 POINTS
Devices		
Training Cards	**Card A or B** **Sector 2, Block 8**	

Description

This exercise will introduce you to writing data to a MIFARE Classic Card. The goal is to write your username in the first block of sector 2 (Block 8) on one of the MIFARE Classic Cards. This means that the username can only be 16 bytes (or 16 characters) long, if your username is shorter than 16 characters, please add null bytes (Hex: 00) after your username until you have 16 characters. An example for the username Spencer (Please use a different name):

Convert the username from text to hexadecimal using for example CyberChef or any other tool of your liking:

https://cyberchef.org/#recipe=To_Hex('None',0)&input=U3BlbmNlcg

This would give you the following result in hexadecimal. if not familiar with hexadecimal, each character is converted here to two characters (0-9,a,b,c,d,e,f):

```
5370656e636572
```

Since you would need to specify the full block during the commands for writing you would need to append leading zeros until the full data contains 32 characters in hexadecimal. The hexadecimal representation of the username Spencer is 14 characters long, thus we would need to append 18 leading zeros (32 minus 14 equals 18) resulting in the following data for that block:

```
5370656e6365720000000000000000000
```

For this exercise the access keys will be provided:

Access Key A: FFFFFFFFFFFF
Access Key B: FFFFFFFFFFFF

Use your RFID device to write a username to one of the cards and scan it on the OctoBox to validate.

Proxmark

Similar on how the read commands worked during the previous exercise, you might have noticed that the hf mf command also contains the **hf mf wrbl** command, which will be the command you can use here. You would need to specify the correct block number and data and provide a valid key and keytype with the command.

```
usage:
    hf mf wrbl [-hab] --blk <dec> [--force] [-k <hex>] [-d <hex>]

options:
    -h, --help                  This help
    --blk <dec>                 block number
    -a                          input key type is key A (def)
    -b                          input key type is key B
    --force                     override warnings
    -k, --key <hex>             key, 6 hex bytes
    -d, --data <hex>            bytes to write, 16 hex bytes

examples/notes:
    hf mf wrbl --blk 1 -d 000102030405060708090a0b0c0d0e0f
    hf mf wrbl --blk 1 -k A0A1A2A3A4A5 -d 000102030405060708090a0b0c0d0e0f
```

Flipper Zero

Small disclaimer, the blocks for the Flipper Zero start counting from 0 instead of 1 so for all exercises, please lower the block number within the exercise and challenge descriptions for the Flipper Zero by one.

Flipper Zero can read and modify card data in a few different ways, you can install the MIFARE Classic Editor to change data, or create a dump of the MIFARE Classic file when using the read function in NFC module, store the export and then edit the data within the mobile application of the Flipper or either by downloading the dump of the Card through the qFlipper application and modifying the data in a text editor. The solutions will mainly use the latter one, but the concepts remain the same across all methods.

If you notice missing data within your card dumps in the later exercises and challenges, thus blocks of data that are marked as "??" you will need to run the read operation again and make sure it finishes completely which can take some time.

ACR122U

The crid tool has the --write_block function available to write data to MIFARE Classic Card. Use the help menu to identify the required options and flags to make the write function succeed. Make sure the data you are providing is in total 32 characters long in hexadecimal.

```
# Write "Hello, World!" to block 3 using key type B and key value a1b2c3d4e5f6
crid --write_block 3 --data "48656C6C6F2C20576F726C6421" --key_type B --key_value a1b2c3d4e5f6
```

References

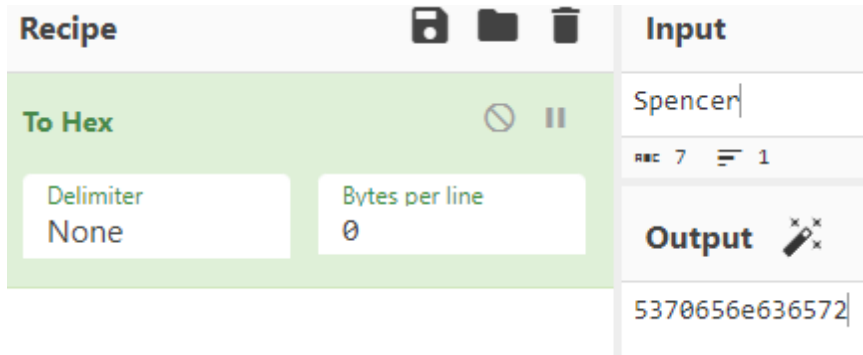
Proxmark3	Use the Proxmark3 cli to find which hf mf command you need to use.
ACR122U (CRID)	use the crid tool with write_block and read_block features
Convert Data	https://cyberchef.org/#recipe=To_Hex('None',0)

Walkthrough: Username

Proxmark3

Since the Proxmark3 command uses hexstrings for data, we need to convert our username to a hexstring, which you can do in various ways but also easily with CyberChef:

https://cyberchef.org/#recipe=To_Hex('None',0)&input=U3BlbmNlcg



Make sure the leading zeros are fine a since the key is given and the access bits permissions are correct, we can invoke the following command on the Proxmark3 CLI to write your username:

```
hf mf wrbl --blk 8 -k FFFFFFFFFFFF -d 5370656e6365720000000000000000
```

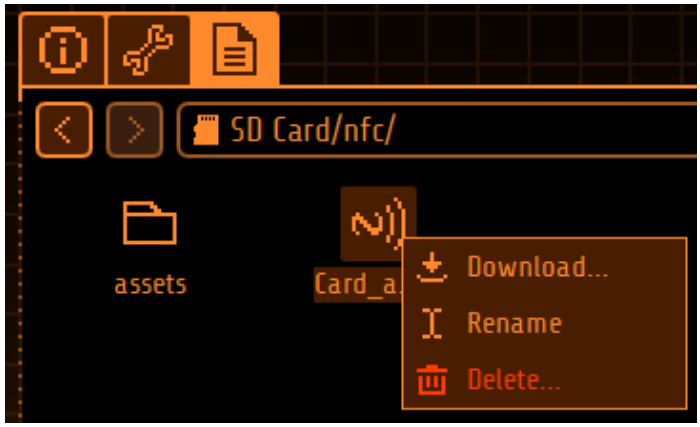
```
[usb] pm3 --> hf mf wrbl --blk 8 -k FFFFFFFFFFFF -d 5370656e6365720000000000000000
[=] Writing block no 8, key A - FFFFFFFFFFFF
[=] data: 53 70 65 6E 63 65 72 00 00 00 00 00 00 00 00 00
[+] Write ( ok )
```

Flipper Zero

Until this date, I did not find an easy function to write specific values to specific blocks on a MIFARE Cards, like other devices can. However, when reading a MIFARE card there is an option to save the card to a file with the Flipper Zero and using the qFlipper application you can download the file on the computer. The file has the extension .nfc file and has an easy structure to understand when opening in a text editor:

```
Mifare Classic type: 1K
Data format version: 2
# Mifare Classic blocks, '??' means unknown data
Block 0: 1C 79 28 6F 22 08 04 00 62 63 64 65 66 67 68 69
Block 1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 3: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
Block 4: 57 65 6C 63 6F 6D 65 20 74 6F 20 52 46 49 44 21
Block 5: 54 68 65 20 66 69 72 73 74 20 66 6C 61 67 3A 00
Block 6: 66 6C 61 67 7B 57 33 6C 63 30 6D 65 21 7D 00 00
Block 7: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
Block 8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 9: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 11: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
```

1) On the qFlipper application, open the file explorer and follow the path **SD Card/nfc/** and download the card data you just stored:



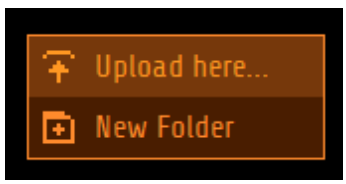
2) Once downloaded open the file in a text editor. But before we can modify we first need to convert the username we want store in the correct format. You can use the following Cyberchef link throughout the course to convert your strings to hexadecimal:

https://cyberchef.org/#recipe=To_Hex('Space',0)

3) Once you have the correct format replace your hexadecimal data on block 8

```
Block 7: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
Block 8: 53 70 65 6e 63 65 72 00 00 00 00 00 00 00 00 00
Block 9: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

4) Then save that file and upload it back to the flipper:



5) Once its uploaded you can use the NFC module and open the saved menu function to list all cards saved in memory. You select your card and either emulate or write the info to another training card. Writing might give you some error stating that not all blocks were written, but normally it should be fine for the username block.

ACR122U

The tool CRID can also be used to upload data to a MIFARE classic card as well through the write block feature:

```
crid --write_block 8 --data 5370656e6365720000000000000000
```

Which should output the following information

```
DEBUG: Available readers: ['ACS ACR122 0']
INFO: Using reader ACS ACR122 0
INFO: Connected to reader..
DEBUG: Reader connected
INFO: Connected to card..


                _____ _____ _____ _____
               / ____|  __ \_   _|  __ \
              | |    | |__) || | | |  | |
              | |    |  _  / | | | |  | |
              | |____| | \ \_| |_| |__| |
               \_____|_|  \_\_____|_____/
                                      v1

INFO: Authenticated using the key on block 8..
INFO: Authenticated using the key on block 8..
INFO: Write successful for block 8.
```

And when viewing the block information:

```
crid --read_block 8 --data_format string
```

```
PS C:\Users\kernelpanic> crid --read_block 8 --data_format string
DEBUG: Available readers: ['ACS ACR122 0']
INFO: Using reader ACS ACR122 0
INFO: Connected to reader..
DEBUG: Reader connected
INFO: Connected to card..


                _____ _____ _____ _____
               / ____|  __ \_   _|  __ \
              | |    | |__) || | | |  | |
              | |    |  _  / | | | |  | |
              | |____| | \ \_| |_| |__| |
               \_____|_|  \_\_____|_____/
                                   v0.2.0

INFO: Loaded the key [255, 255, 255, 255, 255, 255] as location 0..
INFO: Authenticated using the key [255, 255, 255, 255, 255, 255] as 96 for block 8.
Block 8: Spencer.........
INFO: Closing reader..
```

Validate your card on the OctoBox to obtain some points.

Easy	Exercise #3: Access Bits	75 POINTS
Devices		
Training Cards	**Card A or B** **Sector 3, Block 12**	

Description

During this exercise we will investigate the access bits of MIFARE Classic card. Access bits can be used to specify what permissions key A or B has. The different permissions are:

- read
- write
- increment
- decrement/transfer/restore

Each sector has its own access bits and are always located at **7-10th bytes in the last block of the sector**.

The goal of this exercise is to write some data in the first block of sector 3 (Block 12) on one of the MIFARE Classic Cards and test it against the OctoBox. You will notice that the commands will need to change slightly since the access bits will be configured in some specific way. To understand how the access bits are configured for Sector 3. Take the 7-10th bytes from the last block within the sector, block 15:

Block 15: FF FF FF FF FF FF 78 77 88 69 A1 A2 A3 A4 A5 A6

787788

Use the following tool https://slebe.dev/mifarecalc/ to understand what these bytes mean and what needs to change to make the write command work.

Proxmark3 also has the command hf mf acl -d XXXXXX that can help you figure out the access bits.

In order to obtain points on the OctoBox for this challenge, prove that you write by writing the following flag to block 12:

```
String: flag{icanwrite!}
Hexstring: 666C61677B6963616E7772697465217D
```

The access keys are provided again for this exercise:

Access Key A: FFFFFFFFFFFF

Access Key B: A1A2A3A4A5A6

References

MIFARE Access Bit Calculator	https://slebe.dev/mifarecalc/

Proxmark3 Access Bit Command	hf mf acl -d XXXXXX

Walkthrough: Access Bits

Proxmark3

Just as the username exercise if we can try to write the flag on block 12 authenticating with key A using the following command:

```
hf mf wrbl --blk 12 -d 666C61677B6963616E7772697465217D
```

Then you would receive the following message:

```
[=] Writing block no 12, key A - FFFFFFFFFFFF
[=] data: 66 6C 61 67 7B 69 63 61 6E 77 72 69 74 65 21 7D
[-] Write ( fail )
[?] Maybe access rights? Try specify keytype `hf mf wrbl -b ...` instead
```

When reviewing the whole sector with hf mf rdsc -s 3 take a closer look at the access bit.

```
[=]   # | sector 03 / 0x03                                  | ascii
[=] ----+-------------------------------------------------+-----------------
[=]  12 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]  13 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]  14 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]  15 | 00 00 00 00 00 00 78 77 88 69 00 00 00 00 00 00 | ......xw.i......
```

It can be seen that the access bits for that specific block can are configured as follows:

787788

Which means the following when converting to more human readable format using the following Proxmark3 Command:

```
hf mf acl -d 787788
```

```
[=]   # | Access rights
[=] ----+-------------------------------------------------------------------
[=]   0 | read AB; write B
[=]   1 | read AB; write B
[=]   2 | read AB; write B
[=]   3 | write A by B; read ACCESS by AB; write ACCESS by B; write B by B
```

The correct file can then be written to block 12 on the card using key b:

```
hf mf wrbl --blk 12 -b -k A1A2A3A4A5A6 -d 666c61677b6963616e7772697465217d
```

```
[=] Writing block no 12, key B - A1A2A3A4A5A6
[=] data: 66 6C 61 67 7B 69 63 61 6E 77 72 69 74 65 21 7D
[+] Write ( ok )
```

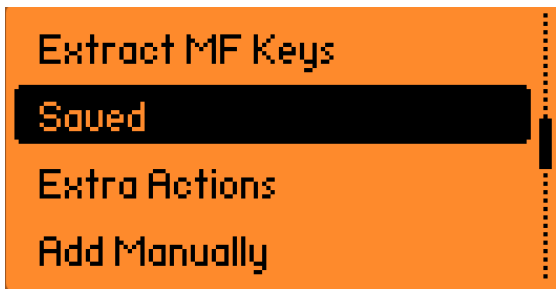
Flipper Zero

For the flipper it would be a little bit different, as you would be modifying the text file directly again and its not really enforcing those access bits. Although it will be enforced when you would try to write to a card that has that sector configuration. If that fails you can emulate the card as well. For this exercise modifying the file and writing to a card is what will work.

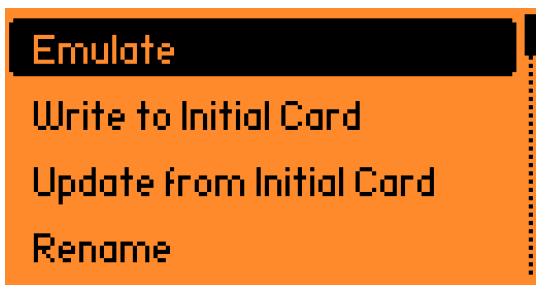
First get your latest card file save that includes the username as well, then modify the block 12 to include the flag:

```
Block 11: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
Block 12: 66 6C 61 67 7B 69 63 61 6E 77 72 69 74 65 21 7D
Block 13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

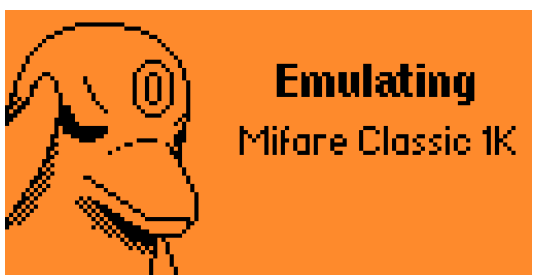
Then upload the file back through the qFlipper and within the NFC module go to the menu “saved”:



Select your file name and either **Emulate** or **Write to the Initial card**:



Keep in mind that writing errors might just indicate that other sectors were not successful.



ACR122U

Execute the following command to retrieve the sector 1:

```
crid --read_sector 3
```

Which should output the following information:

Displaying sector 3

Block	Data
Block 12	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 13	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 14	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 15	00 00 00 00 00 00 78 77 88 69 00 00 00 00 00 00

As the access bits for that specific block are configured as follows

78 77 88

Which means the following when converting to more human readable format:

	Read	Write
Block 12	Key A | Key B	Key B
Block 13	Key A | Key B	Key B
Block 14	Key A | Key B	Key B
Block 15	Key A | Key B	Key B

Write the flag using key B setting and the correct key and it should work:

```
crid --write_block 12 --data 666C61677B6963616E7772697465217D --key_type B --
key_value A1A2A3A4A5A6
```

```
INFO: Authenticated using the key on block 12..
INFO: Authenticated using the key on block 12..
INFO: Write successful for block 12.
```

Once you scanned this card on the OctoBox and your username is still present on the Card it will automatically award you the points.

Medium	Exercise #4: Brute Force	100 POINTS
Devices		
Training Cards	**Card A or B** **Sector 4, Block 18**	

Description

Within this exercise you will need to obtain full access to sector 4 by brute forcing the keys for sector 4 and write a flag to block 18. There is no need for performing any of the MIFARE attacks for this one but just an old trick of the book, brute forcing the key is the goals here. Specifically, an “online” brute force attack.

Access Key A: FFFFFFFFFFFF
Access Key B: ????????????

The flag can be written on the third block of sector 4 (Block 18) on one of the MIFARE Classic Cards:

```
String: flag{Brut333333}
Hexstring: 666C61677B4272757433333333333337D
```

Once you have done the write successfully, test it against the OctoBox.

Proxmark

Take a look at the **hf mf chk** command implemented in the Proxmark, within the Proxmark3 repo you can find wordlists for MIFARE in the folder **client/dictionaries** , mfc_keys_bmp_sorted.dic for example.

Flipper Zero

Flipper Zero does the brute force when initiating the read command, but the default list will not contain the correct key. In order to add new wordlists, copy the mf_classic_dict_user.nfc file onto the SD Card -> nfc -> assets folder. You can find the wordlist on the Flipper Zero GitHub repository:
https://github.com/UberGuidoZ/Flipper/blob/main/NFC/mf_classic_dict/mf_classic_dict_user.nfc

ACR122U

The crid tool has the ---brute_force_keys function available where you can specify a list of keys it will try. The list you could use is also available in the Crid repository:

https://github.com/VinnieV/crid/blob/main/mifare_access_keys_top100.dic

References

MIFARE Keys	https://github.com/VinnieV/crid/blob/main/mifare_access_keys_top100.dic
Flipper Zero Adding MIFARE Keys Documentation	https://github.com/UberGuidoZ/Flipper/blob/main/NFC/mf_classic_dict/ReadMe.md

Walkthrough: Brute Force

Proxmark3

The Proxmark3 support a brute force mode for MIFARE keys using the hf mf chk command:

```
usage:
    hf mf chk [-hab*] [-k <hex>]... [--tblk <dec>] [--mini] [--1k] [--2

options:
    -h, --help                      This help
    -k, --key <hex>                 Key specified as 12 hex symbols
    --tblk <dec>                    Target block number
    -a                              Target Key A
    -b                              Target Key B
    -*, --all                       Target both key A & B (default)
    --mini                          MIFARE Classic Mini / S20
    --1k                            MIFARE Classic 1k / S50 (default)
    --2k                            MIFARE Classic/Plus 2k
    --4k                            MIFARE Classic 4k / S70
    --emu                           Fill simulator keys from found keys
    --dump                          Dump found keys to binary file
    -f, --file <fn>                 Filename of dictionary
```

Take a look in the clients/dictionaries folder and try out a few for MIFARE classic lists and hopefully you get lucky!

```
pm3 ~/proxmark3$ ls client/dictionaries/
extras/          iclass_default_keys.dic  mfc_keys_bmp_sorted.dic    mfc_keys_mrzd_sorted.dic
ht2_default.dic  mfc_default_keys.dic     mfc_keys_icbmp_sorted.dic  mfdes_default_keys.dic
```

The command that should work is the following:

```
hf mf chk -b --tblk 18 -f client/dictionaries/mfc_keys_bmp_sorted.dic
```

```
[usb] pm3 --> hf mf chk -b --tblk 18 -f mfc_keys_bmp_sc
[+] loaded 58 keys from hardcoded default array
[+] loaded 1000 keys from dictionary file C:\Users\kerr
[+] loaded 1000 keys from dictionary
[=] Start check for keys...
[=] .......
[=] time in checkkeys 3 seconds


[+] found keys:

[+] -----+-----+--------------+---+--------------+----
[+]  Sec | Blk | key A        |res| key B        |res
[+] -----+-----+--------------+---+--------------+----
[+]  000 | 003 | ------------ | 0 | ------------ | 0
[+]  001 | 007 | ------------ | 0 | ------------ | 0
[+]  002 | 011 | ------------ | 0 | ------------ | 0
[+]  003 | 015 | ------------ | 0 | ------------ | 0
[+]  004 | 019 | ------------ | 0 | 5C43A75C65A0 | 1
[+] -----+-----+--------------+---+--------------+----
[+] ( 0:Failed / 1:Success )
```

Now we know key B we can write the flag to block 18 and scan it with the OctoBox to get some points!

hf mf wrbl --blk 18 -b -k 5C43A75C65A0 -d 666C61677B4272757433333333333337D

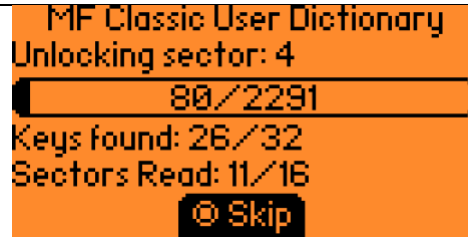
Flipper Zero

The Flipper Zero can brute force keys and it will go through many keys by default, but for this exercise the key is not included in the default list, and you will need to add your own user lists to the Flipper Zero as well:

https://github.com/UberGuidoZ/Flipper/blob/main/NFC/mf_classic_dict/mf_classic_dict_user.nfc

Download the file and either use the SDCard or the qFlipper application to upload the file to the device. When viewing the MIFARE Classic Keys configuration screen within the extra actions menu of the NFC module you would need to see that keys are added to User dictionary.

Next up is to perform the read command within the NFC module to brute force with the new keys. You can stop the reading and brute forcing as soon as its passed Sector 4, which can take some time:



Your brute force was not successful if the following line is still present in your save file:

```
Block 19: FF FF FF FF FF FF 3F 03 CC 69 ?? ?? ?? ?? ?? ??
```

If the brute force succeeded you will have the following data in Block 19 in the save file:

```
Block 19: FF FF FF FF FF FF 3F 03 CC 69 5C 43 A7 5C 65 A0
```

Once the save file has the correct key B on block 19 then you can go ahead and modify the file with writing the flag on Block 18:

```
Block 18: 66 6C 61 67 7B 42 72 75 74 33 33 33 33 33 33 7D
Block 19: FF FF FF FF FF FF 3F 03 CC 69 5C 43 A7 5C 65 A0
```

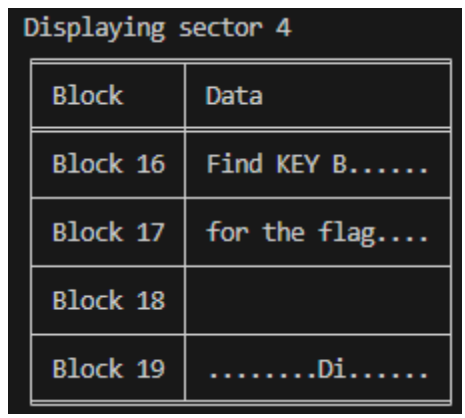
As usual upload the file again through qFlipper and write to a card or emulate and you will be awarded some points when checking the flag on the OctoBox.

ACR122U

Execute the following command to retrieve the sector 4:

```
crid --read_sector 4 --data_format string
```

Which should output the following information:



So we need to find KEY B which is indicated by the hint within the sector and the access bit. For this challenge we begin with a simple brute force attack and to save time, we will work with a predefined

wordlist that you can find within the fileserver or within the GitHub Repository of crid. Then execute the following command to initiate the brute force command

```
crid --brute_force_keys 18 --key_list mifare_access_keys_top100.dic --key_type B
```

```
Total keys: 100
100% |########################################################################|
INFO: Valid key found (Type B): 5C43A75C65A0 for block 18.
```

Now the key can be used to write the flag to block 18:

```
crid --write_block 18 --key_type B --key_value 5C43A75C65A0 --data
666C61677B4272757433333333333337D
```

```
INFO: Loaded the key [92, 67, 167, 92, 101, 160] as location 1..
INFO: Authenticated using the key [92, 67, 167, 92, 101, 160] as 97 for block 18.
INFO: Loaded the key [92, 67, 167, 92, 101, 160] as location 1..
INFO: Authenticated using the key [92, 67, 167, 92, 101, 160] as 97 for block 18.
INFO: Write successful for block 18.
```

We can validate that the write was successful by reading back block 18:

```
crid --read_block 18 --key_type B --key_value 5C43A75C65A0 --data_format string
```

```
INFO: Loaded the key [92, 67, 167, 92, 101, 160] as location 1..
INFO: Authenticated using the key [92, 67, 167, 92, 101, 160] as 97 for block 18.
Block 18: flag{Brut333333}
```


Medium	**Exercise #5: Key Recovery**	125 POINTS
Devices		
Training Cards	**Card A or B** **Sector 5, Block 22**	

Description

Within this exercise you will need to obtain full access to sector 5 and write a flag to block 22. Instead of brute forcing the keys we will use the vulnerabilities within the MIFARE Classic chips to obtain all access keys from the card.

The keys for this sector are as follows, with key B unknown:

Access Key A: FFFFFFFFFFFF
Access Key B: ???????????

The flag can be written on the third block of sector 5 (Block 22) on Card A or B:

```
String: flag{H5ck3r}
Hexstring: 666c61677b4835636b33727d00000000
```

Once you have done the writing successfully, test it against the OctoBox.

Proxmark

The Proxmark3 implements the different attacks highlighted within the Theory, such as:

- The darkside attack
  hf mf darkside
- The nested attack
  hf mf nested
- The staticnested attack
  hf mf staticnested
- The hardnested attack
  hf mf hardnested

However there is an easier way by leveraging a builtin script within the Proxmark3 that enumerates all these and automatically attempt to exploit these. This can be done with **hf mf autopwn**.

Flipper Zero

Flipper Zero does has an application you can install called Mfkey32. It will be available under NFC within the apps modules once you installed the application. Running the tool might be confusing, but first you will need to save a dump of the current card into a file using the reader command in NFC module, like we have done in the previous exercises.

Once you have the card saved, you will need to open the saved card and use the Detect reader command. This will allow you to tap the reader a couple of times to capture nonces which are random cryptographic tokens generated during authentication, once enough nonces are collected the Mfkey32 can perform cryptographic operations and brute forcing to obtain the keys.

ACR122U On Linux you can use the tool miLazyCracker, which is wrapper around all

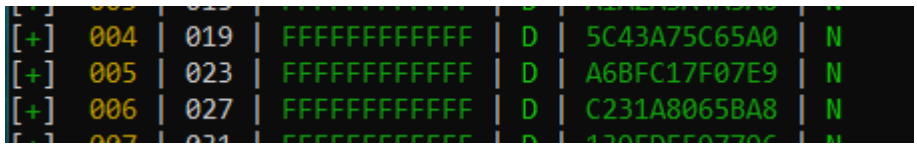
References

MIFARE Classic Offline Cracker	https://github.com/nfc-tools/mfoc
LibNFC	http://www.libnfc.org/api/
Flipper Zero MFKey32 Documentation	https://docs.flipper.net/nfc/mfkey32
miLazyCracker	https://github.com/nfc-tools/miLazyCracker

Walkthrough: Key Recovery

Proxmark3

The hf mf autopwn command will enumerate all possible vulnerabilities for the MIFARE Classic card and attempt to recover all keys and data from the card.When executing this you will be able to see that it found key B for sector 5:



Writing the flag on block 22 is now easy:

```
hf mf wrbl --blk 22 -b -k A6BFC17F07E9 -d 666c61677b4835636b33727d00000000
```

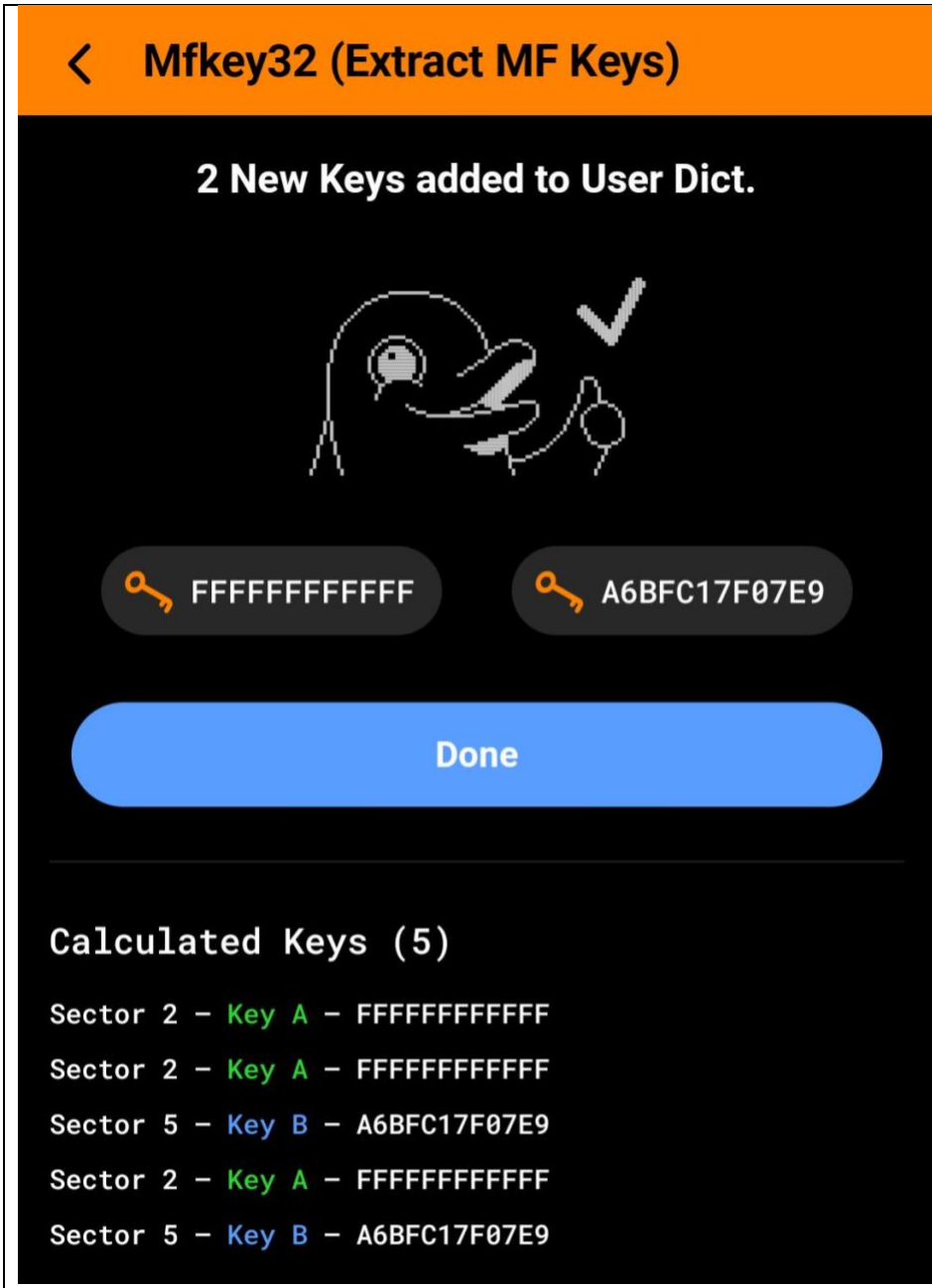
Scan the card on the OctoBox to get some points!

Flipper Zero

First make a save file of the RFID card.

Now within the NFC module, use the save menu to open the previously stored capture of your card and choose the Extract MF Keys. With this screen loaded, you can go to the OctoBox open the correct challenge and tap the reader a couple of times until 10 nonces are captured:

Now that the nonces are stored click next a couple of times and open the Mfkey32 application from the Apps within your Flipper Zero Mobile Application. It can take some time to calculate the keys, but once successful the following show be shown:



Now make sure to configure and use the correct keys for Block 22 and 23:

Block 22: 66 6c 61 67 7b 48 35 63 6b 33 72 7d 00 00 00 00
Block 23: FF FF FF FF FF FF 3F 03 CC 69 A6 BF C1 7F 07 E9

As usual upload the file again through qFlipper and write to a card or emulate and you will be awarded some points when checking the flag on the OctoBox.

ACR122U

If you are running Linux you will be able to execute the miLazyCracker script and miLazyCrackerFreshInstall.sh script from: https://github.com/nfc-tools/miLazyCracker

Execute the miLazyCracker script when you card is present on the reader:
sudo ./miLazyCracker.sh

```
Sector 00 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 01 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 02 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 03 - Found   Key A: ffffffffffff Unknown Key B
Sector 04 - Found   Key A: ffffffffffff Unknown Key B
Sector 05 - Found   Key A: ffffffffffff Unknown Key B
Sector 06 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 07 - Unknown Key A               Unknown Key B
Sector 08 - Found   Key A: ffffffffffff Unknown Key B
Sector 09 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 10 - Found   Key A: ffffffffffff Found   Key B: af720e83d0f1
Sector 11 - Unknown Key A               Unknown Key B
Sector 12 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 13 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 14 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff
Sector 15 - Found   Key A: ffffffffffff Found   Key B: ffffffffffff


Using sector 00 as an exploit sector
Sector: 7, type A, probe 0, distance 64 .....
  Found Key: A [b2784d1832f8]
  Data read with Key A revealed Key B: [67a89b08c934] - checking Auth: OK
Sector: 11, type A, probe 0, distance 64 .....
  Found Key: A [02872fb92433]
  Data read with Key A revealed Key B: [000000000000] - checking Auth: Failed!
Sector: 3, type B, probe 0, distance 64 .....
  Found Key: B [a1a2a3a4a5a6]
Sector: 4, type B, probe 0, distance 64 .....
  Found Key: B [5c43a75c65a0]
Sector: 5, type B, probe 0, distance 64 .....
  Found Key: B [a6bfc17f07e9]
Sector: 8, type B, probe 0, distance 64 .....
  Found Key: B [be13377331eb]
Sector: 11, type B, probe 0, distance 64 .....
  Found Key: B [14318d91bfe5]
```

It now found key for sector 5, which is A6BFC17F07E9, use the key to write the correct data to block 22.

```
crid --write_block 22 --key_type B --key_value A6BFC17F07E9 --data
666c61677b4835636b33727d00000000
```

```
INFO: Loaded the key [166, 191, 193, 127, 7, 233] as location 1..
INFO: Authenticated using the key [166, 191, 193, 127, 7, 233] as 97 for block 22.
INFO: Loaded the key [166, 191, 193, 127, 7, 233] as location 1..
INFO: Authenticated using the key [166, 191, 193, 127, 7, 233] as 97 for block 22.
INFO: Write successful for block 22.
```

If you are using macOS you will need to perform the MIFARE attacks without miLazyCracker and run the tools that miLazyCracker would run, separately:

```
mfoc -O Exercise5
```

```
Sector: 5, type B, probe 0, distance 64 .....
Found Key: B [a6bfc17f07e9]
```

This should also provide you a full dump of the card.

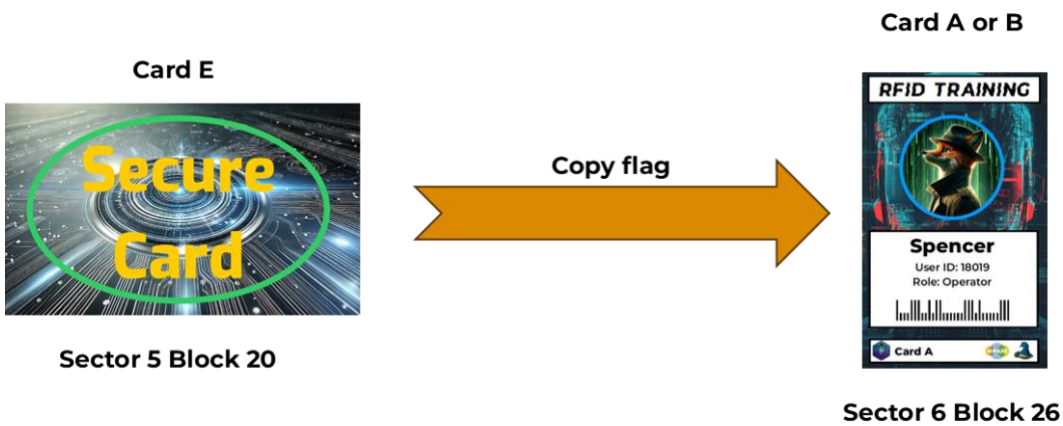
Medium	Exercise #6: Key Recovery 2	150 POINTS
Devices		
Training Cards	**Card E** **Sector 6 Block 26**	

Description

As you might have noticed its not possible to perform the previous attacks on Card D. This is because all keys are random on the card which makes sure the exploits don't have a default sector to use for the exploitation.

It would require us to brute force or intercept the keys transmitted by the readers to obtain the keys now. However, Philippe Teuwen found a backdoor on some unofficial common MIFARE Classic chip that allows us to bypass all access controls. This research has only been implemented in the Proxmark3 currently.

The goal is to obtain the flag from sector 5 on Card D and place it on block 26 in Card A or B. Instead of brute forcing the keys we will use the vulnerabilities within the MIFARE Classic chips to obtain all access keys from the card.



Once you have done the writing successfully, test it against the OctoBox.

Proxmark

The Proxmark3 is capable of doing the backdoor commands with:
hf mf ecfill -c 4 --key A396EFA4E24F
hf mf eview

Flipper Zero

Not possible with the Flipper Zero.

ACR122U

No possible with ACR122U.

References

MIFARE Classic: exposing the static encrypted nonce variant Philippe Teuwen	https://eprint.iacr.org/2024/1275.pdf

Walkthrough: Key Recovery 2

Proxmark3

As you will notice hf mf autopwn will not work since it did not find any known key to use it for exploitation. However, the latest Proxmark3 version has the backdoor commands implemented for the chip that the card uses. The whole data layout can be dumped by executing the following command:

hf mf ecfill -c 4 --key A396EFA4E24F

```
[usb] pm3 --> hf mf ecfill -c 4 --key A396EFA4E24F
[+] Fill ( ok ) in 313 ms
```

To then view the data:

hf mf eview

```
 5 |   20 | 66 6C 61 67 7B 62 34 63 6B 44 30 30 33 7D 00 00 | flag{b4ckD003}
   |   21 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   |   22 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
```

Writing the flag on block 26 is now easy:

```
hf mf wrbl --blk 26 -b -k 112233445566 -d 666C61677B6234636B443030337D0000
```

```
 6 |   24 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   |   25 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
   |   26 | 66 6C 61 67 7B 62 34 63 6B 44 30 30 33 7D 00 00 | flag{b4ckD003}
   |   27 | 11 22 33 44 55 66 FF 07 80 69 11 22 33 44 55 66 | ."3DUf...i."3DUf
```

Scan the card on the OctoBox to get some points!

Medium	Exercise #7: Magic MIFARE	150 POINTS
Devices		
Training Cards	**Card A or B** **UID**	

Description

The UID is the first 4 or 7 bytes within the first block of the card. This block is supposed to be read-only and a way to ensure authenticity. However, you can buy special MIFARE cards called Magic MIFARE Cards that have a modified chipset that allow modifying the UID. There are different versions of these, indicated with Gen1,2,… . It usually works by sending a custom APDU command to enable writing the UID, most of the times this will be implemented in an easy command within your tool.

The goal of this challenge is modifying the UID of your card so it has the following UID:

UID: DEADBEEF

Once you have modified the UID of your card successfully, test it against the OctoBox.

Proxmark

The hf mf csetuid command in the Proxmark3 CLI can be used to change the UID of the card.

Flipper Zero

Currently did not find a way to change the UID of the card using the Flipper Zero, but emulation is possible. So within the NFC module you can manually add a card with 4 byte UID.

ACR122U

Currently this is not implemented within the crid tool but there is a tool within the lib-nfc suite that allow you to write 4-byte UID's to magic cards. The tool can be executed using the **nfc-mfsetuid** command.

References

MIFARE Classic Offline Cracker	https://github.com/nfc-tools/mfoc
LibNFC	http://www.libnfc.org/api/

Walkthrough: Magic MIFARE

Proxmark3

If you want to verify if the card is capable of the mifare magic backdoor commands for gen1 cards you can execute the following command:

```
hf mf cview
```

This should ouput the whole card without any problems. Next step would be to overwrite the UID of the card using the following command:

```
hf mf csetuid -u DEADBEEF
```

```
[usb] pm3 --> hf mf csetuid -u DEADBEEF
[+] old block 0... 1D540E6F2808040062636465666768 69
[+] new block 0... DEADBEEF2208040062636465666768 69
[+] Old UID... 1D 54 0E 6F
[+] New UID... DE AD BE EF  ( verified )
```

When reading the card with hf search it showed the UID modification was successful, and with this you can test it on the OctoBox.

```
[usb] pm3 --> hf search
[|] Searching for ISO14443-A tag...
[+]  UID: DE AD BE EF
```

Flipper Zero

There are two options to change the UID, either modify the NFC save file or emulate a new UID. Both methods are demonstrated here:

Take your latest NFC Save file and change the UID field to DE AD BE EF:

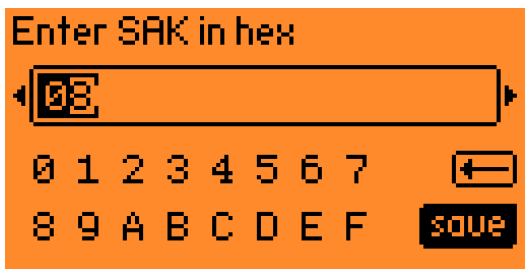
```
5 # UID is common for all form
6 UID: DE AD BE EF
7 # ISO14443-3A specific data
```

As usual upload the file again through qFlipper and write to a card or emulate and you will be awarded some points when checking the card on the OctoBox.

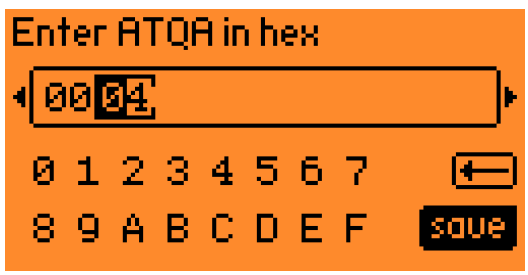
With the flipper you can also emulate the UID easily when using the Add Manually functionality within the NFC module:

Then select NFC-A 4 bytes UID

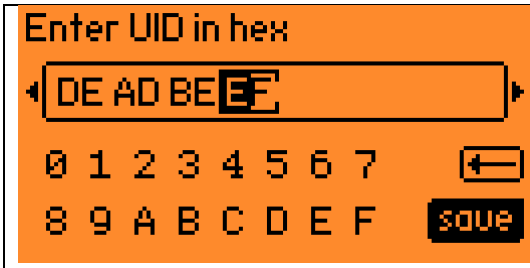
SAK: 08



ATQA: 00 04

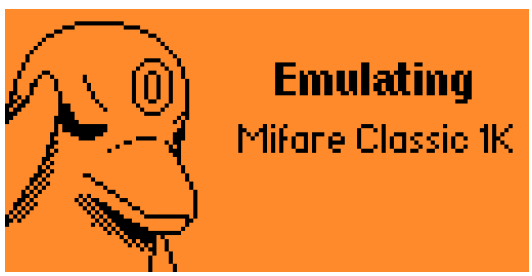


UID: DE AD BE EF



Then save the card as any filename you want and then open that card again in the next menu or under the Saved menu of the NFC module, and start the Emulate UID function:

This emulates the UID of the card, which is enough for this exercise but could differ from other systems where other data on MIFARE card is also verified, which is often the case for access cards.



ACR122U

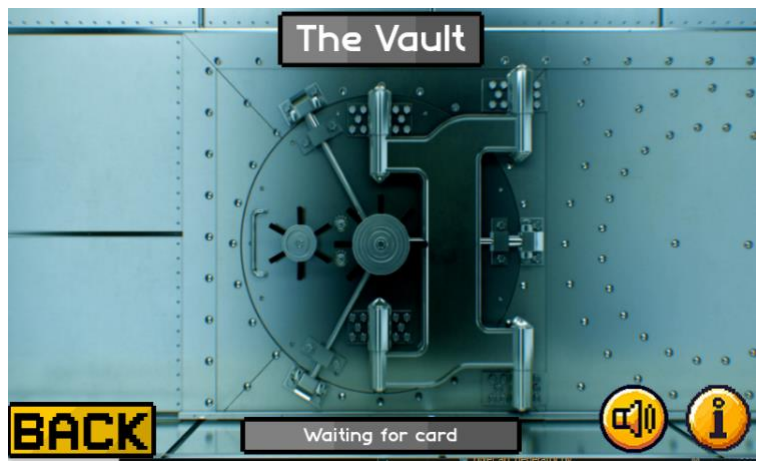
Execute the command **nfc-mfsetuid DEADBEEF** to write the UID of the card:

```
▶ nfc-mfsetuid DEADBEEF
NFC reader: ACS / ACR122U PICC Interface opened
Sent bits:     26 (7 bits)
Received bits: 04  00
Sent bits:     93  20
Received bits: 12  34  56  78  08
Sent bits:     93  70  12  34  56  78  08  3c  a2
Received bits: 08  b6  dd

Found tag with
 UID: 12345678
ATQA: 0004
 SAK: 08

Sent bits:     50  00  57  cd
Sent bits:     40 (7 bits)
Received bits: a (4 bits)
Sent bits:     43
Received bits: 0a
Card unlocked
Sent bits:     a0  00  5f  b1
Received bits: 0a
Sent bits:     de  ad  be  ef  22  08  04  00  46
Received bits: 0a
```


11.2. MIFARE Classic Challenges

Medium	Challenge #1: Vault	150 POINTS
Devices		
Training Cards	**Card A or B** **Sector 7**	

Description



During the first challenge for MIFARE Classic, you need to open a vault. The vault is protected, but maybe you can figure out how to reconfigure card A or B based on the backup data on Sector 7. Craft the correct card to open the vault.

Hints (base64)

Hint 1	U29tZSBkYXRhIG5lZWRzIHRvIGJlIGFwcGxpZWQgdG8gc2VjdG9yIDA=
Hint 2	VGhlIHRpbWVzdGFtcCBibG9jayAzMCBpcyBpbiBlcG9jaCBmb3JtYXQ=
Hint 3	VGhlIHRpbWVzdGFtcCBtaWdodCByZXF1aXJlIGFuIHVwZGF0ZQ==

Walkthrough: Vault

Proxmark3

When reviewing sector 7 data with the Proxmark3 you will obtain the following:
hf mf rdsc -s 7 -b -k 67A89B08C934

```
 7 |   28 | 56 61 75 6C 74 20 55 49 44 3A 00 00 BA DC 0F FE | Vault UI
   |   29 | 56 61 6C 69 64 20 75 6E 74 69 6C 3A 00 00 00 00 | Valid un
   |   30 | 67 54 B7 F7 00 00 00 00 00 00 00 00 00 00 00 00 | gT......
   |   31 | B2 78 4D 18 32 F8 FF 07 80 69 67 A8 9B 08 C9 34 | .xM.2...
```

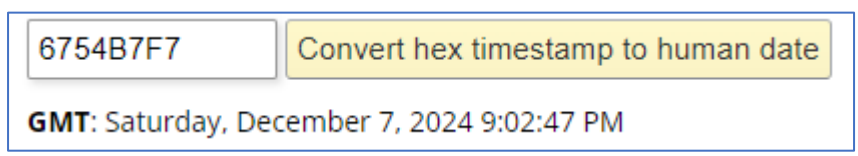
The vault ID is listed in the first block in hexadecimal, so make sure to change your card UID to BADC0FFE:

hf mf csetuid -u BADC0FFE

```
usb] pm3 --> hf mf csetuid -u BADC0FFE
+] old block 0... 7CCC0B3F84080400626364
+] new block 0... BADC0FFE97080400626364
+] Old UID... 7C CC 0B 3F
+] New UID... BA DC 0F FE  ( verified )
```

As you will notice this is not enough, but the rest of the data indicates there is some extra time validation going on. When decoding block 30 to a valid epoch time you will receive a valid recent date:

6754B7F7



We just need to update this timestamp to something greater than the current time, for example by increasing the number significantly:

```
hf mf wrbl --blk 30 -b -k 67A89B08C934 -d FF54B7F700000000000000000000000000
```

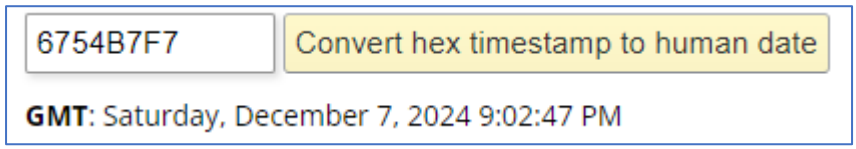
Flipper Zero

Make sure you have the keys for sector 7 and are able to read the full sector in your dumped NFC file:

Block 28: 56 61 75 6C 74 20 55 49 44 3A 00 00 BA DC 0F FE
Block 29: 56 61 6C 69 64 20 75 6E 74 69 6C 3A 00 00 00 00
Block 30: 67 54 B7 F7 00 00 00 00 00 00 00 00 00 00 00 00
Block 31: B2 78 4D 18 32 F8 FF 07 80 69 67 A8 9B 08 C9 34

Which translates to the following:

Block 28: Vault UID: BADC0FFE (hex)
Block 29: Valid Until:
Block 30 contains the hex value: **6754B7F7**, which can be translated to valid datetime stamp:



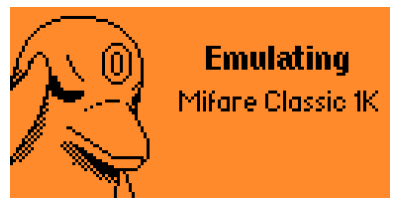
First make sure the UID of the card is BADC0FFE by modifying the UID value and block 0 in the save file as follows:

UID: BA DC 0F FE
Block 0: BA DC 0F FE 84 08 04 00 62 63 64 65 66 67 68 69

The next up is modifying the timestamp to increase so the card is still valid:

Block 30: **FF** 54 B7 F7 00 00 00 00 00 00 00 00 00 00 00 00

As usual upload the file again through qFlipper and write to a card or emulate and you will be awarded some points when checking the flag on the OctoBox.



ACR122U

Read sector 7:

crid --read_sector 7 --key_type A --key_value B2784D1832F8

Displaying sector 7

Block	Data
Block 28	56 61 75 6C 74 20 55 49 44 3A 00 00 BA DC 0F FE
Block 29	56 61 6C 69 64 20 75 6E 74 69 6C 3A 00 00 00 00
Block 30	67 54 B7 F7 00 00 00 00 00 00 00 00 00 00 00 00
Block 31	00 00 00 00 00 00 FF 07 80 69 67 A8 9B 08 C9 34

With data format string:

Displaying sector 7

Block	Data
Block 28	Vault UID:.....
Block 29	Valid until:....
Block 30	gT..............
Block 31	.........ig....4

The text indicated that the UID of the card needs to be BADC0FFE, so lets change the UID of the card

nfc-mfsetuid BADC0FFE

Next up is making sure that the timestamp value is updated

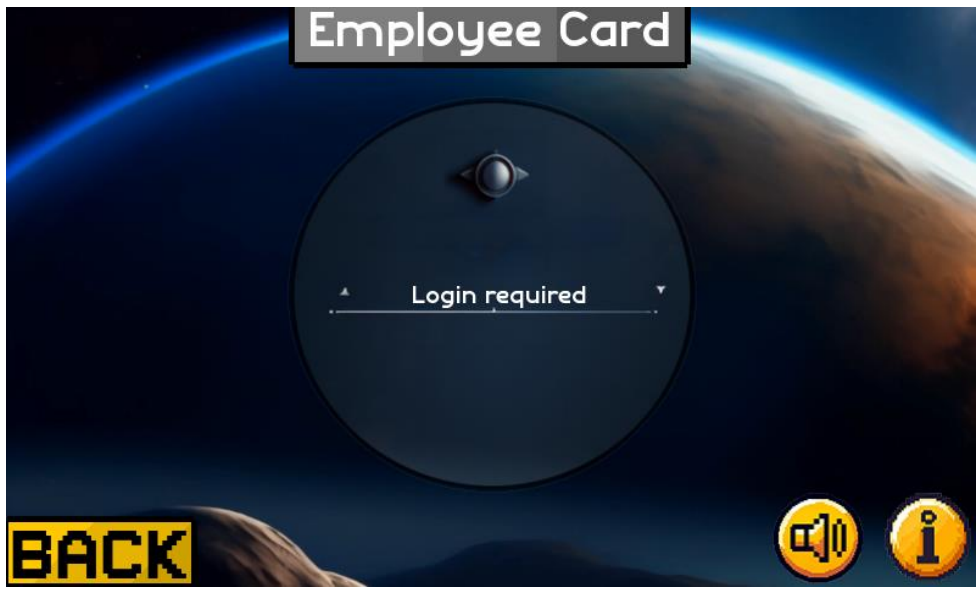
crid --write_block 30 --data FF54B7F70000000000000000000000000 --key_type B --key_value 67A89B08C934

```
INFO: Write successful for block 30.
```

Then test it against the OctoBox and you should get points awarded.

Hard	Challenge #2: Employee Portal	200 POINTS
Devices		
Training Cards	**Card A and B** **Sector 8, Block 34**	

Description



Harper (Card B) lost access to the company login portal! Are you able to restore her access?

1. First make sure you can read sector 8 fully
2. Review the cards and figure out the correct user ID for Harper
3. Create an RFID card for Harper by changing block 34

Only Block 34 needs to be written, block 33 can stay there as reference for the original value. But you can always reset the card if you want to have all data restored.

In order to validate, test the card against the OctoBox.

References

Hint 1	UmV2aWV3IHRoZSBpbmZvcm1hdGlvbiBwcmludGVkIG9uIHRoZSBjYXJkIHRvIGRldGVybWluZSB0aGUgdXNlciBJROKAmXM=
Hint 2	UG9zdG5ldCBCYXJjb2RlIHRvb2xzIG1pZ2h0IGhlbHAgeW91

Walkthrough: Employee Portal

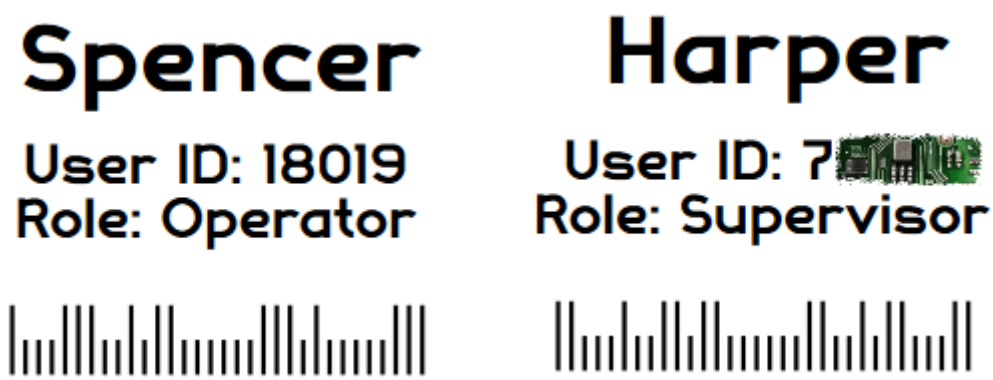
Proxmark3

When you have obtained key B for sector 8 you can read the full contents with the following command:

```
hf mf rdsc -s 8 -b -k BE13377331EB
```

```
[=]   # | sector 08 / 0x08                                | ascii
[=] ----+-----------------------------------------------+----------------
[=]  32 | 55 73 65 72 20 49 44 3A 00 00 00 00 00 00 00 00 | User ID:.......
[=]  33 | 31 38 30 31 39 00 00 00 00 00 00 00 00 00 00 00 | 18019..........
[=]  34 | 31 38 30 31 39 00 00 00 00 00 00 00 00 00 00 00 | 18019..........
[=]  35 | 00 00 00 00 00 00 3F 03 CC 69 00 00 00 00 00 00 | ......?..i.....
```

As you can see the user ID is just put there as ASCI format represented in hex within the sector. The next step would be to identify which user ID to use. When comparing the details printed on the cards you will notice that the same user ID is present on Card A for Spencer.
Harper also has a user ID but its blanked out:



When looking further you will see some specific barcode, which is using POSTNET. When decoding both values on the cards you will have the following:
Card A (Spencer): 18019
Card B (Harper): 73012

Modify that sector with the Proxmark, using the following command:

```
hf mf wrbl --blk 34 -b -k BE13377331EB -d 37333031320000000000000000000000
```

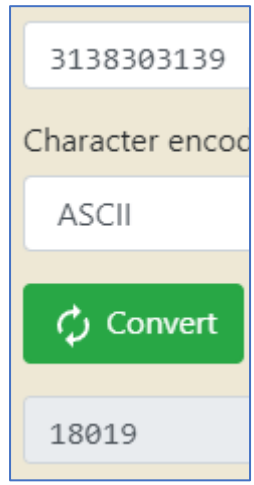
```
[usb] pm3 --> hf mf wrbl --blk 34 -b -k BE13377331EB -d 37333031320000000000
[=] Writing block no 34, key type:B - BE13377331EB
[=] data: 37 33 30 31 32 00 00 00 00 00 00 00 00 00 00 00
[+] Write ( ok )
[?] try `hf mf rdbl` to verify
```

This should provide access to the Employee Portal as Harper for this challenge when scanning your card on the OctoBox.

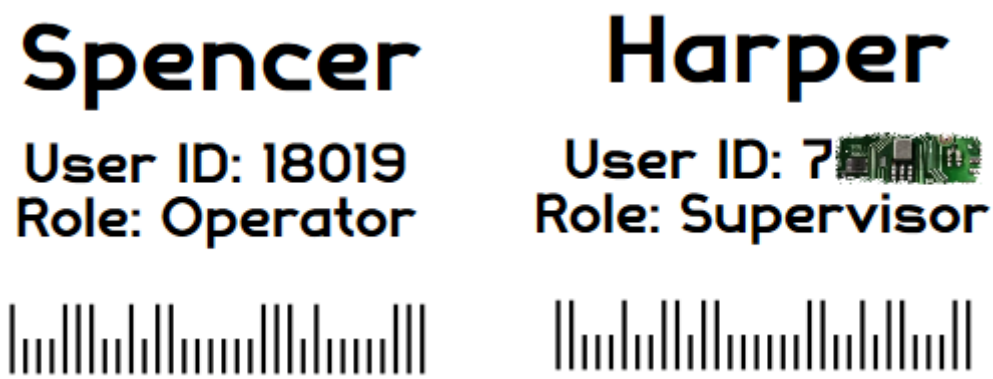
Flipper Zero

When you have obtained key B for sector 8 you can read the full contents:

Block 32: 55 73 65 72 20 49 44 3A 00 00 00 00 00 00 00 00
Block 33: 31 38 30 31 39 00 00 00 00 00 00 00 00 00 00 00
Block 34: **31 38 30 31 39** 00 00 00 00 00 00 00 00 00 00 00



As you can see the user ID is just put there as ASCI format represented in hex within the sector. The next step would be to identify which user ID to use. When comparing the details printed on the cards you will notice that the same user ID is present on Card A for Spencer.
Harper also has a user ID but its blanked out:

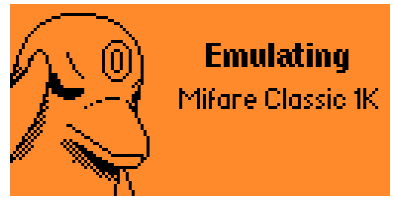


When looking further you will see some specific barcode, which is using POSTNET. When decoding both values on the cards you will have the following:
Card A (Spencer): 18019
Card B (Harper): 73012

Modify block 34 so it states the UID 73012 instead which is 3733303132 in hex:

Block 32: 55 73 65 72 20 49 44 3A 00 00 00 00 00 00 00 00
Block 33: 37 33 30 31 32 00 00 00 00 00 00 00 00 00 00 00
Block 34: **37 33 30 31 32** 00 00 00 00 00 00 00 00 00 00 00

Upload the file back and either emulate or write the contents to the training card. This should provide access to the Employee Portal when scanning your card on the OctoBox.



ACR122U

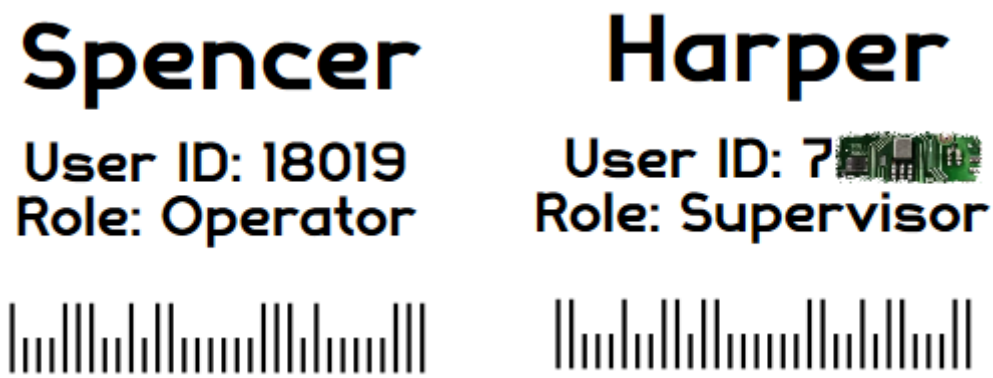
Once you have identified key B for sector 8 you will be able to read the sector fully to analyse the data and determine how to target a specific user ID.

```
crid --read_sector 8 --key_type B --key_value BE13377331EB --data_format string
```

Displaying sector 8

Block	Data
Block 32	User ID:.....
Block 33	18019.....
Block 34	18019.....
Block 35	.....?..i.....

As you can see the user ID is just put there as ASCI format represented in hex within the sector. The next step would be to identify which user ID to use. When comparing the details printed on the cards you will notice that the same user ID is present on Card A for Spencer.
Harper also has a user ID but its blanked out:



When looking further you will see some specific barcode, which is using POSTNET. When decoding both values on the cards you will have the following:
Card A (Spencer): 18019
Card B (Harper): 73012.

Let's write that user ID to the card:

```
crid --write_block 34 --key_type B --key_value BE13377331EB --data
37333031320000000000000000000000
```

This should restore access for Harper when scanning your card on the OctoBox.

Hard	Challenge #3: Vending Machine	250 POINTS
Devices		
Training Cards	**Card A and B** **Sector 10**	

Description

OctoBox has a wonderful vending machine and the two training cards both have some credits assigned. Can you hack your way in to gain access to unlimited credits?

The goal is to scan a card with 1000 or more credits assigned. The vending machine makes use of sector 10.

You might need to **scan the card on the vending machine** to understand how much credits are currently present, which can be useful piece of information to understand the data on the card.

Once you have more than 1000 credits you can buy the flag in the vending machine.

References

Hint 1	VGhlIGxhc3QgYnl0ZXMgcmVwcmVzZW50IGEgY2hlY2tzdW0gY2FsY3VsYXRlZCBvbiB0aGUgZGF0YQ==
Hint 2	VGhlIGNoZWNrc3VtIHVzZXMgWE9S

Walkthrough: Vending Machine

Proxmark3

When you have obtained key B for sector 8 you can read the full contents with the following command:

```
hf mf rdsc -s 10 -b -k AF720E83D0F1
```

```
[=]   # | sector 10 / 0x0A                                      | ascii
[=] ----+-------------------------------------------------------+-----------------
[=]  40 | 56 65 6E 64 69 6E 67 20 6D 61 63 68 69 6E 65 3A | Vending machine:
[=]  41 | 00 1F 00 00 02 26 00 00 00 00 00 00 00 02 39 | .....&.........9
[=]  42 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]  43 | 00 00 00 00 00 00 9F 01 E6 69 AF 72 0E 83 D0 F1 | .........i.r....
```

You can do the same with both cards to obtain the data. If you then scan the card on the vending machine you will notice that the **Card A has 550 credits** and **Card B has 720 credits**.

Card A: 00 1F 00 00 02 26 00 00 00 00 00 00 00 02 39 (hex) → 550 credits [7]
Card B: 00 15 00 00 02 D0 00 00 00 00 00 00 00 02 33 (hex) → 720 credits

When converting some of the values on the card to different data formats you can find that 02 26 can be converted to 550 as decimal. However, when you try to modify the value of those numbers to a higher number, you will see that the vending machine states incorrect checksum.

After some puzzling you will see that the last two bytes are a checksum and is calculated using simple XOR functions:

Bytes 1 and 2 are XOR'ed with bytes 5 and 6 and stored in bytes 15 and 16:

001F XOR 0226 = 0239[8]

To set a very large credit amount you could either calculate the checksum or just leave out the transaction number as 0000 and use the same data for the checksum as the data for the credits:

```
hf mf wrbl --blk 41 -b -k AF720E83D0F1 -d 00000000FFFF0000000000000000FFFF
```

This should provide access to the vault for this challenge when scanning your card on the OctoBox.

Flipper Zero

[7] https://www.binaryhexconverter.com/hex-to-decimal-converter

[8] https://cyberchef.org/#recipe=From_Hex('Auto')XOR(%7B'option':'Hex','string':'001F'%7D,'Standard',false)To_Hex('None',0)To_Upper_case('All')&input=MDIyNg

Reviewing sector 10 on the save file from the Flipper will look as follows:

Block 40: 56 65 6E 64 69 6E 67 20 6D 61 63 68 69 6E 65 3A
Block 41: 00 1F 00 00 02 26 00 00 00 00 00 00 00 02 39
Block 42: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 43: FF FF FF FF FF FF 9F 01 E6 69 AF 72 0E 83 D0 F1

You can do the same with both cards to obtain the data. If you then scan the card on the vending machine you will notice that the **Card A has 550 credits** and **Card B has 720 credits.**

Card A: 00 1F 00 00 02 26 00 00 00 00 00 00 00 02 39 (hex) → 550 credits
Card B: 00 15 00 00 02 D0 00 00 00 00 00 00 00 02 33 (hex) →720 credits

When converting some of the values on the card to different data formats you can find that 02 26 can be converted to 550 as decimal. However, when you try to modify the value of those numbers to a higher number, you will see that the vending machine states incorrect checksum.

After some puzzling you will see that the last two bytes are a checksum and is calculated using simple XOR functions:

Bytes 1 and 2 are XOR'ed with bytes 5 and 6 and stored in bytes 15 and 16:

001F XOR 0226 = 0239[9]

Modify block 41 on the save file from the Flipper:

Block 40: 56 65 6E 64 69 6E 67 20 6D 61 63 68 69 6E 65 3A
Block 41: 00 00 00 00 FF FF 00 00 00 00 00 00 00 FF FF
Block 42: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Block 43: FF FF FF FF FF FF 9F 01 E6 69 AF 72 0E 83 D0 F1

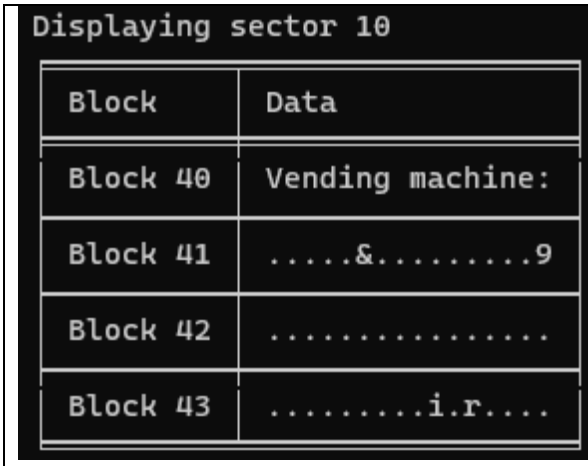
Upload the file back and either emulate or write the contents to the training card. This should give you enough points to win the challenge, when scanning your card on the OctoBox.

ACR122U

Once you have identified key B for sector 10 you will be able to read the sector fully to analyse the data and attempt to understand how the vending machine creates a transaction.

```
crid --read_sector 10 --key_type B --key_value AF720E83D0F1 --data_format string
```

[9] https://cyberchef.org/#recipe=From_Hex('Auto')XOR(%7B'option':'Hex','string':'001F'%7D,'Standard',false)To_Hex('None',0)To_Upper_case('All')&input=MDIyNg



As you can see the first block is just stating the challenge name but the second does not make much sense. But lets review the hex representation of Block 41.

Initate the command to read block 41

```
crid --read_block 41 --key_type B --key_value AF720E83D0F1
```

You can do the same with both cards to obtain the data. If you then scan the card on the vending machine you will notice that the **Card A has 550 credits** and **Card B has 720 credits**.

Card A: 00 1F 00 00 02 26 00 00 00 00 00 00 00 00 02 39 (hex) → 550 credits [10]
Card B: 00 15 00 00 02 D0 00 00 00 00 00 00 00 00 02 33 (hex) → 720 credits

When converting some of the values on the card to different data formats you can find that 02 26 can be converted to 550 when taken as an binary number. However, when you try to modify the value of those numbers to a higher number, you will see that the vending machine states incorrect checksum.

After some puzzling you will see that the last two bytes are a checksum and is calculated using simple XOR functions:

Bytes 1 and 2 are XOR'ed with bytes 5 and 6 and stored in bytes 15 and 16:

001F XOR 0226 = 0239[11]

To set a very large credit amount you could either calculate the checksum or just leave out the transaction number as 0000 and use the same data for the checksum as the data for the credits:

```
crid --write_block 41 --key_type B --key_value AF720E83D0F1 --data
00000000FFFF00000000000000FFFF
```

This should provide access to the vault for this challenge when scanning your card on the OctoBox.

[10] https://www.binaryhexconverter.com/hex-to-decimal-converter

[11] https://cyberchef.org/#recipe=From_Hex('Auto')XOR(%7B'option':'Hex','string':'001F'%7D,'Standard',false)To_Hex('None',0)To_Upper_case('All')&input=MDIyNg

Hard	Challenge #4: Hotel Rooms	350 POINTS
Devices		
Training Cards	**Card A and B** **Sector 11**	

Description

You have access to your hotel room in room number 419 using Card A (Spencer) and coincidently you also found an expired badge Card B (Harper) close to your hotel room door. Can you understand how the door verifies that you have access to the room and obtain access to room number 420?

With the access keys known you would read the data from both cards and attempt to understand what the data represents. Cross-referencing any known information in different data formats and modifying the data slightly while observing any behavior changes. Detect any potential data validation controls.

Important information:

Assume the current date is: **10 August 2024**
And we booked a reservation between: **8 and 11 August 2024**

You gain the points if you can open hotel room 420.

References

Hint 1	QmxvY2sgNDUgY29udGFpbnMgaG90ZWwgcm9vbSBudW1iZXIsIHN0YXJ0IGFuZCBlbmQgZGF0ZSBvZiByZXNlcnZhdGlvbg==

Hint 2	U29tZSBvbGQgaGFzaGluZyBmdW5jdGlvbiBuZWVkcyB0byBiZSB1c2Vk
Hint 3	TUQ1IGlzIHlvdXIgZnJpZW5kIGhlcmU=

Walkthrough: Hotel Rooms

Proxmark3

With the access keys known you would read the data from both cards and attempt to understand what it would mean.

```
hf mf rdsc -s 11 -b -k 14318D91BFE5
```

Card A:

```
[=]   # | sector 11 / 0x0B                                | ascii
[=] ----+-----------------------------------------------+----------------
[=]  44 | 53 65 63 75 72 65 20 48 61 73 68 20 44 61 74 61 | Secure Hash Dat
[=]  45 | 01 A3 66 B4 97 26 6B 88 BA 00 00 00 00 00 00 00 | ..f..&k........
[=]  46 | DB 94 EF 4E 88 1F 97 3B 4C 23 40 10 51 99 B7 1D | ...N...;L#@.Q..
[=]  47 | 00 00 00 00 00 00 F0 FF 00 69 00 00 00 00 00 00 | .........i.....
```

Card B:

```
  #  | sector 11 / 0x0B                                | ascii
-----+-----------------------------------------------+----------------
 44  | 53 65 63 75 72 65 20 48 61 73 68 20 44 61 74 61 | Secure Hash Data
 45  | 01 A4 66 B1 F4 20 66 B4 97 20 00 00 00 00 00 00 | ..f.. f.. .....
 46  | C2 9F 4E 1A 6C A8 34 C0 B1 37 72 87 FF 31 5A F0 | ..N.l.4..7r..1Z
 47  | 00 00 00 00 00 00 F0 FF 00 69 00 00 00 00 00 00 | .........i.....
```

Card A:
01A3 → 419 (Room)
66B49720 → Thursday, August 8, 2024 10:00:00 AM
66B88BA0 → Sunday, August 11, 2024 10:00:00 AM

Card B:
01A4 → 420 (Room)
66B1F420 → Tuesday, August 6, 2024 10:00:00 AM
66B49720 → Thursday, August 8, 2024 10:00:00 AM

Block 46 is currently unkown, but block 44 indicates it might be a hash. When **MD5** hashing the data from block 45 it matches with the data of block 46.

All this information can be pieced together to create a modified card. First create the block data for room 420, with the timestamps of Card A and write it to block 45.

```
hf mf wrbl --blk 45 -b -k 14318D91BFE5 -d 01A466B4972066B88BA0000000000000
```

```
[usb] pm3 --> hf mf wrbl --blk 45 -b -k 14318D91BFE5 -d 01A466B4972066B88BA0000
[=] Writing block no 45, key type:B - 14318D91BFE5
[=] data: 01 A4 66 B4 97 20 66 B8 8B A0 00 00 00 00 00 00
[+] Write ( ok )
```

This is still not enough since you would need to change the MD5 hash as well, calculate the hash with Python as follows:

import hashlib
hashlib.md5(bytes.fromhex("01A466B4972066B88BA0000000000000")).digest().hex().upper()

Results in **ADE530843136A5AC3E31AA6F6BC7B4E7**

So write that value to block 46:

hf mf wrbl --blk 46 -b -k 14318D91BFE5 -d ADE530843136A5AC3E31AA6F6BC7B4E7

```
[usb] pm3 --> hf mf wrbl --blk 46 -b -k 14318D91BFE5 -d ADE530843136A5AC3E31
[=] Writing block no 46, key type:B - 14318D91BFE5
[=] data: AD E5 30 84 31 36 A5 AC 3E 31 AA 6F 6B C7 B4 E7
[+] Write ( ok )
```

This should provide access to hotel room 420.

Flipper Zero

Read out the save files for sector 11 for both Card A and B and compare the data:

Card A:
01A3 → 419 (Room)
66B49720 → Thursday, August 8, 2024 10:00:00 AM
66B88BA0 → Sunday, August 11, 2024 10:00:00 AM

Card B:
01A4 → 420 (Room)
66B1F420 → Tuesday, August 6, 2024 10:00:00 AM
66B49720 → Thursday, August 8, 2024 10:00:00 AM

Block 46 is currently unkown, but block 44 indicates it might be a hash. When **MD5** hashing the data from block 45 it matches with the data of block 46.

All this information can be pieced together to create a modified card. First create the block data for room 420, with the timestamps of Card A and write it to block 45.

So modify a save file to include the following blocks:

Block 45: 01 A4 66 B4 97 20 66 B8 8B A0 00 00 00 00 00 00
Block 46: AD E5 30 84 31 36 A5 AC 3E 31 AA 6F 6B C7 B4 E7

Upload the file back and either emulate or write the contents to the training card. This should give you enough points to win the challenge, when scanning your card on the OctoBox.

ACR122U

With the access keys known you would read the data from both cards and attempt to understand what it would mean.

```
crid --read_sector 11 --key_type B --key_value 14318D91BFE5 --data_format string
```

Card A:

Displaying sector 11

Block	Data
Block 44	Secure Hash Data
Block 45	..f.. f.....
Block 46	...N...;L#@.Q...
Block 47	.....i.....

Card B:

Displaying sector 11

Block	Data
Block 44	Secure Hash Data
Block 45	..f..&kIr.....
Block 46	..N.l.4..7r..1Z.
Block 47	.....i.....

Card A:
01A3 → 419 (Room)
66B49720 → Thursday, August 8, 2024 10:00:00 AM
66B88BA0 → Sunday, August 11, 2024 10:00:00 AM

Card B:
01A4 → 420 (Room)
66B1F420 → Tuesday, August 6, 2024 10:00:00 AM
66B49720 → Thursday, August 8, 2024 10:00:00 AM

Block 46 is currently unkown, but block 44 indicates it might be a hash. When **MD5** hashing the data from block 45 it matches with the data of block 46.

All this information can be pieced together to create a modified card. First create the block data for room 420, with the timestamps of Card A and write it to block 45.

```
crid --write_block 45 --key_type B --key_value 14318D91BFE5 --data
01A466B4972066B88BA0000000000000
INFO: Write successful for block 45.
```

This is still not enough since you would need to change the MD5 hash as well, calculate the hash with Python as follows:

import hashlib
hashlib.md5(bytes.fromhex("01A466B4972066B88BA0000000000000")).digest().hex().upper()

Results in **ADE530843136A5AC3E31AA6F6BC7B4E7**

So write that value to block 46:

```
crid --write_block 46 --key_type B --key_value 14318D91BFE5 --data
ADE530843136A5AC3E31AA6F6BC7B4E7
INFO: Write successful for block 46.
```

This should provide access to hotel room 420.

Hard	Challenge #5: Speedrun	0 POINTS
Devices		
Training Cards	**Speedrun Challenge Card (Pokemon or Yu-Gi-Oh Theme Card)**	

Description

For this challenge you will need to clone a configured badge as fast as possible in any way you want, but with one rule:

It's not allowed to write or change the challenge card.

Make sure you are using a wiped challenge card provided by the instructor.

1) Place the challenge card on the reader and wait until the game screen highlights Ready and the play button becomes enabled.
2) Press the play button, where the card will be configured and as soon as the timer starts you can take the card and clone/emulate the card as fast as possible.
3) Place your cloned card on the reader when finished and observe your time.

Goodluck!