Eavesdropping on and Emulating MIFARE Ultralight and Classic Cards Using Software Defined Radio

Ilias Giechaskiel[1]

Abstract
In this report, we describe a Software-Defined Radio (SDR) approach for eavesdropping on Near Field Communications (NFC) and Radio Frequency Identification (RFID) cards operating at 13.56 MHz. We show that GNU Radio and Python make a great platform for prototyping, while maintaining sufficient performance for passive attacks without extensive optimizations and using only modest processing power. We successfully eavesdrop on real MIFARE Ultralight and Classic 1K cards by capturing the raw radio waves with a home-made antenna. We recover the plaintext of both reader and tag fully by demodulating the incoming radio waves, parsing individual bits and error detection codes into packets, and then decrypting them if necessary. On the transmission side, we achieve full software emulation of the reader and of MIFARE Ultralight and Classic 1K cards (including encryption), and partial hardware emulation, where we correctly modulate the signal, but not within the strict timing limits of the protocol. Our transmissions can also be used to prevent legitimate communication by interfering with the intended reader or tag signals.

[1] *CDT in Cyber Security, University of Oxford, Oxford, United Kingdom*

1. Introduction

Contactless cards and tags have become very popular in recent years, with everyday applications including e-passports [25], ticketing [26, 6, 8], access control [27], and payment [16, 7] systems. However, as these devices operate wirelessly, adversaries can pick up the radio signals and eavesdrop on the communication between a tag and a reader. Traditionally, such attacks on radio communications required dedicated hardware for particular frequencies and modulation types, but with the advent of Software-Defined Radio (SDR), it is possible to use generic equipment and perform the demodulation in software. Even so, despite a range of embedded devices and Field-Programmable Gate Arrays (FPGAs) that are capable of various attacks on Near Field Communication (NFC), Radio Frequency Identification (RFID), and related technologies, to the best of our knowledge no open-source SDR implementation exists for High-Frequency (HF) NFC.[1]

To this end, we developed such an implementation on an Ettus Research Universal Software Radio Peripheral (USRP) using Python and GNU Radio with an antenna made out of simple wire that allows passive eavesdropping on reader-tag communication. Though our implementation is easily extensible, we focused on MIFARE cards by NXP Semiconductors, since MIFARE has “a market share of more than 77% in the transport ticketing industry”, with “150 million reader and 10 billion contactless and dual interface IC’s sold” [19]. Specifically, we use Ultralight [21] and Classic 1K [20] cards, as the former does not employ any encryption, while the latter uses a broken cryptographic algorithm (Section **??**), making them ideal candidates for such exploration. Moreover, we achieve full software and partial hardware reader and tag emulation, that can also be used to jam signals between a legitimate tag and reader. In summary, our contributions are as follows:

1. We implement in pure Software-Defined Radio a demodulator for NFC/RFID readers and tags operating in the 13.56 MHz frequency, which decodes radio waves into plaintext packets.
2. We test our implementation by eavesdropping on real MIFARE Classic 1K and Ultralight communications with an RFID reader using a home-made antenna and a USRP, successfully decoding any encrypted packets.
3. We additionally implement in software the emulation of both readers and tags, including encryption if necessary.
4. Though our transmission capabilities cannot keep up with the strict timing requirements of the protocol, we show how our implementation can jam real reader-tag communications and prevent the successful transmission of data.
5. Overall, our work shows that prototyping using Software-Defined Radio is sufficient in practice for passive attacks, without the need for extensive optimizations or heavy computing power.

PAPER STRUCTURE

2. Related Work

Early work on RFID Hacking was conducted in a non-academic context, and focused on finding vulnerabilities in access control systems [23, 26]. Later, Buettner and Wetherall [4, 5, 3] experimented more systematically with RFID and SDR, but focused primarily on Gen 2 cards operating at 900 MHz.

[1] Though they exist for UHF Gen2 cards. See `https://github.com/brunoprog64/rfid-gen2` and `https://github.com/yqzheng/usrp2reader` for instance.

Their work was extended by others, typically in the context of proposing better protocols [2, 29], but still for Ultra High Frequencies (UHF), with the exception of a recent work by Hassanieh et al. [10], which also included an extension to HF.

There have also been a number of designs which use microcontrollers and Field-Programmable Gate Arrays (FPGAs) for signal processing, such as the Proxmark 3 [28], and RFIDler [15]. Though such projects allow the use of custom firmware for additional functionality, they also require dedicated hardware in their design.

The MIFARE Classic cryptographic protocol was reverse-engineered by Nohl et al. by dissolving the plastic surrounding the chips, recovering the individual logic gates and converting them to a high-level algorithm [18]. Garcia et al. then discovered additional vulnerabilities of the protocol based on its nested authentication and parity bits [8]. Due to the wide range of applications of the MIFARE Classic, the topic became very popular for Master's thesis projects [24, 6, 22, 27], which found additional vulnerabilities, or examined the problem within the context of a specific application.

Finally, given the widespread availability of NFC-enabled mobile devices, researchers have also focused on NFC relay attacks using mobile phones [16, 7], as well as exploring [17] and protecting [9] the NFC mobile phone stack.

3. Background

The terms Radio Frequency Identification (RFID), Near Field Communication (NFC), contactless smartcards, proximity cards and vicinity cards are often used interchangeably, but they are covered by different standards and concern different parts of the radio spectrum. In this project, we looked at the ISO/IEC 14443 standard, with physical characteristics defined in [11], modulation and encoding in [13], initialization and anticollision in [14] and transmission protocols in [12].

Specifically, we focus on **Type A** communications, whose **carrier frequency** is $f_c = 13.56$ MHz. The reader – or **Proximity Coupling Device (PCD)** – communicates with the card – or **Proximity Integrated Circuit Card (PICC)** – through **100% Amplitude Shift Keying (ASK)**, with data using the **Modified Miller** encoding. The communication from tag to reader utilizes **Load Modulation** with a **subcarrier frequency** of $f_s = f_c/16 = 847.5$ kHz, using **Manchester Encoding**, and both transmit at a rate of 106 kbit/s.

Because the carrier frequency is $f_c = 13.56$ MHz, the wavelength is $c/f_c \approx 22$ meters, making it impossible to deploy antennas that would fit in a card-size form-factor. Additionally, because the cards are **passive** (i.e. do not have their own power source), both the communication and the power source are achieved through **inductive coupling** from the PCD's antenna loop to the PICC's antenna loop.

SECTION STRUCTURE

3.1 PCD Transmissions

Amplitude Shift Keying (ASK) of depth $X\%$ is a form of digital modulation which specifies that if the amplitude of the signal representing a digital 1 is equal to A and the amplitude of the signal representing a digital 0 is equal to B, then $X = \frac{A-B}{A+B}$, as showing in Figure 1. Specifically, for 100% ASK, no signal is sent at all during a digital 0, indicating that such periods must be very brief, since the PICC needs to keep charge (using a capacitor) for the period of silence.

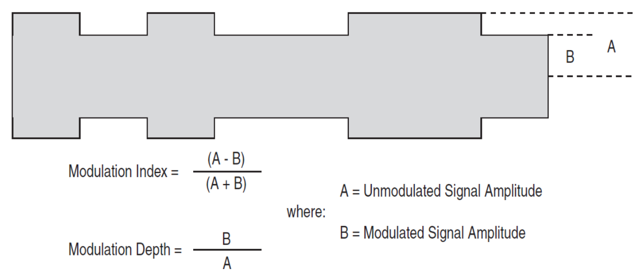

Figure 1. Amplitude Shift Keying (ASK) [1]

This is achieved through the Modified Miller Encoding, which ensures that there is no period of more than $3\mu s$ of silence ("pause"). Specifically, every bit is represented as a (combination of) signals lasting a total of $t_b = 128/f_c \approx 9.44\mu s$. The encoding (show in Figure 2) is as follows:

- A 1 is encoded as an unmodulated signal for $t_b/2 \approx 4.72\mu s$, followed by a period of silence for $3\mu s$, followed by an unmodulated signal for $t_b/2 - 3 \approx 1.72\mu s$
- A 0 after a 0 is encoded as a silence period for $3\mu s$ followed by an unmodulated signal for $t_b - 3 \approx 6.44\mu s$
- A 0 after a 1 is encoded as an unmodulated signal for a period of $t_b \approx 9.44\mu s$
- To indicate the beginning and the end of a transmission, a logical 0 is used for both the start and the end

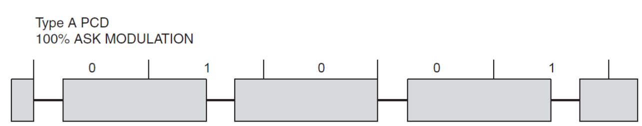

Figure 2. Miller Encoding at 100% ASK [1]

In practice, however, because of hardware imperfections, the pause is not perfect, but needs to comply with the requirements shown in Figure 3. As a result, the modulated carrier for the encodings resembles Figure 4.

3.2 PICC Transmissions

As mentioned above, the tag does not have sufficient power for active transmissions. Consequently, the PICC achieves data transmission passively, by changing its *load*, which can be inferred as a voltage drop on the PCD, hence the term **load modulation**. Switching the load generates a **subcarrier**, which has a frequency $f_s = f_c/16 = 847.5$ kHz.

The bits are then encoded using **On-Off Keying (OOK)** or **Manchester Encoding** as follows, with a total duration also equal to $t_b \approx 9.44\mu s$, which also equals 8 periods of the subcarrier, also shown in Figures 5 and 6:

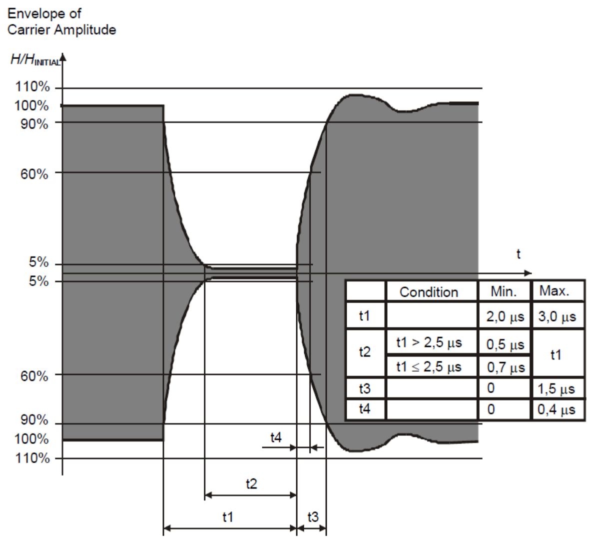

Figure 3. Real Pause Requirements [13]

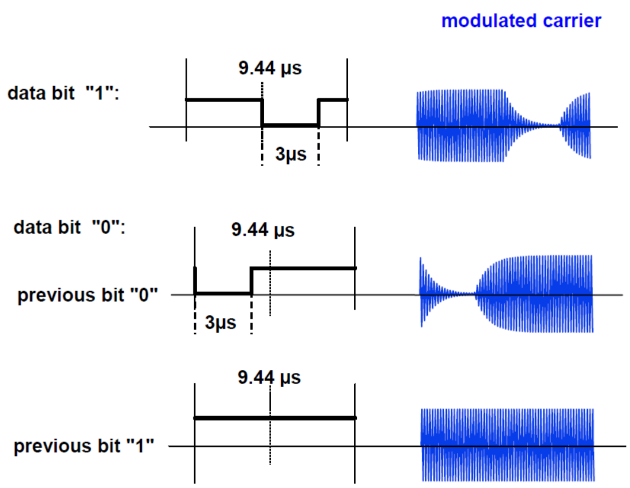

Figure 4. Realistic Miller Encoding [19]

- A 1 is encoded by modulating the subcarrier for the *first* half ($= t_b/2 \approx 4.72\mu s$) of the bit duration
- A 0 is encoded by modulating the subcarrier for the *second* half ($= t_b/2 \approx 4.72\mu s$) of the bit duration
- A logical 1 starts the transmission
- No modulation signifies the end of a transmission

3.3 The Protocol

Though the ISO/IEC 14443A protocol is general, we will focus on a few key aspects that are relevant to our discussion. As a result, we will refer the reader to [14] for more details such as timing requirements.

First of all, it is worth noting that each byte is **ordered** from the Least Significant Bit (LSB) to the Most Significant Bit (MSB), and that each byte is followed by an **odd parity bit**, meaning that an even number of high bits (ones) is followed by another high bit (one), whereas an odd number of ones is followed by a low bit (zero). For example, the byte 0x3F is encoded as 1111 1100 1.

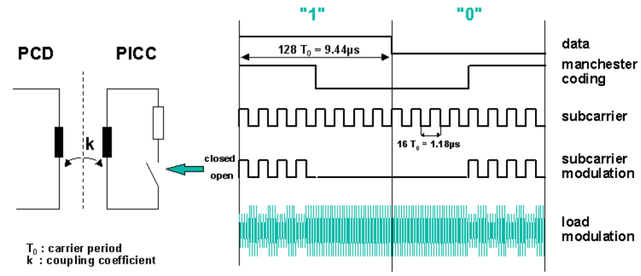

Figure 5. Manchester Encoding with Load Modulation [19]

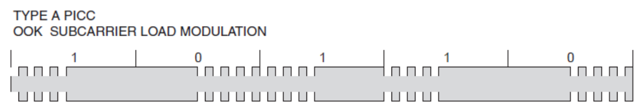

Figure 6. Envelope of Load Modulation [1]

The PCD signifies that it is waiting to read tags by repeatedly sending a **REQA (0x26)** or a **WUPA (0x52)**, where the "A" signifies that type A protocol is used. The difference between the REQA request and the WUPA wake-up request is that the latter also wakes up PICCs that were previously asked to HALT. Unlike all other commands, they are sent using a *short frame* that only consists of 7 bits, and which does not include a parity bit. As a result, the REQA command (including the beginning and end transmission zero bits) is sent as the sequence of bits 0 0110 010 0.

The standard has also defined **Anticollision** and **Selection** phases before the transmission of actual data which are used to ensure the non-interference from multiple tags and the correct selection of the tag. However, we only discuss them in the context of the MIFARE cards in Appendix A, since they are not necessary for understanding the rest of this report.

Finally, we mention that to detect errors with longer transmissions, for some requests and responses a **Cyclic Redundancy Check (CRC)** is used on the transmitted bytes (but excluding start/end and parity bits). The polynomial used is $x^{16} + x^{12} + x^5 + 1$, with a starting value of 0x6363, under the assumption that "FF0 shall be the leftmost flip-flop where data is shifted in [and] FF15 shall be the rightmost flip-flop where data is shifted out" [14]. For example, the **HALT/HLTA** command uses 2 bytes (0x50 0x00) followed by the two CRC bytes which can be calculated as 0x57 0xCD.

3.4 MIFARE Classic 1K Encryption

Even though the MIFARE Ultralight is ISO/IEC 14443 A compliant [21], the MIFARE Classic 1K uses a proprietary cryptographic protocol called CRYPTO1 [20]. The details of the protocol were not made publicly available, with the MIFARE datasheets only broadly explaining the 3-pass protocol [20]. Each **sector** (equal to 4 **blocks** of 16 bytes each) has two 6-byte keys (**Key A and B**), which on delivery are set to

[0xFF 0xFF 0xFF 0xFF 0xFF 0xFF] (but can be changed later on a per-sector basis). Each authentication happens with one of the two keys chosen by the reader, and can only be used to access a specific sector. The memory layout for a Classic 1K card is shown in Figure 7, and shows that each sector contains one block (the **sector trailer**) which contains the two keys and some **access bits** which determine the allowed operations for the 4 blocks (See Appendix A for more details).

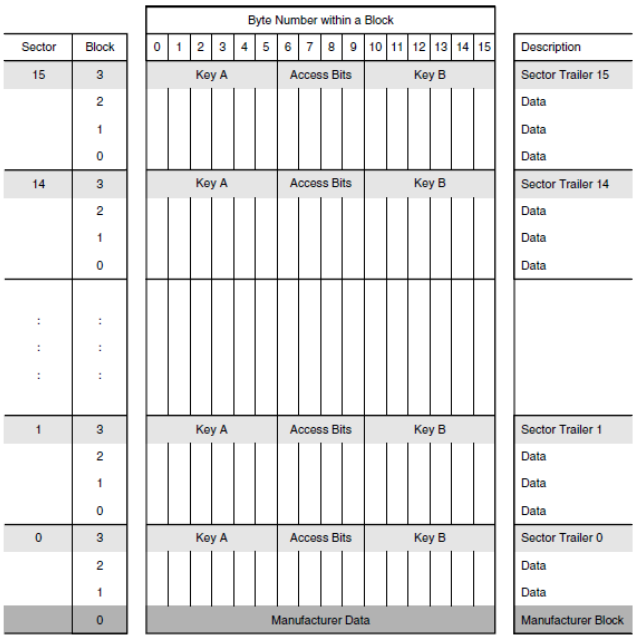

Figure 7. MIFARE Classic 1K Memory Layout [20]

The PCD indicates to the PICC that it wants to authenticate through a command indicating which key to be used and what address to use. As shown in Figures 8 and 9, the three-pass scheme consists of a 4 byte challenge sent from the PICC to the PCD (Token RB), an 8-byte challenge and response (of 4 bytes each) send from the PCD to the PICC (Token AB) and a response from the PICC to the PCD (Token BA), if the reader's encryption was correct. After the first challenge (Token RB), **all** traffic is encrypted, even subsequent authentications, which leads to a weakness known as **nested authentication** [8].

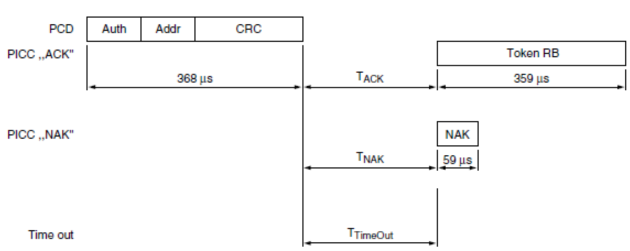

Figure 8. First Part of Authentication [20]

Though we discuss the encryption algorithm in greater detail in Appendix B, it is worth noting a few things based on the research in [18, 8]. The CRYPTO1 encryption scheme is a stream cipher which consists of a 48-bit (equal to the key length) Linear Feedback Shift Register (LFSR) and a non-linear **filter** function [18]. The encryption incorporates both the tag's Unique Identifier (UID) and the random nonce RB, which however is only generated using a 16-bit LFSR. Nonetheless, both challenge responses only depend on the tag's nonce (and not the reader's nonce or the UID), and use the same LFSR as the Random Number Generator (RNG).

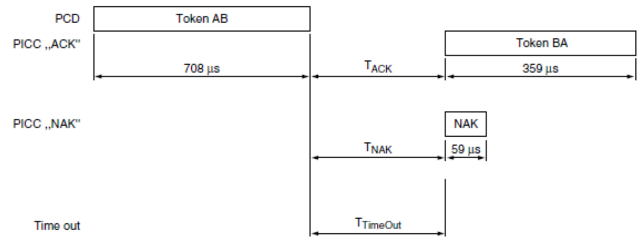

Figure 9. Second Part of Authentication [20]

What is more, the parity bits are also encrypted (making the MIFARE Classic 1K *incompatible* with the ISO 14443 protocol), and "the bit of keystream used to encypt the parity bits is reused to encrypt the next bit of plaintext" [8]. This vulnerability, in combination with the nested authentication mentioned above (which causes the token RB to also be encrypted) leaks data which can be used to guess the nonce or to reveal the secret key.

Acknowledgments

I would like to thank Kasper Rasmussen for being my supervisor, for his invaluable help throughout the project, and for entrusting me with his expensive equipment. I would also like to thank Simon Crowe for helping shape the focus of the project. Finally, I would like to thank Kellogg College and their Research Support Grant which enabled the purchase of some of the components that were used in this project.

References

[1] ATMEL. *Understanding the Requirements of ISO/IEC 14443 for Type B Proximity Contactless Identification Card*, 11 2005.

[2] BRIAND, A., ALBERT, B., AND GURJAO, E. Complete software defined RFID system using GNU radio. In *IEEE International Conference on RFID-Technologies and Applications (RFID-TA 2012)*.

[3] BUETTNER, M., AND WETHERALL, D. A software radio-based UHF RFID reader for PHY/MAC experimentation. In *IEEE International Conference on RFID (RFID 2011)*.

[4] BUETTNER, M., AND WETHERALL, D. A flexible software radio transceiver for UHF RFID experimentation. Tech. Rep. UW-CSE-09-10-02, University of Washington - Computer Science Department, 2009.

[5] BUETTNER, M., AND WETHERALL, D. A ”Gen 2” RFID monitor based on the USRP. *SIGCOMM Comput. Commun. Rev.* (2010).

[6] DE KONING GANS, G. Analysis of the MIFARE classic used in the OV-Chipkaart project. Master’s thesis, Radboud University Nijmegen, 2008.

[7] FRANCIS, L., HANCKE, G., MAYES, K., AND MARKANTONAKIS, K. *Practical Relay Attack on Contactless Transactions by Using NFC Mobile Phones*.

[8] GARCIA, F. D., ROSSUM, P. V., VERDULT, R., AND SCHREUR, R. W. Wirelessly pickpocketing a Mifare Classic card. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (SSP 2009)*.

[9] GUMMESON, J. J., PRIYANTHA, B., GANESAN, D., THRASHER, D., AND ZHANG, P. Engarde: Protecting the mobile phone from malicious nfc interactions. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2013)*.

[10] HASSANIEH, H., WANG, J., KATABI, D., AND KOHNO, T. Securing RFIDs by randomizing the modulation and channel. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2015)*.

[11] ISO. Identification cards—Contactless integrated circuit cards—Proximity cards—Part 1: Physical characteristics. ISO 144443-1:2008, International Organization for Standardization, Geneva, Switzerland, 2008.

[12] ISO. Identification cards—Contactless integrated circuit cards—Proximity cards—Part 4: Transmission protocol. ISO 144443-4:2008, International Organization for Standardization, Geneva, Switzerland, 2008.

[13] ISO. Identification cards—Contactless integrated circuit cards—Proximity cards—Part 2: Radio frequency power and signal interface. ISO 144443-2:2010, International Organization for Standardization, Geneva, Switzerland, 2010.

[14] ISO. Identification cards—Contactless integrated circuit cards—Proximity cards—Part 3: Initialization and anticollision. ISO 144443-3:2011, International Organization for Standardization, Geneva, Switzerland, 2011.

[15] LAURIE, A. RFIDler. `https://github.com/ApertureLabsLtd/RFIDler`. Accessed: 2015-06-19.

[16] MICHAEL, W. Performing relay attacks on ISO 14443 contactless smart cards using NFC mobile equipment. Master’s thesis, Der Technischen Universität München, 2010.

[17] MILLER, C. Exploring the NFC attack surface. Presented at Black Hat USA 2012.

[18] NOHL, K., EVANS, D., STARBUG, S., AND PLÖTZ, H. Reverse-engineering a cryptographic RFID tag. In *Proceedings of the 17th Conference on Security Symposium (USENIX Security 2008)*.

[19] NXP SEMICONDUCTORS. About MIFARE. `https://www.mifare.net/en/about-mifare/`. Accessed: 2015-06-19.

[20] NXP SEMICONDUCTORS. *MIFARE Classic 1K - Mainstream contactless smart card IC for fast and easy solution development*, 2 2011. Rev. 3.1.

[21] NXP SEMICONDUCTORS. *MIFARE Ultralight contactless single-ticket IC*, 7 2014. Rev. 3.9.

[22] PENRI-WILLIAMS, K. E. Implementing an RFID ’Mifare Classic’ attack. Master’s thesis, City University London, 2009.

[23] PLÖTZ, H. RFID hacking. Presented at the 23rd Chaos Communication Congress (CCC 2006).

[24] PLÖTZ, H. Mifare classic – eine analyse der implementierung. Master’s thesis, Humboldt-Universität zu Berlin, 2008.

[25] RICHTER, H., MOSTOWSKI, W., AND POLL, E. Fingerprinting passports. NLUUG spring conference on security (2008).

[26] RYAN, R., ANDERSON, Z., AND CHIESA, A. Anatomy of a subway hack. Presented at the 16th DEFCON Hacking Conference (DEFCON 2008).

[27] TAN, W. H. Practical attacks on the MIFARE Classic. Master’s thesis, Imperial College London, 2009.

[28] WESTHUES, J. Proxmark 3. `https://github.com/Proxmark/proxmark3`. Accessed: 2015-06-19.

[29] ZHENG, Y., AND LI, M. ZOE: Fast cardinality estimation for large-scale RFID systems. In *INFOCOM* (2013).

Appendices

A MIFARE Commands

B CRYPTO1

As mentioned in Section 2, researchers were able to reverse-engineer the protocol [18, 8], and we summarize their findings here. Specifically, in the notation of [8], let the (unencrypted) nonce RB be denoted by n_T, the token AB be denoted as $\{n_R\}, \{a_R\}$ and the token BA be denoted as $\{a_T\}$. Also denote by k the key, u the tag's unique identifier (UID), and for any x, let x_i be its i-th bit (for i for which this is well-defined). The CRYPTO1 algorithm uses a Linear Feedback Shift Register (LFSR) of size equal to 48 bits which is initialized by the key (also of length 48). Thus, as $\alpha_i = a_i a_{i+1} \dots a_{i+47}$ the internal state of the LFSR at time i, we get that:

- $a_i := k_i$, for $0 \le i \le 47$
- $a_{48+i} := L(a_i, \dots, a_{47+i}) \oplus n_{T,i} \oplus u_i$, for $0 \le i \le 31$
- $a_{80+i} := L(a_{32+i}, \dots, a_{79+i}) \oplus n_{R,i}$, for $0 \le i \le 31$
- $a_{112+i} := L(a_{64+i}, \dots, a_{111+i})$, $\forall i \in \mathbb{N}$

where L is the LFSR "**feedback function**" defined by:
$L(x_0 x_1 \dots x_{47}) := x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43}$

The outputs are encrypted using a "**filter function**":
$f(x_0 x_1 \dots x_{47}) := f_c(f_a(x_9, x_{11}, x_{13}, x_{15}), f_b(x_{17}, x_{19}, x_{21}, x_{23}), f_b(x_{25}, x_{27}, x_{29}, x_{31}), f_a(x_{33}, x_{35}, x_{37}, x_{39}), f_b(x_{41}, x_{43}, x_{45}, x_{47}))$, where

$$f_a(y_0, y_1, y_2, y_3) := ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))$$
$$f_b(y_0, y_1, y_2, y_3) := ((y_0 \wedge y_1) \vee y_2) \oplus ((y_o \oplus y_1) \wedge (y_2 \vee y_3))$$
$$f_c(y_0, y_1, y_2, y_3, y_4) := (y_0 \vee ((y_1 \vee y_4) \wedge (y_3 \oplus y_4))) \oplus ((y_0 \oplus (y_1 \wedge y_3)) \wedge ((y_2 \oplus y_3) \vee (y_1 \wedge y_4)))$$

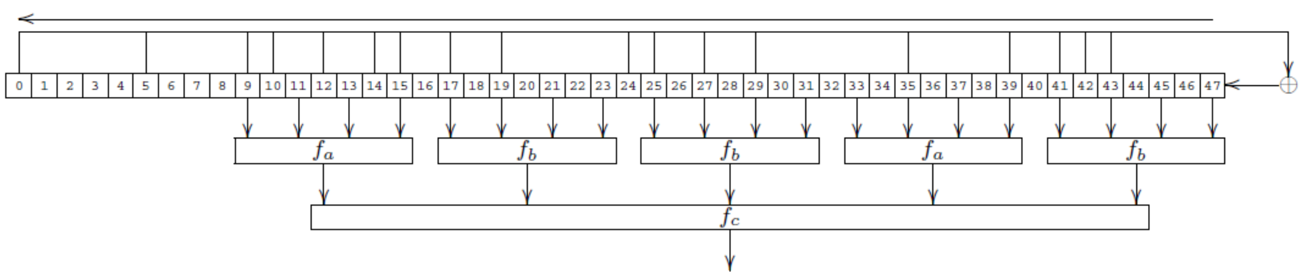

Figure 10. Crypto1 LFSR [8]

C Example Traces