The following full text is a preprint version which may differ from the publisher's version.

For additional information about this publication click this link.
http://hdl.handle.net/2066/94153

Please be advised that this information was generated on 2017-11-30 and may be subject to change.
Abstract. This paper gives a hands-on introduction to the Proxmark, a versatile tool for RFID security research. It can be used to analyze and reverse engineer RFID protocols deployed in billions of cards, tags, fobs, phones and keys. We give a heads up introduction on how to embed new modulation and encoding schemes into the Proxmark, which helps to get a grip on the low level RF-communication details. As example we point out several (devastating) weaknesses which are made at this low levels. Most notably the MIFARE Classic with its weakly encrypted parity bits, which enables an attacker to recover the secret key. Furthermore, we describe the practical cryptanalysis of several proprietary RFID protocols and ciphers. In this part we introduce the Proxmark as an effective attack tool that can perform practical attacks a hundred times faster than regular RFID readers.

Keywords: Proxmark, RFID, NFC, contactless smartcards, open-source

1 Introduction

Radio Frequency Identification (RFID) is one of the most pervasive technologies nowadays. It was first introduced for identification purposes only, but quickly expanded to other applications like transport ticketing systems and access control. The security and privacy of these systems is often overlooked. This is, to a good extend, due to the fact that it is hard to know what are the underlying security mechanisms employed. A surprisingly large number of access control systems use the tag’s unique identifier (UID) as their only security mechanism. Moreover, large scale ticketing systems use simple memory cards \cite{1,46,52,55} for fare collection. This type of cards lack any cryptographic capabilities and therefore the security of these systems relies on UID blacklisting mechanisms. When portable tag-emulating devices are available \cite{11,18,39,42,57,58}, these security mechanisms become obsolete.

Many RFID tags and contactless smart cards use proprietary security mechanisms for authentication and confidentiality. Since these tags are widely deployed in access control and ticketing systems, it is important to independently assess their security. This paper explains how the Proxmark device can be used for RFID protocol analysis and to exploit implementation attacks. It facilitate message eavesdropping and emulation of both tags and readers. We analyse various communication protocols that operate at a low frequency (125kHz) and high frequency (13.56MHz). The Proxmark supports all major modulation and encoding schemes. Therefore, it is able to communicate with many different proprietary communication protocols used by various RFID tags. These tools are fully programable and allow for quick prototyping, testing and debugging of new RFID protocols, like proposed in \cite{1,13,19,21,25,56}. All the software, firmware and hardware that is described in this paper is open source and open design.

This paper is organized as follows. Section 2 starts with some background information about the Proxmark device. Section 3 addresses the motivation why RFID protocol research is useful and necessary. Next, the hardware and basic components of the Proxmark device are introduced in Section 4. Finally, we select in Section refsec:usecases two interesting use-cases and show how to analyse and attack widely deployed RFID products.
2 Background

The Proxmark III has been developed by Jonathan Westhues. The Proxmark III, shown in Figure 2.1, replaces its predecessors and introduces a high level of flexibility in both signal processing and protocol implementation. It is additionally equipped with a Field Programmable Gate Array (FPGA) which is mainly responsible for the low-level signal processing and allows to set up multiple signal processing schemes. In general, when we speak about the Proxmark, we refer to this latest version.

Fig. 2.1. The Proxmark III

The hardware design and firmware of this latest version is in the public domain since May 2007 under the General Public License. The device costs around €200 and since the schematics are online, it can be ordered through any local printed circuit board (PCB) supplier. Although, most assembled Proxmark devices are sold by one of the main suppliers: Rysc Corp., GeZhi Electronic Corp. Ltd. and hackable-devices. The following websites contain all the information that is required to assemble, compile, flash, use and develop new features for the Proxmark.

- http://eq.cx/proxmark3.pl The first website about the Proxmark device, created by Jonathan Westhues in 2007. Jonathan made the project free to use and published all the necessary designs and source codes. Five years later, already more than a thousand Proxmarks were sold for extensive RFID protocol and security research.
- http://www.proxmark.org Contains a lot of information about new RFID modulation, encoding and protocols that were added the last years. This website hosts the main community forum, which is currently used by more than 3000 members. This forum answers all frequently asked questions concerning the Proxmark, but also contains various topics about microcontroller and FPGA development.
- http://proxmark3.googlecode.com This is the development website which hosts the most recent subversion (SVN) repository. Only in 2012 there are already 26 active committers, who regularly fix problems and contribute new features to the Proxmark firmware. The website also hosts a small wiki that contains a manual for using the Proxmark device. Most features and commands are explained in detail, backed up by several output examples and pictures.

1 http://www.proxmark3.com
2 http://www.xfpga.com
3 http://www.hackable-devices.org
Throughout this paper we focus on contactless smart cards. For contact based smart cards there are comparable tools available in the literature [8,12] which allows eavesdropping, emulation, man-in-the-middle attacks and fast querying by using a dedicated FPGA.

3 Motivation

Manufacturers often claim that their tags provide ‘state-of-the-art’, ‘field-proven’ or ‘unbreakable’ security, but it is hard to know what this means and how much security you actually get. The widely used RFID communication standards like [33–37] define the low-level transmission layers. However, these standards do not include any details of the secure communication layer. Semiconductor companies are inclined to create ad-hoc RFID designs that use proprietary protocols and cryptographic algorithms [2,16,31,32,44,47–49,51,54]. Such designs are often kept secret to provide security-through-obscurity. It has been shown many times that without feedback from the scientific community, it is hard to build secure algorithms [38,41]. There are numerous examples in the literature [3–7,9,10,14,15,20,22–24,26–30,40,43,45,50,53,59–62] showing that once the secrecy of an algorithm is lost, so is its security. As long as RFID tags do not comply with open and community-reviewed encryption standards, the security of these tags need to be independently assessed. In order to perform these assessments, we need tools to analyze the underlying security protocols. While designing new RFID products and protocols, it is also useful to have a set of tools at hand for easy protocol prototyping, testing and debugging. There are several reasons for this kind of “analytic” research:

1. **Public scrutiny.** Unlike other fields, it is not really possible to prove the security of a secure design/product. The way in which security research progresses is by proposing new constructions and then exposing them to the community for critical scrutiny.
2. **Informing consumers.** Manufacturers often boost “unbreakable security” of their products (while they often can be hacked within a few seconds). These claims lead consumers (as non-security experts) to use such products in very critical/security sensitive applications.
3. **Higher standards.** The industry is reluctant to improve their products, even when they are informed of their weaknesses. Only when customers demand better products industry is willing to improve.
4. **Responsible disclosure.** When weaknesses are found, it is important to inform the manufacturer ahead of disclosure such that they can take the necessary measures. Without scientific scrutiny, weaknesses are often found too late, when fraud/misuse has already taken place.

As long as RFID tags do not comply with open and community-reviewed encryption standards, the security of these tags need to be independently assessed. In order to perform these assessments, we need a powerful and flexible tool to analyse the underlying security protocols. The Proxmark device is such a tool. It gives direct access to the real-time data and timing information which is very useful during protocol analysis. Furthermore, it allows the user to create a dedicated firmware that can quickly query and repeat a modulation and encoding scheme. This particular function can be used to perform practical attacks a hundred times faster than regular RFID readers.

4 Proxmark

4.1 Hardware

The Proxmark III supports both low (125 kHz-134 kHz) and high frequency (13.56 MHz) signal processing. This is achieved by two parallel antenna circuits that can be used independently. Both circuits are connected to a 4-pin Hirose connector to connect an external loop antenna. When the Proxmark is in **reader mode** it drives the antenna coils with the appropriate frequency. This is unnecessary when the Proxmark works in **eavesdropping mode** or in **card emulation mode** because then the electromagnetic field is generated by the reader. The signal from the antenna is routed through the FPGA after it has been digitized by an 8-bit Analog-to-Digital Converter (ADC).
After some filtering, the FPGA relays the necessary information to perform the decoding of the signal to the microcontroller. This prevents the microcontroller from being overloaded with signal data. An FPGA has a great advantage over a normal microcontroller in the sense that it emulates hardware. A hardware description can be compiled and flashed into an FPGA. Basic arithmetic operations can be performed in parallel and faster than in a microcontroller. An FPGA is of course slower than a hardware implementation but pure hardware lacks flexibility.

The microcontroller is responsible for the protocol part. It receives the digital signal from the FPGA and decodes it. The decoded signal can just be copied to a buffer in the EEPROM memory. Additionally, an answer to an incoming message can be programmed to be sent immediately, communicating this to the FPGA which then modulates the appropriate signal.

The Proxmark has a USB interface to the computer. The current implementation uses the default Human Interface Device (HID) USB protocol. Flashing of the microcontroller and the FPGA can be done via USB. Only the first time the JTAG interface is used to set up a bootloader on the microcontroller. The hardware design that can be flashed into the FPGA is written in Verilog. Verilog is a hardware description language which allows to describe a hardware design in a C-style syntax.

4.2 Software

The Proxmark can operate in three different modes: sniffing mode; card emulation mode; and reader mode. It is possible to use the Proxmark for very different modulation schemes and protocols as long as there are in the supported frequency range. Some well known protocols and modulation schemes are already available. There are some requirements to implement the mentioned modes for new protocols. First, we need an underlying physical layer which takes care of the Digital Signal Processing (DSP). Next, the modes of operation should be implemented as functions on the microcontroller. Finally, the client should be able to call these functions and display the results.

The processing and generation of the protocol messages is partly done by the FPGA and partly by the microcontroller. The FPGA deals with signal processing issues like edge detection and then communicates the result to the microcontroller. The microcontroller then tries to decode the bit stream depending on the modulation scheme. In order to generate a signal the microcontroller will send a bit stream to the FPGA. This stream is encoded using the corresponding modulation scheme, e.g., Manchester or Modified Miller. The FPGA modulates according to this bit stream. The decision to split the DSP in two parts is mainly because of the limited capacity of the FPGA. It cannot do signal processing and message decoding/encoding at the same time.

The microcontroller implements the transport layer. First it decodes the samples received from the FPGA. These samples are stored in a Direct Memory Access (DMA) buffer. The samples are binary sequences that represent whether the signal was high or low. The software on the microcontroller tries to decode these samples. When the Proxmark is in sniffing mode this is done for both the reader and tag signal at the same time. Whenever one of the decoding procedures returns a valid message, this message is stored in another buffer (BigBuf). The BigBuf is especially useful for protocol analysis. Every single message is stored in this buffer. When a card is emulated or when the Proxmark is used as a reader the BigBuf can be used to store status messages or protocol exceptions.

The client application works as a console application and connects to the Proxmark via the standard HID USB protocol. The microcontroller continuously polls for new USB packets. Currently it is not possible to stream the retrieved samples directly to the PC in real-time. When the microcontroller retrieves a command from the client, it runs this command and stores any resulting messages in its memory buffer. Next, the client sends a second command to retrieve the data from this buffer.

5 Use-cases

This section shows how to use the Proxmark in practice, by using it for two concrete use-cases. The first use-case exploits one of the many vulnerability that were found in the MIFARE Classic
It shows how to recover the secret key from a building access control reader, by using only one eavesdropped authentication trace. In the second use-case is shown how to recover the secret master key from an iClass Elite access control reader using the Proxmark as a tag emulator.

### 5.1 MIFARE Classic

Enter the following command to start eavesdropping ISO/IEC 14443 type A communication:

```
hf 14a snoop
```

Hold the Proxmark antenna next to the reader and present the MIFARE Classic tag. Blinking of the lights indicate transmission was captured. Press the button on the Proxmark to stop the gathering frames or wait until the buffer is full. The trace most likely contains more than only the authentication information. Before the reader can exchange messages with a MIFARE Classic tag, it needs to perform the anti-collision protocol, see [36]. Most of these messages can be ignores, except second message from the tag, which reveals the unique identifier (UID) of the tag. To retrieve the eavesdropped trace from the Proxmark, the following command can be used:

```
hf 14a list
```

If the trace is received correctly, it looks very similar to the example trace which is shown below. If the trace partial frames, unexpected messages, not a clear message toggling between reader and tag, then the reception was not good enough. Try a different angle, more/less space between reader-antenna-tag, various speed to present the tag. If the antenna is tuned enough, it should pick up the signal and produce a similar trace to the one below.

```
+ 50782: : 26
+ 33822: : 26
+ 50422: : 26
+ 64: 0: TAG 04 00
+ 944: : 93 20
+ 64: 0: TAG 9c 59 9b 32 6c
+ 1839: : 93 70 9c 59 9b 32 6c 6b 30
+ 64: 0: TAG 08 b6 dd
+ 3783: : 60 32 64 69
+ 113: 0: TAG 82 a4 16 6c
+ 1287: : a1 e4 58 ce 6e ea 41 e0
+ 64: 0: TAG 5c ad f4 39
```

With this information it is possible to execute the `mfkey` tool accordingly:

```
./mfkey <uid> <nt> <{nr}> <{ar}> <{at}>
./mfkey 9c599b32 82a4166c a1e458ce 6eea41e0 5cadf439
```

Which results for this example in the following output:

```
MIFARE Classic key recovery

Recovering key for:
  uid: 9c599b32
    nt: 82a4166c
  {nr}: a1e458ce
  {ar}: 6eea41e0
  {at}: 5cadf439
```

LFSR successors of the tag challenge:

- \( nt' \): 8d65734b
- \( nt'' \): 9a427b20

Keystream used to generate \( \{ar\} \) and \( \{at\} \):
- \( ks2 \): e38f32ab
- \( ks3 \): c6ef8f19

Found Key: \([ff \ ff \ ff \ ff \ ff \ ff]\)

### 5.2 iClass

Enter the following command to start emulating the card serial number (CSN) of an iClass Elite card.

```
hf iclass sim 0 031FEC8AF7FF12E0
```

After presenting the Proxmark to an iClass Elite reader, it tries to authenticate the (emulated) card with random challenges. The Proxmark captures the authentication attempts which are used to recover the secret master key. The output that is generated by the Proxmark is shown as follows:

```
--simtype:00 csn:03 1f ec 8a f7 ff 12 e0
#db# READER AUTH (len=09): 05 00 00 00 00 bf 5d 67 7f
```

At least one authentication attempt for the following list is required:

<table>
<thead>
<tr>
<th>Card Serial Number (CSN)</th>
<th>recovers matrix byte index</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 0B 0F FF F7 FF 12 E0</td>
<td>00 01</td>
</tr>
<tr>
<td>00 04 0E 08 F7 FF 12 E0</td>
<td>02</td>
</tr>
<tr>
<td>00 09 OD 05 F7 FF 12 E0</td>
<td>03</td>
</tr>
<tr>
<td>00 OA 0C 06 F7 FF 12 E0</td>
<td>04</td>
</tr>
<tr>
<td>00 0F OB 03 F7 FF 12 E0</td>
<td>05</td>
</tr>
<tr>
<td>00 08 OA 0C F7 FF 12 E0</td>
<td>06</td>
</tr>
<tr>
<td>00 OD 09 09 F7 FF 12 E0</td>
<td>07</td>
</tr>
<tr>
<td>00 0E OB 0A F7 FF 12 E0</td>
<td>08</td>
</tr>
<tr>
<td>00 03 07 17 F7 FF 12 E0</td>
<td>09</td>
</tr>
<tr>
<td>00 3C 06 E0 F7 FF 12 E0</td>
<td>10</td>
</tr>
<tr>
<td>00 01 05 1D F7 FF 12 E0</td>
<td>11</td>
</tr>
<tr>
<td>00 02 04 1E F7 FF 12 E0</td>
<td>12</td>
</tr>
<tr>
<td>00 07 03 1B F7 FF 12 E0</td>
<td>13</td>
</tr>
<tr>
<td>00 00 02 24 F7 FF 12 E0</td>
<td>14</td>
</tr>
<tr>
<td>00 05 01 21 F7 FF 12 E0</td>
<td>15</td>
</tr>
</tbody>
</table>

These bytes reveal the first line of the matrix that is constructed by using the master key and some weak combination of encrypting itself. With this information it is possible to brute force the key with a total complexity of only \(2^{25}\) DES encryptions.

### References


