國立臺灣大學電機資訊學院電機工程學系

碩士論文

Department of Electrical Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

MIFARE Classic 上的實務攻擊與防禦
Practical Attacks and Defenses of MIFARE Classic

池明洋

Ming-Yang Chih

指導教授：鄭振牟 博士

Advisor: Chen-Mou Cheng, Ph.D.

中華民國 99 年 6 月

June, 2010

誌謝

在台大兩年的碩士生活首要感謝我的兩位指導教授 鄭振牟 博士與 楊柏因博士：感謝鄭老師兩年來耐心、細心的指導我這塊朽木，在研究的路上提供我持續不斷的支援以及協助，教導我做人處事的道理；感謝楊老師總是能在我士氣低落時，帶給我歡笑，使我能再次挑戰困難的問題。兩位老師提供我一個很棒的研究環境，讓我在研究上、生活上皆無憂慮。感謝兩位年輕又優秀的學者兩年的研究指導，僅在此至上最誠摯的謝意與敬意。

其次感謝 631 全體的同學：周彤、麋鹿、CS、蕭老、大明星、歐文，他們給我許多研究上的建議以及指導。不論什麼時候有什麼問題，大家都相當樂意幫我解決。特別是歐文總在我需要的時候，無私的幫我解決很多研究上的問題。感謝以上的戰友們及學弟們一次次配合與參與，我所安排的實驗室活動，希望各位能有研究以外共同的回憶。

再者感謝台大棒球隊兩年來的照顧，感謝多位已畢業的學長分享許多職場以及過往的趣事。感謝現役的學弟在每次練習以及比賽的幫助。有你們每一分的努力才讓我們這兩年的大專決賽能有這樣好的表現，感謝你們給我這麼好的回憶，謝謝。

最後要感謝我的女友不間斷的關心，督促著我的學習。謹以此論文獻給每一位愛我及我愛的人。

池明洋

2010/6/26

於台大電機所 631 快速密碼實驗室

摘要

MIFARE Classic 是近年來最廣泛被使用的非接觸式智慧卡，應用在門禁、大眾運輸工具、電子錢包等系統上。MIFARE Classic 上密碼保護機制與結構已被發表在許多的論文上。在本論文中我們提出各式各樣在 MIFARE Classic 攻擊實作的經驗。我們實作兩類的攻擊:一是假造讀卡機、二是側錄合法的交易。第一類的攻擊在兩天內利用 NVIDIA 高速運算顯示卡上實作密鑰的窮舉搜尋法與隨機數和連認證的漏洞離線的破解卡片上所有的金鑰。第二類是針對 MIFARE Classic 加解密器: CRYPTO-1 上攻擊方法的改進。經過我們的改進，攻擊者不僅可以破解自己的卡同時也能破解別人的卡。我們所實作的攻擊徹底讓 MIFARE Classic 的密碼保護失去效用，讓未經授權的攻擊者能任意更改卡片上資料，如同沒有任何保護的記憶卡。更進一步，我們提出有關防止目前已知的攻擊的建議，而此防禦機制加強對卡片資料的防護並加強後端清算機制的效率。

關鍵字: MIFARE Classic, CRYPTO-1, cryptanalysis, GPU, RFID security.

Abstract

MIFARE Classic is a proprietary contactless smart card technology widely used in public transportation ticketing systems of cities across the world. MIFARE Classic's cryptographic protection to the stored data has been reverse-engineered and broken in a recent series of papers. In this thesis, we report our experiment experiences attacking a real MIFARE Classic system. Specifically, we implement a brute-force search using NVIDIA graphics cards to verify the claims in the literature. We also implement and improve more advanced attacks that take advantage of other design and implementation flaws of CRYPTO-1, MIFARE Classic's proprietary cipher. These attacks disarm all cryptographic protection of MIFARE Classic and in effect render it a contactless memory card technology. Last but not least, we present our ideas how to defend against most attacks using practical mechanisms that do not require any hardware changes. Our proposed mechanisms can be easily implemented on a variety of MIFARE Classic readers on the market and only require commodity PCs be used in the backend system with intermittent network connectivity.

Keywords: MIFARE Classic, CRYPTO-1, cryptanalysis, GPU, RFID security.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Developed and sold by NXP Semiconductors, Inc., MIFARE Classic is a proprietary contactless smart card technology widely used in public transportation ticketing systems of cities across the world, the most famous examples being Beijing's OnePass, Boston's CharlieCard, London's Oyster Card, and Taipei's EasyCard. As MIFARE Classic gets more and more popular, it is also being adopted in several cities as a means for implementing micro payment systems.

MIFARE Classic provides cryptographic protection to the stored data. Although the design and implementation of the cipher in MIFARE Classic are kept secret by NXP Semiconductors, the detail has been recovered and disclosed in a recent series of papers [1, 2, 3, 4, 5, 6].

The core cryptographic primitive used by MIFARE Classic is a proprietary stream cipher called CRYPTO-1. It uses a 48-bit linear feedback shift register (LFSR) as state. In each clock cycle, a fixed subset of the LFSR bits are fed into a non-linear filter function to generate one bit in the output keystream. For more details of the cipher, the interested reader is referred to Nohl et al. [3]

The most fatal design flaw of CRYPTO-1 is that the state space of 2^{48} is too small. For example, Garcia et al. estimated [5] that, given enough keystream bits or information about them, a brute-force search using an FPGA cluster like COPACOBANA [7] takes about 36 minutes. Furthermore, CRYPTO-1 has several other design and implementation flaws that allow even more efficient cryptanalyses.

In this thesis, we report our experience attacking a real MIFARE Classic system, as well as present our ideas how to defend most attacks in practice. Specifically, we implement a brute-force search using NVIDIA Graphics Processing Units (GPUs) to verify the aforementioned claim made by Garcia et al. Moreover, we also propose and implement a new long-range attack based on existing sniffing attacks that take advantage of CRYPTO-1's other flaws. These attacks disarm all cryptographic protection of MIFARE Classic and in effect render it a contactless memory card technology. Lastly, we propose a set of practical mechanisms that can defend against most attacks without requiring any hardware changes.

1.1 Background

In jargon of the trade in the field of radio-frequency identification (RFID), an RFID tag is called a Proximity Integrated Circuit Card (PICC), whereas an RFID reader is called a Proximity Coupling Device (PCD) [8]. We will follow this terminology for the rest of this thesis.

There are several kinds of MIFARE Classic PICCs with different capacities. Take MIFARE Classic 1K as an example. The one-kilobyte memory is divided into 16 sectors, each with four 16-byte blocks. Each sector has a tail block, which has two 48-bit secret keys, called Key A and Key B, and a 32-bit condition for the sector's access control.

The radio-frequency communication of MIFARE Classic consists of three phases: anti-collision, three-way authentication, and memory operation. The anti-collision follows the ISO-14443 standard [9], in which PCD selects a PICC by its unique identifier (UID). Anti-collision marks the beginning of a transaction, after which PCD can initiate a three-way authentication session if it wishes to access blocks in protected sectors. After the PICC receives an authentication command, it will first send out a tag challenge nonce N_t. This nonce, along with the PICC's UID, is used to initialize the CRYPTO-1 cipher so that all subsequent messages exchanged can be encrypted. The PCD then sends out the reader response A_r based on N_t and a reader challenge nonce N_r. The PICC checks whether A_r is as expected, and if so, it will send a tag response A_t, which is a function of

N_t (not N_r!), to the PCD. The authentication is concluded with PCD verifying whether A_t is as expected.

After the three-way authentication is successful, all subsequent communication is encrypted, including memory operations within the same sector as well as authentication requests for blocks in other sectors. Using the detailed command set recovered by Gans et al. [4], the attacker can easily launch a known-plaintext attack.

1.2 Motivation

Today in Taipei city, we can buy groceries and various merchandizes in convenient stores, ride buses and subways, and shop in department stores with EasyCard, which adopts the MIFARE Classic PICCs as the payment tool. Similar MIFARE-Classic-based systems are also deployed in Guangzhou, London, Boston, the Netherlands, . . . , etc. Moreover, in some systems MIFARE Classic PICCs are also used as e-cash.

MIFARE Classic is a kind of RFID technology with cryptographic protection Its cipher structure was regraded as confidential information by the companies who developed it, and hence the details had not been known to the general public. However, security can not be achieved through obscurity, and many security vulnerabilities of MIFRAE Classic have been discovered and disclosed in the past few years. Nevertheless, there has not been much study on the security of the affected systems, and the security needs to be examined as soon as possible by trusted third parities.

The purpose of this thesis is to investigate the security of MIFARE-Classic-based systems after a thorough study of the related security vulnerabilities and issues of MIFARE Classic. We aim to act as an independent security examiner, the so-called "white hats" who attack systems in good faith, give cost estimate of these attacks, and suggest ways to improve the systems' security.

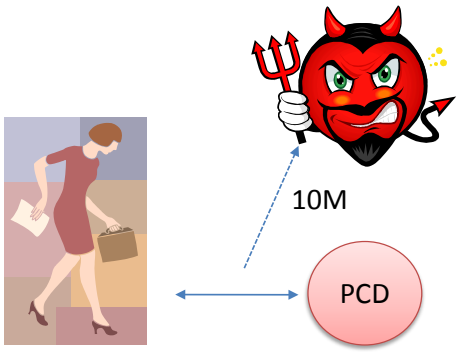

Figure 1.1: Sniffer-based Attack

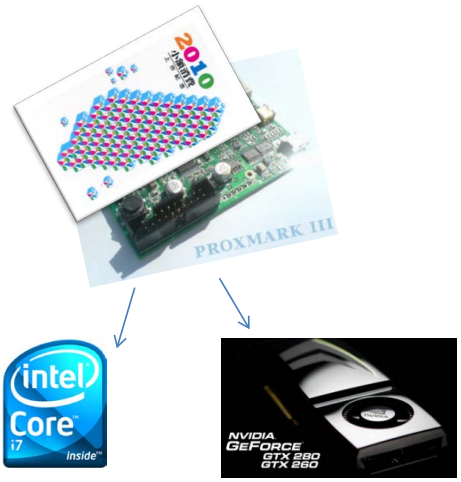

Figure 1.2: PCD-based Attack

1.3 Problem Statement

There have been many different attacks, including PCD-based and sniffer-based attacks, published during recent years. Garcia et al. provided two kinds of PCD-based attacks using FPGA or CPU. They proposed to launch a brute-force search with information obtained via the "encrypted error-code" vulnerability to obtain the first key from a MIFARE Classic PICC. But they only had an estimation based on the cryptanalysis of DES using COPACOBANA, a specialized FPGA cluster for cryptanalysis [7]. Unfortunately we do not have access to such an expensive FPGA cluster, so we turn our attention to cheaper, off-the-shelf GPUs. We implement a brute-force attack on GPU platform and conclude that the attacker can easily obtain secret keys with PICC-only attack for their own PICC.

For previous sniffer-based attacks, they have to get the full traffic between PICC to PCD. However, since PICC uses load modulation, eavesdropping packets sent from PICC is difficult at distance. The antenna of the sniffer needs to be placed in very close proximity, so the two-way sniffer-based attack is not useful in practice. Hence, we would like to improve and devise a more practical attack with long eavesdropping distance. The long-distance attack can assist the attacker to learn the keys of innocent users' MIFARE Classic PICCs.

1.4 Contribution

Our first contribution is to have implemented the PCD-based attack by using the cheaper GPU platform. Today's graphics cards contain powerful GPUs to handle the increasing number of screen pixels in video games. Now GPUs have developed into a powerful, massively parallel computing platform that finds more and more interest outside graphics applications. In cryptography, there have been many attempts of exploiting the computational power of GPUs [10, 11, 12]. Furthermore, there has been evidence that GPUs can be a quite competitive computing platform in terms of price-performance ratio [13, 14]. From a practical perspective, our implementation is more available than the others'. For example, in the Kuanghua market of Taipei, one can easily find that GPUs are fairly cheap and much more affordable than dedicated FPGA cryptanalysis machines like COPACOBANA. So our PCD-based attack is much more available. By using the PCD-based attack, it is easy to alter the content of the PICCs one already has access to. All one needs to do is to put it on a cracking device and wait for two days before one can recover all keys on the MIFARE Classic PICC.

Our second contribution is that we have improved the impractical two-way sniffer-based attack to a more practical long-range attack. We use GNU Radio, an open-source software radio platform [15] to implement the long-range sniffer. Specifically, we design and implement a demodulator on GNU Radio that allows us to eavesdrop the data transmission. Our sniffer can certainly pick up the data exchanged in both direction when the antenna is close to the PICC. If the antenna is far from PICC but not too far from PCD, then our sniffer can pick up the transmission from PCD. To recover key from such one-way traces, we design and implement a new kind of known-plaintext attack.

Finally, we not only attack the MIFARE Classic but also give suggestions how to defend against these attacks for system administrator. The defense methods employ public-key signatures on PCD. We use a fast signature scheme so that we need only firmware modification on the PCD, without any hardware change. We hope our ideas can help the affected systems as an interim solution before MIFARE Classic is completely phased out and replaced in these systems.

1.5 Thesis Outline

The rest of this thesis is organized as follows. In the beginning, we will introduce the recent research results about MIFARE Classic in Chapter 2. In Chapter 2, we will describe the structure of the CRYPTO-1 stream cipher, the communication protocol, memory structure, command set, and other security issues of MIFARE Classic.

Next, in Chapter 3, we will delineate our experiment environment. Our sniffer is based on the GNU Radio software radio platform and the USRP hardware, while we use Proxmark III to emulate PCD. We also use OpenPCD, an off-the-shelf reader with NXP's MIFARE Classic chip, to interact with PICCs once we recovered their secret keys.

In Chapter 4 and 5, we will describe the attacks and the defense for the MIFARE Classic system. Finally, we will conclude this thesis by summarizing what we have learned and providing suggestions for operators to alleviate the damage brought by the MIFARE Classic incident.

Chapter 2

MIFARE Classic

MIFARE is a product developed by the NXP Semiconductors, Inc., a subsidiary company of the Royal Philips Semiconductors, Inc., of the Netherlands. It is the most widely deployed contactless smartcard technology, with over one billion chips sold, as claimed by the producer. MIFARE Classic provides cryptographic protections such as data confidentiality. It is mainly achieved via mutual authentication between PCD and PICC using a proprietary stream cipher called CRYPTO-1. It also has a few proprietary commands for various memory operations. The command set, memory structure, and communication have been delineated by Gans et al. [4] They could also read a protected block without knowing the secret key. Subsequently, the structure of CRYPTO-1 has been reverse-engineered by Nohl et al. [3, 1] People have attacked the pseudo random number generator, the non-linear filter function, and other structures of the cipher. All the above works concentrate on recovering keys from eavesdropping traces. Then, Garcia et al. investigated more serious vulnerabilities [5] and found the vulnerabilities of the encrypted parity, the encrypted error code, and nested authentication. The following is mostly a recapitulation of their works to make this thesis self-content.

The cipher used by MIFARE Classic is called "CRYPTO-1," which is a stream cipher with a 48-bit LFSR (linear feedback shift register) and a two-level non-filter function. The structure of the cipher was developed and protected as a trade secret by NXP, but the circuit of the cipher was reverse-engineered.

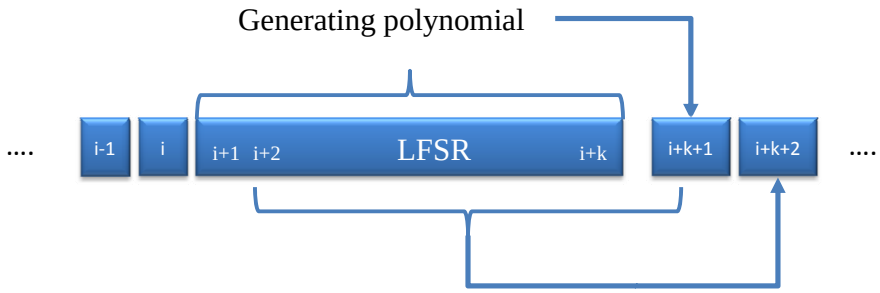

Figure 2.1: An Example of an LFSR

2.1 Linear Feedback Shift Register (LFSR)

LFSR is usually used as a pseudo random number generator (PRNG). Both the cipher and the PRNG in MIFARE Classic are LFSR-based. In general, LFSR consists of a fixed-sized registers initialized by some seed, which often consists of secret key, random number, UID, or a combination of some or all of them. As shown in Figure 2.1, the LFSR state shifts left (or right) a bit and is updated with a bit that is the bitwise XOR of some selected bits upon each clock cycle. The selected bits are taken from some fixed positions that are indicated by a generating polynomial, usually irreducible. Consequently, the same seeds are mapped to the same sequence of numbers, and the total number of states is restricted by the order of the generating polynomial.

2.2 Structure of CRYPTO-1

As shown in Figure 2.2, CRYPTO-1 is composed of a 48-bit LFSR with following generating polynomial

$$x^{48}+x^{43}+x^{39}+x^{38}+x^{36}+x^{34}+x^{33}+x^{31}+x^{29}+x^{24}+x^{23}+x^{21}+x^{19}+x^{13}+x^{9}+x^{7}+x^{6}+x^{5}+1,$$

as well as a non-linear two-level filter function, which will be described in Section 2.2.1. The generating polynomial is interpreted as representing bit positions. Normally, x^{48} refers to the most significant bit, and the constant term, the least significant bit. In this thesis, we call the register state at a specific time as an "internal state." A clock cycle

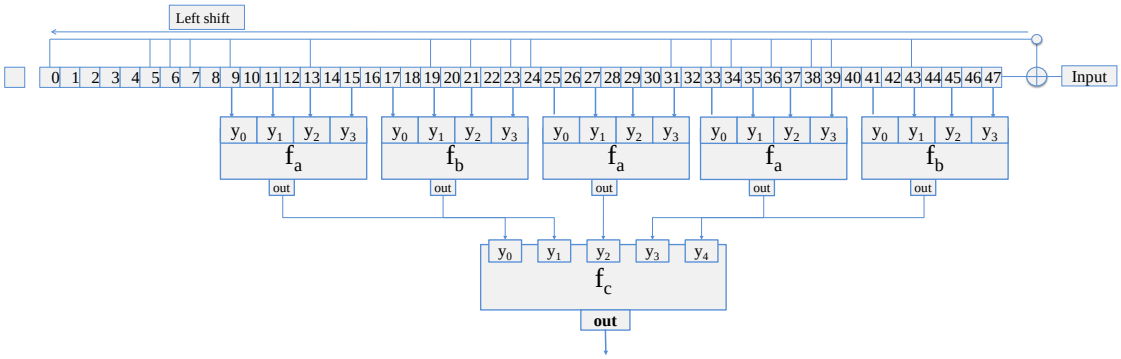

Figure 2.2: The Structure of the CRYPTO-1 Cipher

in CRYPTO-1 is 9.44 μs. The most significant bit is number 47, which is the bit being updated, as in Figure 2.2. In other words, the bits in any internal state satisfy the equation

$$x^{48}+x^{43}+x^{39}+x^{38}+x^{36}+x^{34}+x^{33}+x^{31}+x^{29}+x^{24}+x^{23}+x^{21}+x^{19}+x^{13}+x^{9}+x^{7}+x^{6}+x^{5}+1=0,$$

and we can move x^{48} to the right side of the equation:

$$x^{43}+x^{39}+x^{38}+x^{36}+x^{34}+x^{33}+x^{31}+x^{29}+x^{24}+x^{23}+x^{21}+x^{19}+x^{13}+x^{9}+x^{7}+x^{6}+x^{5}+1=x^{48}.$$

To summarize, the cipher calculates x^{48} by the above equation and shifts the bits appropriately to generate the next state, which can be illustrated with the following example: if the 48-bit internal state is 0x0123456789AB in big-endian notation, then the next state will become 0x0091A2B3C4D5 after one clock cycle.

2.2.1 Non-Linear Filter Function

The non-linear filter function is a two-level filter function consisting of f_a, f_b, f_c:

$f_a(y_0, y_1, y_2, y_3) = ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))$

$f_b(y_0, y_1, y_2, y_3) = ((y_0 \wedge y_1) \vee y_2) \oplus ((y_0 \oplus y_1) \wedge (y_2 \vee y_3))$

$f_c(y_0, y_1, y_2, y_3, y_4) = ((y_0 \vee (y_1 \vee y_4) \wedge (y_3 \oplus y_4)) \oplus ((y_0 \oplus (y_1 \wedge y_3)) \wedge (y_2 \oplus y_3) \vee (y_1 \wedge y_4)))$

Filter Function

The filter function $f : F_2^{48} \rightarrow F_2$ is defined by

$f(x_0x_1...x_{47}) = f_c(f_a(x_9, x_{11}, x_{13}, x_{15}),$

$f_b(x_{17}, x_{19}, x_{21}, x_{23}), f_b(x_{25}, x_{27}, x_{29}, x_{31}),$

$f_a(x_{33}, x_{35}, x_{37}, x_{39}), f_b(x_{41}, x_{43}, x_{45}, x_{47})).$

We note that total number of all possible inputs is 2^{20}. It appears that the output distribution of f is uniform in the sense that the function f outputs roughly the same number of zeros and ones.

2.2.2 Keystream Generation

The CRYPTO-1 stream cipher generates one bit of keystream, or one pseudo random number bit, as follows. The beginning of the operation is to select 20 bits from the odd positions between bit 9 to bit 47, or equivalently according to the polynomial

$$x^{47}+x^{45}+x^{43}+x^{41}+x^{39}+x^{37}+x^{35}+x^{33}+x^{31}+x^{29}+x^{27}+x^{25}+x^{23}+x^{21}+x^{19}+x^{17}+x^{15}+x^{13}+x^{11}+x^{9},$$

as the 20-bit input to the non-linear filter function to calculate one bit of keystream. Secondly, the cipher computes a updating bit by taking the fixed bits in internal register. Thirdly, the internal state is shifted such that the least significant bit is thrown away while the most significant bit updated. In order to disturb the internal state, the cipher takes more seeds including $UID \oplus N_t$ and N_r into the internal state during initialization. These extra seeds are XOR'ed with the updating bit before the internal state is updated.

We note that the 48-bit internal state of the cipher can produce ten consecutive keystream bits without updating LFSR. This is because of the following. Assuming the internal state at a specific time t allows us to compute ten keystream bits at the time from $t - 9$ to t. More clearly, we have

$$ks_{t-i} = x^{47-i} + ... + x^{9-i}, \forall i = 9 \text{ to } 0.$$

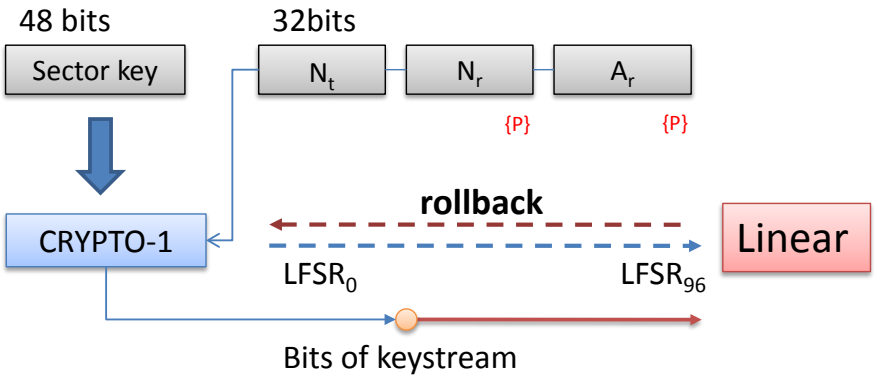

Figure 2.3: The Sequence of Seeding the CRYPTO-1 Cipher

This method will form a basis for some attacks later in this thesis.

2.3 Memory Structure

MIFARE Classic PICC is a memory card with two different memory sizes, 1-kilobyte and 4-kilobyte. The memory is organized into sectors, each sector with four blocks, while each block consists of sixteen bytes. In each sector, every last data block is called a sector trailer, which is used to hold two sector keys, Key A and Key B, and a 3-byte access condition. A block is the minimum unit of memory, which can be a data block, a value block, or a manufacturer block. A data block can store 16 bytes of arbitrary data. The same block can also be configured as a value block, in which case it can record a four-byte value. The value in a value block is stored twice non-inverted and once inverted for data integrity consideration; see block 60 (or 61) of the memory structure example in Figure 2.4. sector [16, 17]. For a MIFARE Classic 1K PICC, the memory is divided into 16 sectors, as shown in Figure 2.4. For a MAFIRE Classic 4K PICC, it comprises 14 sectors with two different numbers of blocks, 4 and 16. Each of the first 32 sectors has four blocks, while the rest sectors have 16 blocks.

For both kinds of PICCs, the access condition controls access rights for each secret key (Key A or Key B) in this

Sector number	Block number	Content（16 Bytes）						
0	0	UID, BCC, Manufacturer (Read Only)						
	1. Data/ Value	Data or Value						
	2. Data/ Value	Data or Value						
	3.Control	Key A	Access cond.	U	Key B			
1	4. Data/ Value	Data or Value						
	5. Data/ Value	Data or Value						
	6. Data/ Value	Data or Value						
	7.Control	Key A	Access cond.	U	Key B			
⋮								
15	60.Data/ Value	*Value*	$\overline{Value}$	*Value*	*A*	$\overline{A}$	*A*	$\overline{A}$
	61. Data/ Value	*Value*	$\overline{Value}$	*Value*	*A*	$\overline{A}$	*A*	$\overline{A}$
	62. Data/ Value	Data or Value						
	63.Control	Key A	Access cond.	U	Key B			

Figure 2.4: Memory Structure of MIFARE Classic 1K.

2.4 Command Set

A MIFARE Classic command is composed of a one-byte command, a one-byte block number, and a two-byte cyclic redundant check (CRC) code. In MIFARE classic protocol, there are seven different commands, namely, READ, WRITE, AUTHENTICATE, INCREMENT, DECREMENT, RESTORE, and HALT, as shown in Table 2.1.

1. READ:

   The command code is $0x30$. To read a block. PICC will send back 16 bytes of block data and two bytes of CRC.

2. WRITE:

   The command code is 0xA0. PCD writes 16 bytes of data to overwrite the entire data block in PICC. After the WRITE command, PICC returns a four-bit acknowledgment (ACK) to PCD. Then, PCD transmits 16 bytes of data to PICC. PICC optionally decrypts the data and writes to memory. If everything is correct, PICC sends to PCD an ACK again.

Operation	Command	Description
AUTHA	0x60 WW YY ZZ	Use keyA to authenticate block 0xWW
AUTHB	0x61 WW YY ZZ	Use keyB to authenticate block 0xWW
READ	0x30 WW YY ZZ	Read data from block 0xWW
WRITE	0xA0 WW YY ZZ	Write data to block 0xWW
DECREMENT	0xC0 WW YY ZZ	Decrement value in block 0xWW
INCREMENT	0xC1 WW YY ZZ	Increment value in block 0xWW
RESTORE	0xC2 WW YY ZZ	Restore previous value in block 0xWW
HALT	0x50 00 57 CD	Fixed command
Note	WW is block number	Two bytes CRC = 0xYYZZ

Table 2.1: MIFARE Classic Command Code Table

3. Value commands:

   The command codes for DECREMENT, INCREMENT, and RESTORE are 0xC0, 0xC1, and 0xC2, respectively. First, PCD sends a command, and PICC gives ACK. Second, PCD sends a four-byte value followed by a two-byte CRC, along with "TRANSFER" to PICC. PICC verifies the value with CRC and sends ACK again. There are three value commands, INCREMENT, DECREMENT, and RESTORE. "INCREMENT" is used to increment the value block by the value, "DECREMENT" is used to decrement the value block by the value, and "RESTORE" is used to restore the value block to previous value.

4. AUTHENTICATE:

   Command codes are 0x60 for Key A and 0x61 for Key B. This command is for PCD to authenticate with PICC using the shared secrete keys. The communication protocol is shown in Figure 2.7 and described in Section 2.5.

5. HALT:

   The command code is 0x50 and always takes zero block, so the complete command is 0X500057CD.

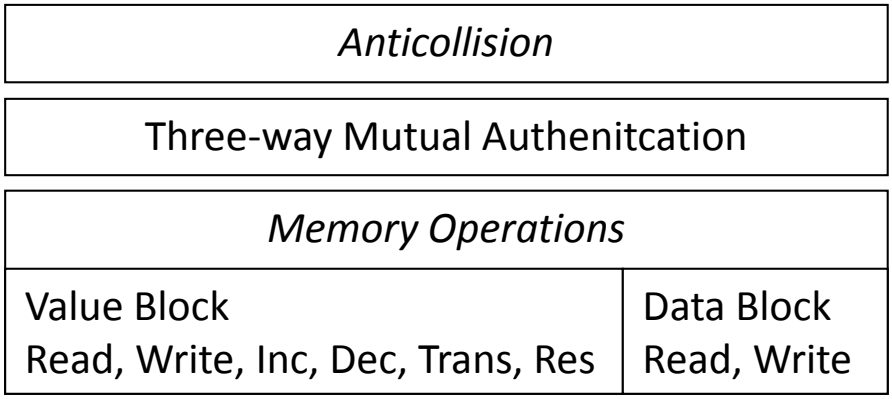

Figure 2.5: Communication Protocol Overview

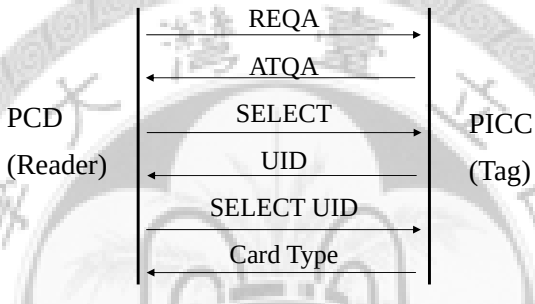

Figure 2.6: Anticollision for ISO-14443 Type-A

2.5 Communication Protocol

The communication protocol of MIFARE Classic is described in the ISO 14443 standard, type-A. Figure 2.5 gives an overview of the communication protocol that consists of three stages, namely, anti-collision, three-way authentication, and memory operation.

First of all, anti-collision, shown in Figure 2.6, selects the PICC with a unique identifier (UID) around the electromagnetic field of the PCD. In a transaction, anti-collision

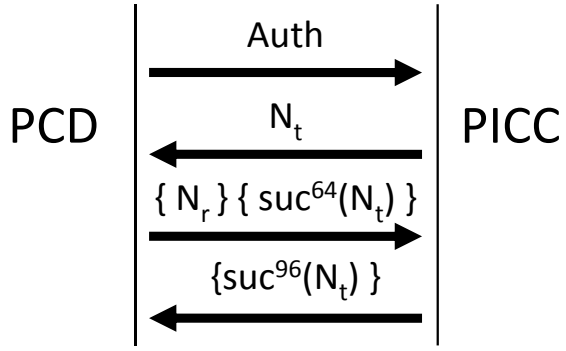

Figure 2.7: Three-way Mutual Authentication

is seen as the initialization between PCD and PICC and usually executes once. When the wireless connection becomes unstable, PCD will execute another anti-collision again to make sure that PICC is still available within its electromagnetic field. On real systems, if a PCD is going to access many blocks, the total transaction time may become longer. In this case, the PCD will divide memory operations into two or three groups and execute an anti-collision at the beginning of each group.

Secondly, PCD will choose a block number and a sector key (A or B) to authenticate with PICC. After PICC receives an AUTHENTICATE command, it will respond with a tag challenge nonce N_t and wait for the PCD to respond. As PCD sends A_r ($suc^{64}(Nt)$), computed based on N_t, and picks up a reader challenge nonce N_r, PICC inspects whether A_r is correct or not. If it is correct, PICC will send a tag response A_t ($suc^{96}(Nt)$) to PCD. PCD also checks A_t against A_r to complete the mutual authentication. For internal state of the cipher, N_t and N_r would be used to seed into the internal state in order to disturb the internal state. This mechanism makes the keystream bits different from transaction to transaction.

Finally, the reader can operate accessible blocks in this sector dictated by the access condition at the sector tail.

2.6 Other Components in MIFARE Classic

2.6.1 Pseudo Random Number Generator (PRNG)

A PRNG generates a sequence of numbers that looks random. MIFARE Classic uses a PRNG based on a 32-bit LFSR with the generating polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$. In the actual MIFARE Classic PRNG implementation, it uses with only one seed. For example, assume we have a 32-bit random number $b_{31}, b_{30}, ..., b_0$. We use little-endian notation for integers here so that 0x12345678 is $b_{24} \dots b_{31}, b_{16} \dots b_{23}, b_8 \dots b_{15}, b_0 \dots b_7$. The PRNG takes $b_{32} = b_{21} \oplus b_{19} \oplus b_{18} \oplus b_{16}$, and then shifts right to throw away b_0 while puts b_{32} into the highest bit.

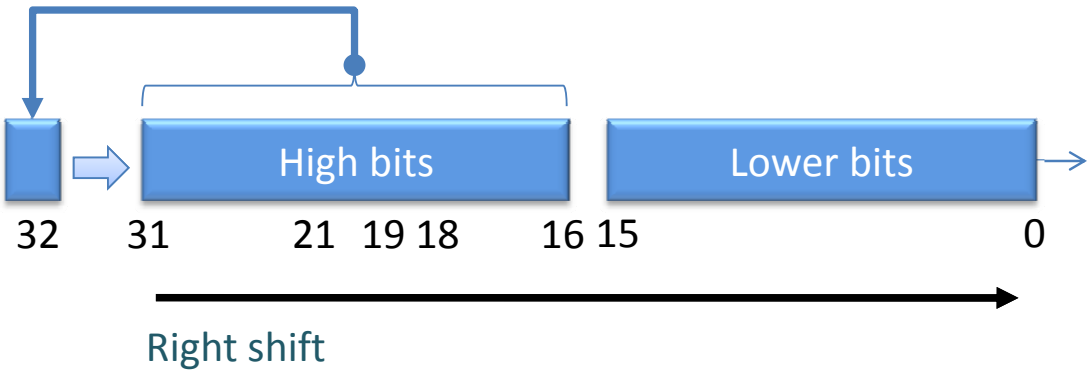

Figure 2.8: Pseudo Random Number Generator

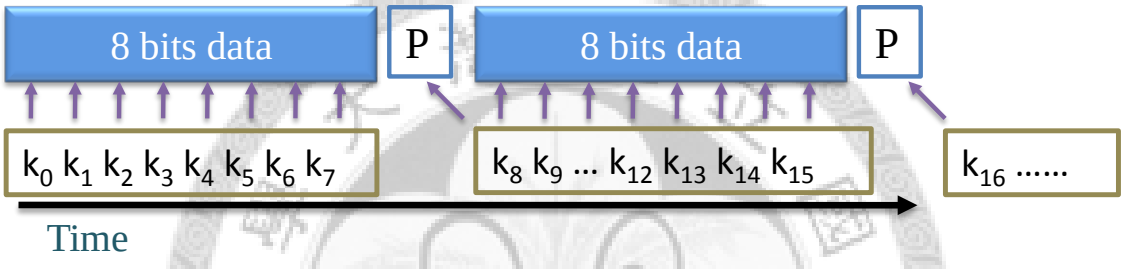

Figure 2.9: The Encrypted Parity

2.6.2 The Encrypted Parity

Use of parity is a simple method to check the data integrity. The MIFARE Classic, following ISO 14443, uses a parity bit for every byte transmitted. In the implementation, the parity bits are computed before encryption and then encrypted using the next keystream bit, as shown in Figure 2.9. Assuming that the plaintext comprises 32 bits $t_1 ... t_{32}$ and the keystream bits $(r_1 ... r_{33})$, the parity bits can be computed by the following equation:

$$p_i = t_{i+1} \oplus t_{i+2} \oplus ... \oplus t_{i+8} \oplus r_{i+9} \oplus 1, \forall i \in \{0, 1, 2, 3\}.$$

2.6.3 Encrypted Error Code 0x5

In a successful three-way mutual authentication, PCD authenticates PICC with shared secret key. As shown in Figure 2.10, when PICC receives a response $\{N_r\}\{A_r\}$, it decrypts the data to plaintext with its secret key, and the plaintext is checked against two condi-

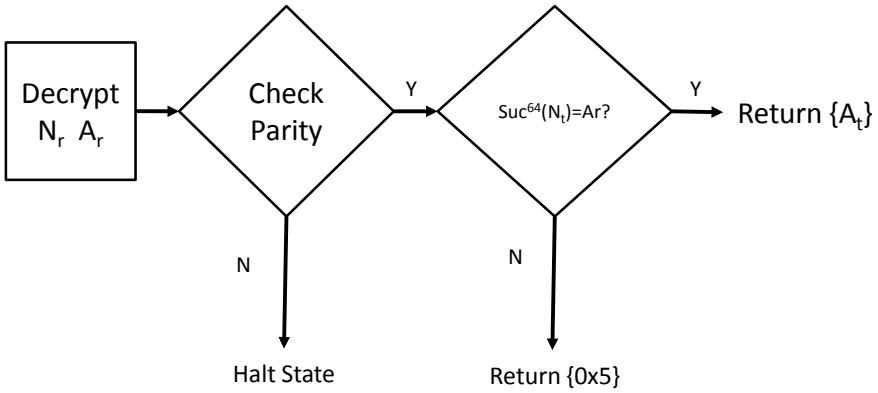

Figure 2.10: Flow of Reader Response Checking

Table 2.2: An Example Error-code Trace

Seq.	Sender	Code	Explanation
1 - 6	⋮	Anticollision	Exchange UID
7	PCD	0x6000f57b	Authenticate block 0
8	PICC	0xf9105fce	N_t
9	PCD	$\{00000000_0\}$ $\{00000000_0\}$	Some random numbers
10	PICC	$\{5\}$	Error code 0x5

tions: (1) data parity, and (2) decrypted reader response A_r. If the data parity is correct but the reader response is not, the PICC will return an encrypted error code 0x5. If both conditions are satisfied, the PICC will send A_t back. Otherwise, PICC will terminate the session without sending any packets.

Assuming that the keystream bits are more or less random from transaction to transaction, the parities of $\{N_r\}\{A_r\}$ will be correct with a probability of 1/256, in which case we will be able to collect an error-code trace as shown in Table 2.2. An error-code trace like this thus gives 12-bit information, four from the known plaintext error-code 0x5 and eight from the fact that the parities are correct, on a keystream that depends on the secret key used for authentication by the PICC.

Chapter 3

Experiment Setup

3.1 Sniffer

The communication of MIFARE Classic follows ISO-14443 Type-A standard, in which modified Miller code and Manchester code are used. We use the "Universal Software Radio Peripheral (USRP)" to capture and digitize the signal. USRP converts the intercepted analog signal to digital form and transfers the raw samples across USB for further processing on PC. If a transaction happens between PCD and PICC, an appropriate antenna is around the magnetic field within a certain distance can capture the signal. The signal is then sent to the RF module on USRP, which converts to digital form and delivers the resulting samples to PC. We develop our own signal processing software on the GNU Radio open-source software radio platform, which allows programmers to create their own signal processing blocks. In the following, we will describe our development environment, hardware device, and the sniffer implementation.

3.1.1 Universal Software Radio Peripheral (USRP)

USRP is designed to allow general-purpose computers to function as high-bandwidth software radios. Typically it is responsible for digital baseband and IF (intermediate frequency) sections of a radio communication system. The basic design philosophy behind USRP is to do all of the waveform-specific processing, like modulation and demodula-

tion, on the host CPU [18]. All of the high-speed general-purpose operations, like digital up- and down-conversion, decimation, and interpolation, are done on USRP's FPGA.

Basic RF Module on USRP

USRP's BasicRX is designed for use with external RF frontends as an IF interface. The ADC input is directly transformer-coupled to an SMA connector of 50-ohm impedance with no mixers, filters, or amplifiers. Our sniffer implementation takes the USRP and its basic RF module as the front-end digital and analog processor. In the following, we regard both devices as a DSP board.

3.1.2 GNU Radio

GNU Radio is a free software development toolkit that handles signal process routine by the software [15]. The simple the procedure on our sniffer can be considered as two parts: (1) raw data collection, and (2) data processing routine. These two parts form a loop that executes continuously. In the first part, raw data are continuously collected with some basic signal processing handled by the general-purpose DSP board with a set of parameters set by the host. The collected data are transferred via USB to the host. In the second and more important part, we use GNU Radio to create the signal processing block to demodulate the raw data received from USB.

3.1.3 Sniffer Implementation

MIFARE Classic utilize is a "high-frequency" RFID system on 13.56 MHz, and the transmission specification follows the ISO 14443-A standard. We develop our own GNU Radio signal processing blocks to demodulate Manchester and modify Miller codes on this frequency. Our software first sets the decimation rate and frequency for DSP board. The typical decimation rate and frequency are 32 and 13.56 MHz, respectively. Our software periodically receives 4096-byte data from USB and enters the signal processing block to process the data and convert them to packets for subsequent cryptanalyses. The software loops between data receiving and processing.

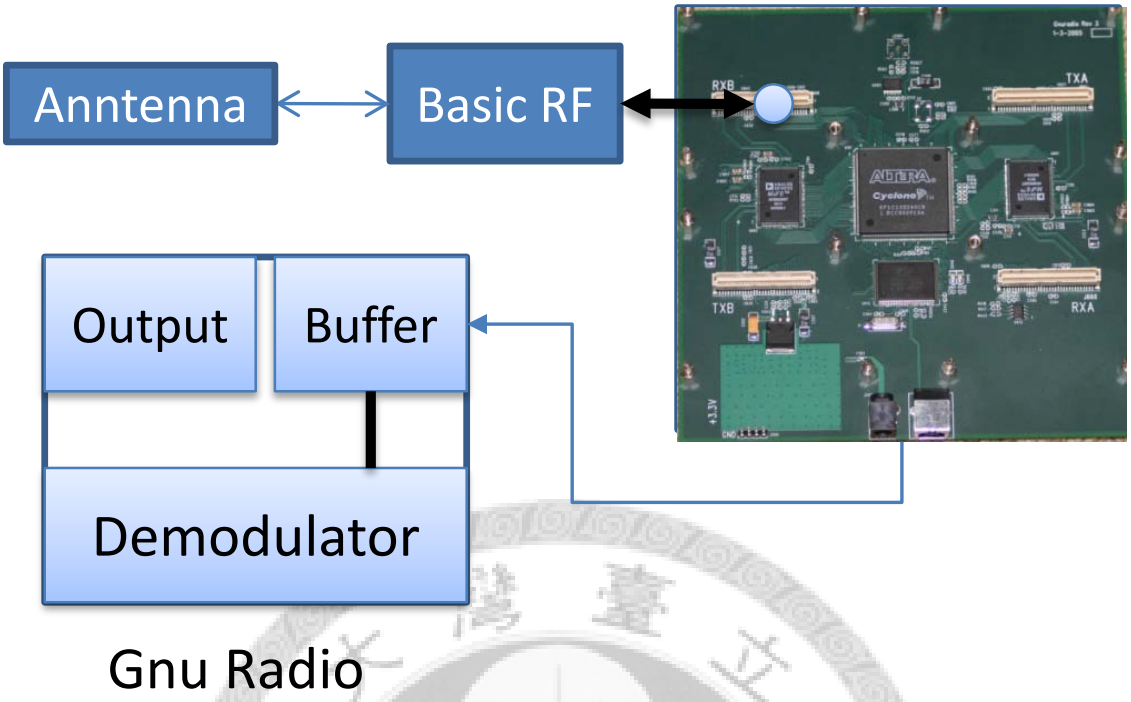

Figure 3.1: Block Diagram of the GNU Radio Sniffer

3.1.4 Converting to Raw Data to Packets

From the standard, we know that the start bits of PCD-to-PICC and PICC-to-PCD transmissions are logical zero and one, respectively. The start bit is used to synchronize the data transmission. After the start bit is found, the data demodulation becomes simple. Hence, the most important task is to find the start bit in the received signal.

Our signal processing block continuously scans the data buffer, trying to find synchronization points at which there are dramatic downward transitions from high to low in the signal. Starting at each of these synchronization points, our software captures about *18* subsequent samples to determine one bit. This number of sample points in one bit interval is decided by a simple formula: sampling rate/decimation rate/baud rate. The sampling rate is 64 MHz, the decimation rate is 32 sample per seconds, and the data baud rate is 106 kilobits per second, resulting in the aforementioned number. Concerning about physical transition, load modulation is very sensitive to the distance between the antenna and the source. Hence, PICC's signal will disappear when the distance is larger than 10 centimeters. Although the distance affects load modulation, PCD's signal is 100% ASK

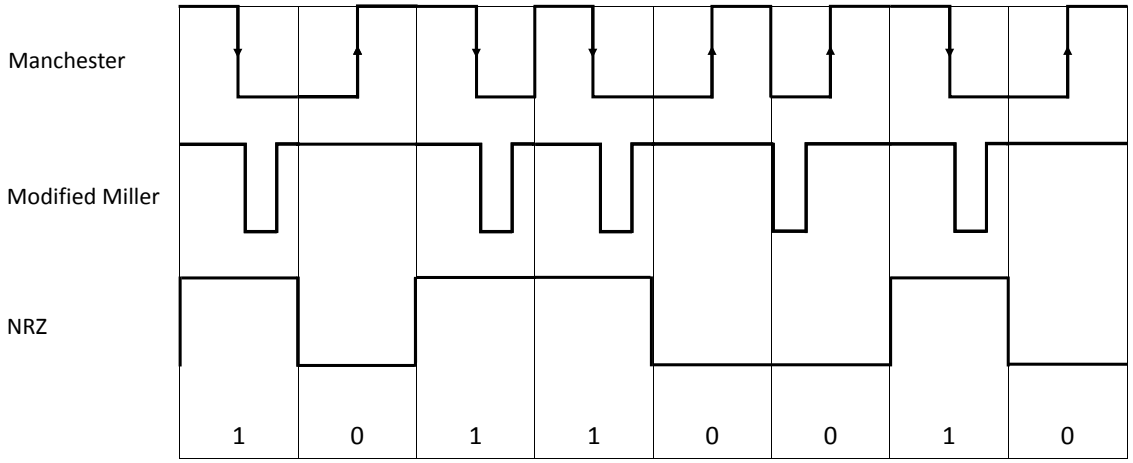

Figure 3.2: Signal coding by frequently changing line codes in RFID systems

modulation and hence is much stronger than that of PICC signal.

In our experiment, the PCD signal can be captured at a distance of at least two meters away. We show an example trace showing an REQA at the beginning of an anti-collision packet in Figure 3.4. While the sampled value of a low signal is approximately zero, the value of a high signal becomes smaller as the distance between PCD and antenna increases. The difference between the high and low signals keeps degrading as distance increases, to a point at which the level of background noise exceeds that of the high signal with a non-negligible probability. At this point, the packet error rate becomes too high, and the subsequent cryptanalyses becomes extremely challenging.

We scan from right to left to find synchronization points, as well as divide signal into bits. To convert raw samples in each of the intervals to bits and to merge bits into packets are the main tasks for our sniffer. In the following, we will explain how to convert a bit in an interval.

PCD ← PICC

An example of the captured load-modulated traffic from PICC to PCD is shown in Figure 3.5. To decode Manchester coding, we divide one time quantum into two parts. In Manchester coding, a waveform change from high to low represents a logical “one”, whereas that from low to high, logical “zero”. However, the load modulation in the sub-

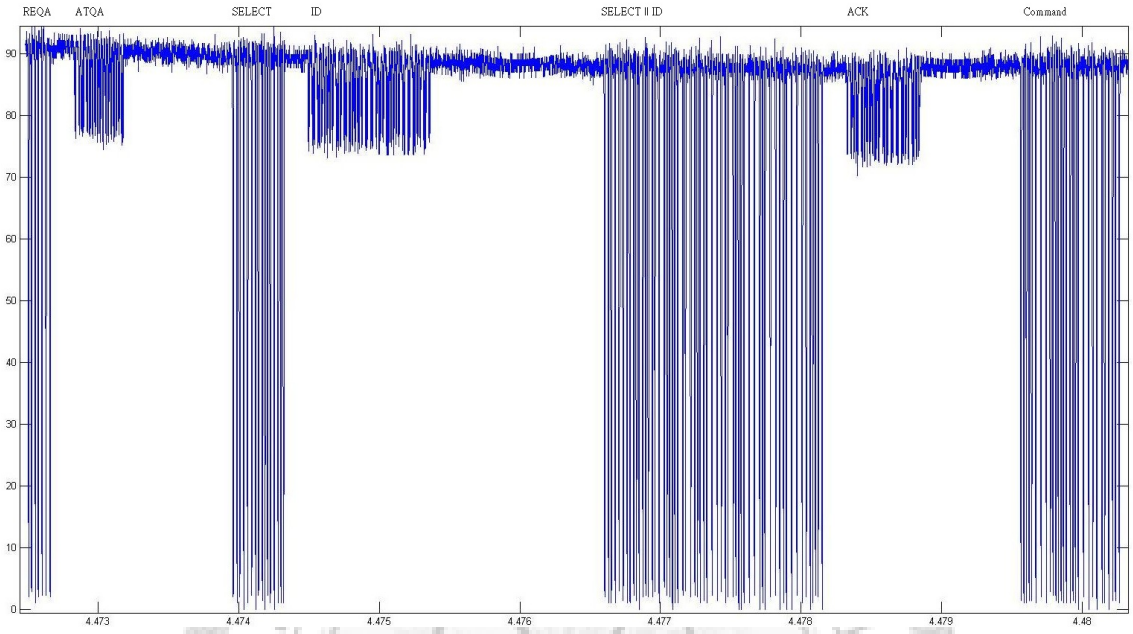

Figure 3.3: Normalized Raw Samples in an Oscilloscope

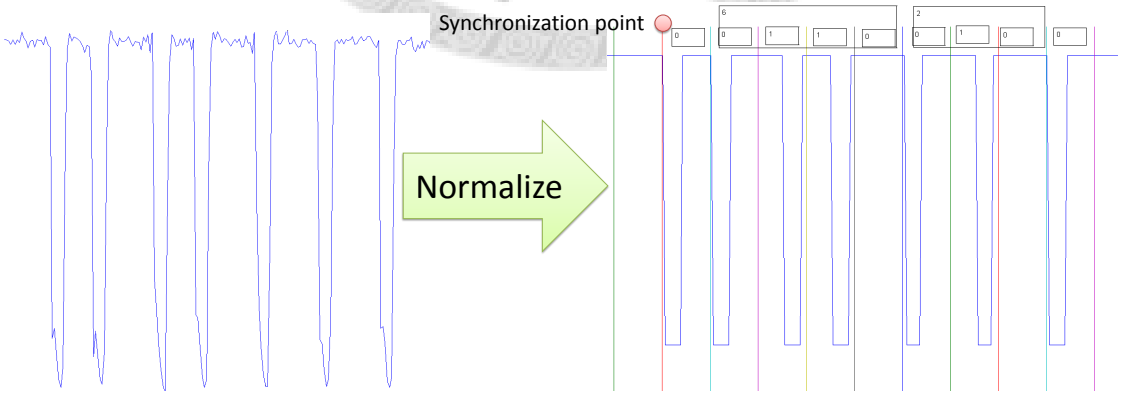

Figure 3.4: Normalized Raw Samples for REQA (0x26)

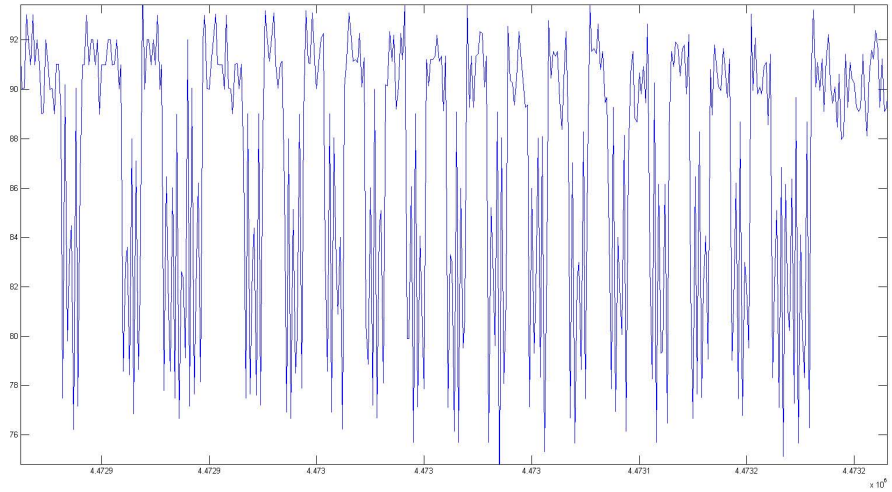

Figure 3.5: Raw Samples of ATQA (0x40)

carrier is sensitive to environment factors including distance, angle, and direction between PICC and antenna. If the antenna is not put on the right position, then the demodulation of the load modulated signal might fail.

In our software, the average of the samples in an interval is used as the base value that is used to determine whether the signals are low or high. Every sample in the interval is compared against this base value and declared a high if it is larger and low otherwise.

At the beginning, our software recursively compares the value between previous point V_{pre} and current point V_{cur} to decide the synchronization point. The condition of comparison is defined as the ratio R such that $R \times V_{pre} < V_{cur}$. Assume there are 18 samples in an interval, $v_0, v_1 ...., v_{17}$, the average of the samples is $\overline{V}$, and the local maximum and minimum are $local_{max}$ and $local_{min}$, respectively. If the lower bound of a logic zero is $lower_0$, then

$$lower_0 = \overline{V} - |local_{max} - local_{min}| \times 0.1,$$

where the threshold ratio of 0.1 is determined experimentally.

Finally, we decide logical one and zero by counting low and high signal in an interval. If the number of high samples in the first half exceeds 5, then we declare that the bit is a logical one. Otherwise, the bit is logical zero.

PCD → PICC

Another transmission code is modified Miller code. We apply the same method to decide synchronization points and base values as aforementioned. The PCD modulation is 100% ASK, which is easier to demodulate than load modulation. We also determine one-bit sampling intervals by dividing the time into two parts. If we detect a low signal in the second part, then we declare a logical-one bit and otherwise, a logical zero. Figure 3.4 shows a sample signal captured by our sniffer, in which the data bits are calculated based low-voltage samples.

3.2 Proxmark3

The Proxmark3 is a general-purpose RFID tool that can disguise as a PCD or PICC, as well as monitor the communication between commercial PCD and PICC [19]. The Proxmark3 works from low-frequency (125 kHz) to high-frequency (13.56 MHz). Proxmark3 consists of several hardware units, including an Atmel AT91SAM7S256 CPU, an Xilinx Spartan XC2S30 FPGA, as well asl analog-digital conversion circuitry, . . . , etc. The detailed description is given below for the sack of completeness of this thesis.

- CPU: ARM, 256 kB of flash memory, 64 kB of RAM
- FPGA: Xilinx Spartan II
- Two independent RF circuits, HF and LF
- Power: through USB port
- Connectivity: one mini-USB port
- User interface: one button, four LEDs
- Open-source design, both hardware and software

The hardware description and development can be found on the official website of Proxmark3 [19]. In this thesis, we utilize the Proxmark3 as PICC and PCD emulator.

In PCD-based attack, we program the ARM on Proxmark3 with `ARMCC`. First, the host PC tells Proxmark3 to emulate PCD. Before the Proxmark3 initiates the communication with PICC, it prepares the packets by pushing them to a stack in RAM. Secondly, it sets the FPGA to handle the appropriate signal channel. FPGA functions as a coprocessor that helps the ARM CPU transform the prepared data to the modulation circuit and takes the demodulated signal back to a buffer of the ARM CPU. We then need to decode the received packet and pass the relevant information to PICC.

Chapter 4

Our Attacks on MIFARE Classic

4.1 Time-memory Trade-off in Attacking CRYPTO-1

Time-memory trade-off (TMD) is a well-known method to effectively search the key space of a cipher. In a TMD scheme, if there are N possible solutions in the search space, it can allow the solution to be found in T operations with M words of memory for various combinations of T and M. The two extremes are exhaustive search (T=N, M=1) and table look-up (T=1, M=N), but there are no general time-memory trade-offs that have been published for attacking CRYPTO-1. In our attack, we recover the internal state in CRYPTO-1 from the segment of keystream by TMD, following the suggestions by Gan et al. [4]

The cryptanalyst first collects enough keystream bits on CRYPTO-1. He can then pre-compute the pre-images by a part of the collected keystream bits to store in the internal state table. Next, he checks whether every item of the table generates the rest bits of keystream and remove impossible states. With the remaining states, the correct keystream can be generate by CRYPTO-1, and the states can be linearly rolled back to initial states.

How many bits of keystream are enough? In theory, we need 48 bits of keystream, which would on average decide a unique internal state assuming the filter function in CRYPTO-1 is uniform. In practice, it is easy to get long keystream in one transaction on MIFARE Classic systems, which will be shown in Section 4.2.2 later. We can get one 64 and more than 32 bits in two-way and one-way traces, respectively.

4.2 Weakness in CRYPTO-1 and its Implementation

MIFARE Classic has dozens of vulnerabilities in CRYPTO-1 as well as its implementation. In this section, we enumerate these problems and discuss their implications.

4.2.1 CRYPTO-1 Structure

Small Key Size

It is customary to regard a modern cryptosystem as "secure" if it has a security level of at least 2^{80}. As an example, the well-known stream ciphers A5/1 and A5/2, used in voice encryption on mobile phone systems for more than twenty years, have internal states that are 64 bits long. Although CRYPTO-1 was invented later than A5/1 and A5/2, the length of the internal state in CRYPTO-1 is even shorter, only 48 bits long. To crack 48 bits by brute-force search is not a hard task for a modern PC nowadays. Thus, the key size or length of internal state in CRYPTO-1 is too short to resist such a brute-force search. Take our GPU implementation as an example. We can find an internal state in ten days by using a modern high-end NVIDIA GPU that costs less than 500 USD. This search work can be easily parallelized so that the cracking time is significantly shortened by increasing the number of GPUs.

Input Size of Filter Function

Another serious issue in CRYPTO-1 structure is that only a small number of bits in the internal state are used as the input to the filter function. Recall that the input of filter function in CRYPTO-1 consists of 20 bits from the 48-bit internal state, as shown below in polynomial form:

$$x^9, x^{11}, x^{13}, x^{15}, x^{17}, x^{19}, x^{21}, x^{23}, x^{25}, x^{27}, x^{29}, x^{31}, x^{33}, x^{35}, x^{37}, x^{39}, x^{41}, x^{43}, x^{45}, x^{47}.$$

Observation 1. ***Input of the Filter Function***

The filter function only takes bits in the odd positions as the input. This implies that

the 48-bit register can be thought of consisting an odd register and an even register. To generate an output bit, the input of the filter switches between odd and even register. At some point, the filter function takes the odd register as input and then shifts the internal state. Next, the even register becomes the input to the filter function and is shifted again. As a result, the odd and even registers takes turn as the input to the filter function. This observation makes the task of recovery much easier.

Observation 2. ***Pre-image Size of the Filter Function***

Given two-level filter function $f : X \rightarrow Y$ *in CRYPTO-1,*

$X = \{x | x \in \{0, 1, 2, ..., 2^{20} - 1\}\}$,

$Y = \{y | y \in \{0, 1\}\}$.

Assume

$X_0 = \{x_0 | x_0 \in X,\ s.t\ f(x_0) = 0\}$,

$X_1 = \{x_1 | x_1 \in X,\ s.t\ f(x_1) = 1\}$.

Then

$O(X_0) = O(X_1) = 2^{19}$.

The observation indicates that the size of the pre-image of the filter function is 2^{19}. Assume we need four bytes to store each item in the table, then the pre-image takes about two megabytes to store. This size is much smaller compared with $O(2^{47})$, the size of the pre-image of the 48-bit internal state, which needs more than 128 terabytes to store. This makes TMD on inverting the filter function feasible for modern-day commodity PCs.

4.2.2 Plaintexts that Provide Consecutive Keystream Bits

With TMD, we can recover the secret key on CRYPTO-1 with a small amount of memory consumption and time when we can obtain a sufficient number of keystream bits. At least two routes that we know provide the attacker with long segment keystream from communication trace. The first route is through the three-way authentication, through which PCD and PICC achieves mutual authentication if they share a same secret key. An example data trace for the authentication looks like:

$$\begin{array}{lll} \text{PCD} \leftarrow \text{PICC} & N_t & \\ \text{PCD} \rightarrow \text{PICC} & \{N_r\}\ \{A_r\} & A_r = suc^{64}(N_t) \\ \text{PCD} \leftarrow \text{PICC} & \{A_t\} & A_t = suc^{96}(N_t) \end{array}$$

We can then obtain 64-bit consecutive keystream bits via

$$\{A_r\}\{A_t\} \oplus A_r A_t.$$

A second route is via nested authentication. In a typical MIFARE Classic transaction, there usually involves more than one memory operations after a three-way authentication succeeds. The memory operations includes *read*, *write*, *increment*, *decrement*, and *restore*, and they manipulate the memory blocks in the authenticated sector. These memory operations are encrypted 32-bit commands, each of which consists of one-byte command code, one-byte block number, and two-byte CRC. The two-byte CRC is computed over the first two bytes. Therefore, the command can be determined by guessing the first two bytes. In order to assist our guessing code, we classify the memory operations into four categories, namely R, W, V, and A, according to the amount of exchanged data. When the category of the command is known, there are at most four possibilities in block number. In addition, we can determine the unique command with the help from the *encrypted parity check bits*, whose detail will be shown in Section 4.4. In any case, it is easy for the cryptanalyst to obtain more than 32 consecutive keystream bits via the second route.

4.2.3 Implementation Vulnerabilities

PRNG

The PRNG in MIFARE Classic PICC is always initialized with the same seed. In other words, the sequences of the pseudo random number generated are picked from a small fixed set. It is then much easier for the attacker to hit the current number from 2^{16} possible numbers in the set. Besides, the attacker can examine whether the deciphered random number is in the set to check whether the key is correct.

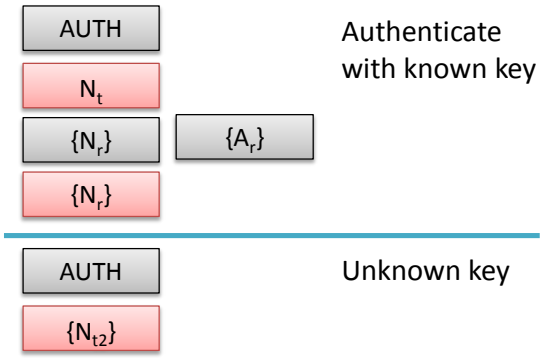

Figure 4.1: Nested Authentication Scheme

Technique 1. *Examining encrypted nonce* n

We denote the set generated by MIFARE Classic's PRNG as MC_{PRNG}*. Each element in* MC_{PRNG} *is a 32-bit number* $n_0 \ldots n_{31}$ *satisfying the property that the first 16 bits can be generated by the other 16 bits. That is,*

$$n_{16+i} = n_{7+i} \oplus n_{5+i} \oplus n_{4+i} \oplus n_i \ \ \forall i \ = \ 0 \text{ to } 15.$$

Say we have $n \in MC_{PRNG}$ *and an encrypted nonce* $\{n\}$ *in the trace. We can generate the corresponding keysteam* ks *by CRYPTO-1 from an internal state* S' *under test. Then, we can check whether*

$$ks \oplus \{N_1\} = N_1' \ \in MC_{PRNG},$$

and if it is true, then ks *may be a correct key.*

Nested Authentication

After the first authentication successfully completes by a known key, PCD can immediately authenticate another sector with an unknown key. The authentication command is encrypted by the current keystream that is generated from the known key, but the tag challenge nonce N_t is encrypted by a new keystream generated from the unknown key.

This implementation vulnerability is a derivative of the PRNG problem, which is shown in Figure 4.1. Because of the small nonce space, we can recover the possible N_t and reveal the keystream that encrypted N_t fairly easily. This vulnerability also applies to

PCD-based remaining-key attack in Section 4.3.2.

Encrypted Parity Bits

In order to ensure data integrity, MIFARE Classic transmits a parity bit for every byte transmitted. Computing parity bits over ciphertext does not leak any information about the plaintext since they can be computed from the ciphertext by anyone. However, the parity bits in MIFARE Classic are computed over plaintext and are encrypted by the next keystream bit. This leaks one bit of information about the keystream for every byte transmitted.

We use a simple example to illustrate how we utilize this vulnerability to launch an attack. Suppose that we are given one byte of ciphertext $C = c_1, \ldots, c_8$ along with its parity bit $\{p\}$. Assume the plaintext T is of $t_1, \ldots, t_8$. In our attacks, we usually try many internal states in parallel, decrypting the trace with each of them. We assume the corresponding keystream bits generated from the internal state I_i are $k_1, \ldots, k_8, k_9$. We can check

$$\{p\} \oplus k_9 = c_1 \oplus c_2 \oplus ... \oplus c_8 \oplus k_1 \oplus k_2 \oplus ... \oplus k_8.$$

Each candidate has a probability of 1/2 to pass this check, so on average we can eliminate half of the candidate internal states for every byte we capture.

Encrypted Error Code

As explained in Section 2.6.3, PICC will send out an error code when PCD's response passes the parity check. This happens roughly with a probability of 1/256. In practice, it only takes a few minutes to acquire enough error-code traces. For cryptanalysis, an error-code trace provides 12-bit of information on the candidate internal states. This implementation vulnerability allows an attacker to launch PCD-based attack, as described in Section 4.3.1.

Observation 3. *Efficiency of obtaining keystream*

[Encrypted Error Code]

For four bits of keystream, we can get four bits of information.

[Encrypted Parity]

For eight bits of keystream, we can get one bit of information.

4.3 PCD-based Attack

PCD-based attack, also known as card-only attack, requires a programmable PCD like Proxmark3. The PCD emulates a MIFARE Classic reader and communicates with the PICC.

The PCD-based attack can be classified based on which key it can recover. If the attacker does not know any key, he needs to launch the first-key attack to recover any key on the target PICC. After obtaining a first key, he can launch a remaining-key attack via nested authentication.

4.3.1 First-key Attack

The first-key attack was first proposed by Garcia et al. First of all, we need to obtain enough information about the secret key from several error-code traces. To acquire error-code traces, we let the programmable PCD try to repeatedly authenticate the chosen block on the target PICC. In the authentication session, when the parities of the randomly generated 64-bit PCD response are correct, the target PICC will send back an encrypted error code, giving four bits of information on the keystream. This happens with a probability of 2^{-8}, so we can in effect obtain 12 bits of information (8 from parities plus 4 from the error code) about the secret key for, on average, every 256 requests we send. We can then attempt to recover PICC's internal state using brute-force search. Garcia et al. reported that the estimated time for such a search is 36 minutes using a special-purpose code-breaking FPGA cluster, COPACOBANA [7], that costs about 20,000 euros.

Checking the characteristic of the encrypted error code is the main work of brute-force attack on GPU. The cracking itself is simple: we concurrently put every key in the cipher to decrypt the trace and check whether the plaintext passes the check or not.

Brute-force Search on GPU

Today's graphics cards contain powerful GPUs to handle the increasing number of screen pixels in video games. Now GPUs have developed into a powerful, massively parallel computing platform that finds more and more interest outside graphics applications. In cryptography, there have been many attempts of exploiting the computational power of GPUs. Furthermore, there has been evidence that GPUs can be a quite competitive computing platform in terms of price-performance ratio.

We adopt GPUs as our search platform. In our implementation, the total number of concurrent threads is 2^{19}, the maximum number allowed in CUDA 2.0. Each thread handles about 2^{48-19} states from the 2^{48} possible states. The main task of each thread is to generate the keystream bits from the assigned internal states and check against the traces. Each thread goes through many iterations, each of which consists of generating one bit of keystream, filtering, updating, and rolling back.

- Filter: $t_i = f(LFSR_i)$ means to take 20 bits from LFSR to output t_i.
- Update: $u(n)$ means to take LFSR to n time slots later.
- Roll back: $r(n)$ means to roll back LFSR n time slots.

To finish examining all error-code traces, each thread needs to generate at least 272 bits of keystream per internal state. Hence, each thread needs to abort searching as soon as a candidate can not satisfy all the check conditions.

Reducing Computation

To speed up the recovery process, the total number of operations needs to be reduced as much as possible. Our strategy is to start from an appropriate position of the internal state. We label the possible starting points as t_0 to t_{99}, which is shown in Figure 4.2. A proper starting point can result in fewer number of operations by eliminating incorrect states early on without wasting time on hundreds of thousands of redundant operations. As we have mentioned earlier, MIFARE Classic suffers from the encrypted parity and encrypted error-code vulnerabilities. Thus, we see that the encrypted error-code vulnerability leads

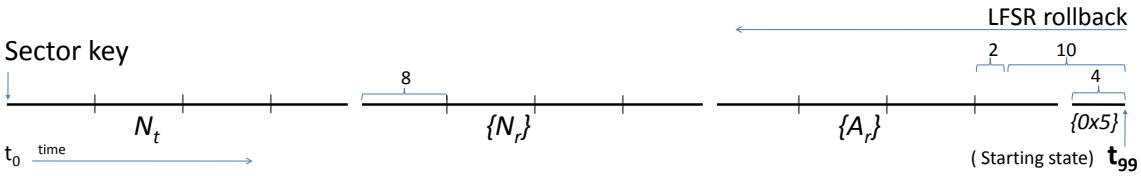

Figure 4.2: Error-code Examination Scheme

to more effective attacks to obtain keystream bits than the encrypted parity vulnerability. If the goal is to get as much keystream information at a minimum cost, then it is much better to set the starting point near t_{95}, as the internal state at this point can give four bits of information on keystream via the error-code vulnerability. Also, we can generate 10 keystream bits without updating the internal state. In practice, we find that setting the starting point to t_{99} is the best. The reason is that it minimizes the number of internal state updates, which is the most expensive operation.

As shown in Figure 4.2, we use two $r(1)$ and 12 $f(LFSR_j)$ to obtain five bits of information from a four-bit error code and one encrypted parity bit at stating point t_{99}. This position rapidly reduces the number of states check to 2^{-5} of the original number, resulting in a minimum number of operations. We repeatedly produce the corresponding keystream bits by testing candidate states against decrypted traces, for which we check the plaintext and roll back LFSR until it goes back to time t_{32}. Formally, we roll back LFSR with N_{t1} from trace 1 and update with N_{t2} from trace 2. There are more efficient way to convert the state. We can pre-compute the state that puts N_{t1} or N_{t2} in LFSR with zero as the initial state and call them S_1 and S_2, respectively. Then the LFSR at t_{32} with N_{t1} is

$$LFSR\_N1_{32} \oplus S_1 \oplus S_2 = LFSR\_N2_{32}.$$

This reduces to 32 LFSR update plus 32 LFSR rollbacks. The remaining process is to take each state that passes the check to generate the keystream bits for the remaining traces. The correct state will successfully decrypt all the traces. In our experience, we usually need six traces to determine the correct internal state.

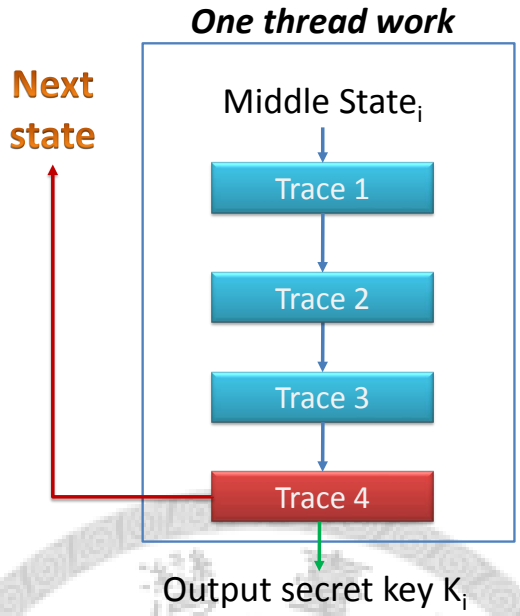

Figure 4.3: The Error-code Check Flow

Correctness

The output of CRYPTO-1' filter function is fairly uniform in the sense that the number of zero and one bits are roughly equal. This means that we can eliminate about half of the candidate states for every bit of information we have on the keystream. One error-code trace consists of eight bits of encrypted parities plus four bits of keystream obtained via error code, so each error-code trace contains 12 bits of information on the keystream. After checking against one error-code trace, only one can survive among 2^{12} candidate states. On average, the probability to pass four error-code traces is 2^{-48}. However, in practice we almost always need five or six error-code traces to arrive at a unique starting internal state. Fortunately, the recover time for four and six traces are approximately the same, as there are very few states left after passing four successful checks

4.3.2 Remaining-key Attack

The attack described in this section assumes that at least one secret key on a MIFARE Classic PICC is known by the attacker. Applying the principle of *nested authentication*, the attacker can obtain some information about the unknown keys is from the encrypted

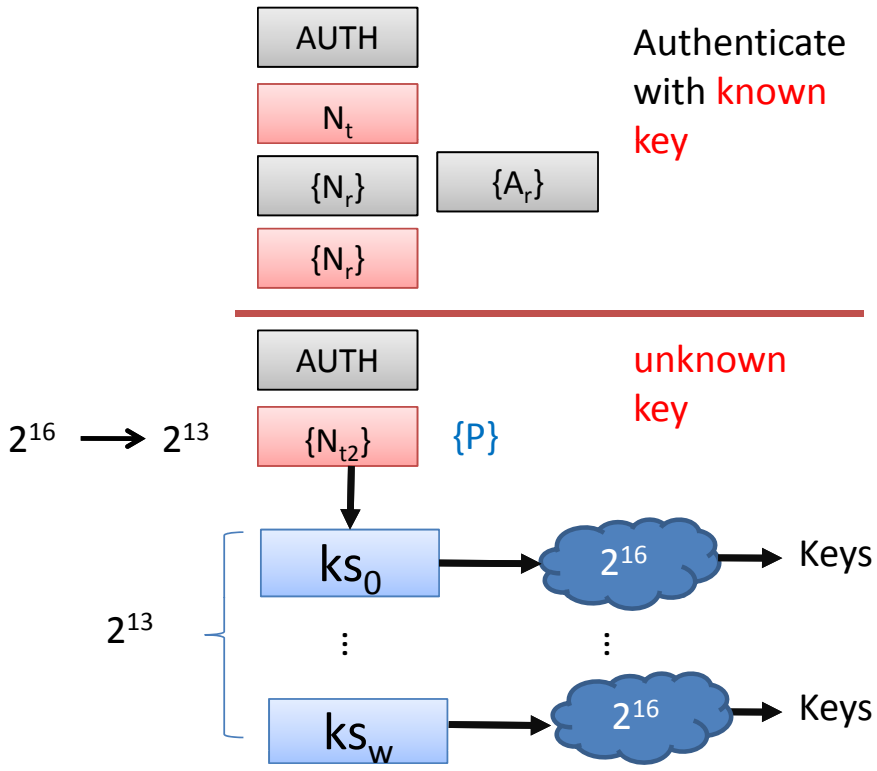

Figure 4.4: Guessing N_{t2} from N_{t_1} and Recovery from ks to Internal States

nonces. As shown in Figure 4.1, we take the first key and pass the three-way mutual authentication. After the first authentication succeeds, we immediately authenticate another sector whose sector key is unknown to get an encrypted nonce $\{N_{t2}\}$. In order to gather more nonces $\{N_{t3}\}$, $\{N_{t4}\}$, and $\{N_{t5}\}$ to assist us to eliminate incorrect states, we can repeatedly execute more *nested authentications*. Next, the PRNG vulnerability lets us obtain 32 bits of information about the keystream ks from N_{t2}. In practice, we calculate N_{t2} from $suc^i(N_t)$, and the total possible numbers of i is 2^{16}. Furthermore, 2^{16} can be reduced to 2^{13} by checking against the three encrypted parity bits. For the above analysis, there are 2^{13} probable 32-bit keystreams. We can test these possibilities one by one.

Observation 4. *Reducing possible nonces*

Given:

encrypted four parity bits $P_n = \{p_0\}, \{p_1\}, \{p_2\}, \{p_3\}$

encrypted nonce $\{N_{t2}\} = \{n_0\}, \ldots, \{n_{31}\}$

Assume:

$N_{t2} = n_0, \ldots, n_{31}$

the corresponding keystream bits $k_0, \ldots, k_8, k_{31}, k_{32}$

We know:

$$\{p_0\} = n_0 \oplus n_1 \oplus \cdots \oplus n_7 \oplus k_8$$

$$\{p_1\} = n_8 \oplus n_9 \oplus \cdots \oplus n_{15} \oplus k_{16}$$

$$\{p_2\} = n_{16} \oplus n_{17} \oplus \cdots \oplus n_{23} \oplus k_{24}$$

$$\{p_3\} = n_{24} \oplus n_{25} \oplus \cdots \oplus n_{31} \oplus k_{32}$$

Compute:

$P =$

$$p_0 = n_0 \oplus n_1 \oplus \cdots \oplus n_7,$$

$$p_1 = n_8 \oplus n_9 \oplus \cdots \oplus n_{15},$$

$$p_2 = n_{16} \oplus n_{17} \oplus \cdots \oplus n_{23}$$

$$p_3 = n_{24} \oplus n_{25} \oplus \cdots \oplus n_{31}$$

and

$$\{P\} \oplus P = k_8||k_{16}||k_{24}||k_{32}$$

Check:

$$\{P\} \oplus P == (n_8 \oplus \{n_8\})||(n_{16} \oplus \{n_{16}\})||(n_{24} \oplus \{n_{24}\})$$

The total number of states is then reduced from 2^{16} *to* 2^{13}.

Recovering Secret Key

To examine 2^{13} possible 32-bit keystreams, the following process will run 2^{13} times. In each iteration, we recover the 32 keystream bits by rolling back N_{t2} to some internal states. These states can be rolled back to the secret key, and then we can generate keystreams to decrypt $\{N_3, N_4, N_5\}$. If all encrypted nonces are still in MC_{PRNG}, then the secret key is correct.

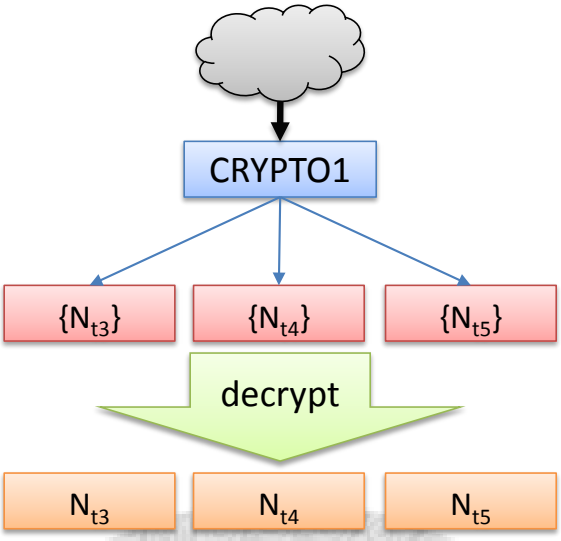

Figure 4.5: Examination of the Recovered States

4.4 Sniffer-based Attack

In Section 2.3, we know the memory access is dictated by the access condition in a sector tail, so there are only three possibilities in block number once we know the sector number of a command. If we do not know the sector number, the full search size is 16 sectors times 3 blocks per sector equals 48, which is a very small number to try. In practice, the sector number can always be deduced from the authentication command.

4.4.1 Keystream to Internal States

At the beginning of this chapter, we have introduced that TMD is a frequently used technique to recover keystream bits from numerous internal states in a stream cipher. Since the number of possible inputs to the non-linear filter function is small, we can create a table using a reasonable amount of memory from a sequence of keystream bits. In the sniffer-based attack, the main task is to capture sufficient keystream bits from the eavesdropped traffic and then to recover the secret key using TMD.

Inc/Res/Dec	Write	Authenticate	Read
{Inc/Dec/Res N}$_{32}$	{Write N}$_{32}$	Auth N $_{32}$	{Read N}$_{32}$
{ACK/NCK}$_{4}$	{ACK/NCK}$_{4}$	N_t $_{32}$	{Data}$_{144}$
{Value + CRC}$_{48}$	{Data || CRC}$_{144}$	{N_r}$_{32}$ {A_r}$_{32}$	{Next Command}$_{32}$
{Transfer}$_{32}$	{ACK/NCK}$_{4}$	{At}$_{32}$	
{ACK/NCK}$_{4}$	{Next Command}$_{32}$	{Next Command}$_{32}$	
{Next Command}$_{32}$	The suffix number is bits of sent packet.		

Figure 4.6: Example Sequence of Commands

Table 4.1: An Example One-way Trace

1	AUTH 0x18	6118e4fe
2	{N_r} {A_r}	3edee7b0 3f307d3e
3	{WRITE 0x18}	98c9b913
4	{write data}	...(18 bytes)
5	{AUTH 0x8}	89be2cea
6	{N_r} {A_r}	1433ad1452895e0c
7	{DEC 0x8}	8d02026d
8	{dec. value}	a2ef4ab078a9
9	{TRAN 0x8}	84aaacec
10	{READ}	5f815afa
11	{AUTH 0x1a}	fbf8c3d9
12	{N_r} {A_r}	bcd863a91cf83b07
13	{WRITE 0x1a}	6fb38b89
14	{write data}	...(18 bytes)
15	{AUTH 0x10}	ff35fcc0
		⋮

4.4.2 Attack Based on Two-way Traffic

Garcia et al. reported that if the sniffer is very close to PICC, it is possible to recover the secret sector key from a trace of 128 consecutive keystream bits [1]. The last 64 bits are used to recover the LFSR state, while the first 64 bits are used to roll back from state to secret key. See Figure 2.7 for a detailed illustration.

4.4.3 Long-distance Attack

In our long-range attack, we can only capture packets transmitted from a legitimate PCD to target PICC. Table 4.1 shows an example trace of such one-way communication. The

idea is to launch a known-plaintext attack. A MIFARE command is composed of a one-byte command byte, a one-byte block number, and a two-byte cyclic redundant check (CRC) code. There are only a handful types of command bytes, the most common ones immediately after being read (READ), write (WRITE), increment (INC), decrement (DEC), and restore (RES). After anti-collision, PCD will select a PICC and send out an authentication request to the selected PICC. Each sector consists of three data blocks and one sector tail. The data block can be operated upon by all commands, while the sector tail can be written or read. Therefore, the total number of possible plaintexts in an encrypted command request is 17: five operations times three data blocks plus read or write to sector tail. With such a small space of plaintext, the complexity of launching a known-plaintext attack is quite low.

Furthermore, we can distinguish different types of commands based on the length of subsequent transmission. As shown in Table 4.1, the number of bytes transmitted after a DEC command (item 5–10) and that after a WRITE command (1–4 as well as 11–14) are different, so it is easy to distinguish these two types of command.

Our long-range attack works as follows. After capturing the authentication request, we immediately know which sector the PCD is going to operate on. Then, to recover the LFSR state, we need to guess the command type as well as block number based on the length of the transmission. We classify the commands into four types: A (next authentication), R (READ), V (INC, DEC and RES), and W (WRITE). Note that the first command of each type is always four bytes long.

1. Type A: the length signature is 4-8, with the eight-byte data being the PCD response;

2. Type R: the length signature is 4, as a READ requires no further communication from PCD to PICC and is always followed by another type of command;

3. Type V: the length signature is 4-6-4, consisting of a six-byte value followed by a four-byte transfer (TRAN);

4. Type W: the length signature is 4-18, as the write data is always 18 bytes long.

Table 4.2: Platform Comparison in First-key Attack

Platform	GPU	COPACOBANA
Time (minutes)	480	36 (estimated)
Cost (euros)	3,000	20,000
Time-cost product	1.44×10^6	7.2×10^5

This classification makes guessing command easier.

The next step is to decide the block number. The protocol specifies that the block must belong to the same sector as the authenticated block. Since we know the authenticated block number, there are only four possible block numbers. Furthermore, for each possible candidate plaintext, we can immediately obtain the corresponding 32-bit keystream. We can then use the four parity bits to check whether our guess is correct.

In practice, we almost always end up with only one possible plaintext, with which we can obtain 32 keystream bits and reduce the possible LFSR states at this point to 2^{16}. We then roll back each of these states and check against $\mathtt{A}_\mathtt{r}$. Here we use another implementation flaw of MIFARE Classic: the space of all possible $\mathtt{A}_\mathtt{r}$ is 2^{16} because it is a function of $\mathtt{N}_\mathtt{t}$, which is generated by an LFSR with a degree-16 generating polynomial and hence only has 16 bits of entropy. Therefore, there will be, on average, only one state left after such checking. Sometimes we are left with more than one candidate, in which case we simply check against the four parity bits in $\mathtt{A}_\mathtt{r}$ and eliminate the false positives.

The speed of sniffer-based attacks is fast. Recovering a session to the secret keys usually needs slightly more than 10 seconds even on a low-end laptop equipped with an Intel Atom-330 CPU running at 1.6 GHz. If attacker knows another 32-bit command, then the recovery time will be further reduced to about 2 seconds.

4.5 Comparison

4.5.1 Implementation Improvement

In this section, we report the performance of our implementations of the PCD-based and sniffer-based attacks.

Table 4.3: Experiment Results

	PCD offline		**Sniffer online**
	First key	Rest key	Long range
Platform	GPU	CPU	CPU
Device #	16	4	1
Time/per key	8 hour	1 hour	less than one min

We stress that our implementation of the PCD-based attack deviates from the suggestions made by Garcia et al. [5] They did not implement the first-key attack but only gave their estimation based on experience with breaking 56-bit DES. For the remaining-key attack based on nested authentication, they proposed to control the time duration between the first and second nonce in order to predict the second nonce precisely. In practice, to control timing is much more difficult than to compute with a cluster. Therefore, we use off-the-shelf GPU and CPU computing platforms.

Gen et al. [4] proposed the sniffer-based attack that recovers secret keys with two-way traffic traces. This attack is restricted due to severe physical constraints, as it requires that the distance between antenna and target PICC be very close. In most cases, the two-way sniffer-based attack is of little value because one can simply launch a PCD-based attack at this distance.

As an improvement, our long-range sniffer-based attack can recover secret keys from one-way traffic traces. This is much more practical than the original attack.

For all of our attacks, we overcome many physical constrains to provide more practical attacks on MIFARE Classic. To compare with others' attacks, our methods are more efficient and more practical in most scenarios. Our experiment results are summarized in Table 4.2 and Table 4.3.

Chapter 5

Proposed Defenses for MIFARE Classic

As we can see, the aforementioned attacks essentially turn MIFARE Classic into a dumb memory card without any cryptographic protection. The wide deployment of MIFARE Classic makes it financially prohibitive to replace the broken hardware with secure alternatives. In this chapter, we propose a defense mechanism that can prevent most attacks without requiring any hardware changes. Such a mechanism can be used as an interim solution before the deployment of next-generation, secure RFID ticketing systems to replace MIFARE Classic.

The basic idea of our proposed mechanism is to cryptographically provide integrity protection for the data on a MIFARE Classic PICC. The computation is done on PCD; hence in most cases we only require software upgrades on PCD. For this, we can either use symmetric- or public-key cryptography. As such a system is often deployed over a wide area across many administrative domains, the key management will be a very challenging task if we use symmetric-key cryptography. However, traditional public-key cryptography like RSA can also be prohibitively expensive on devices like MIFARE Classic PCD, which typically only have very limited computational power. For this reason, we suggest the new multivariate public-key cryptosystems (PKCs), which enjoy two advantages over traditional counterparts like RSA: they run much faster and can survive the attack of the emergent thousand-qubit quantum computers [20].

5.1 Multivariate PKCs: TTS

In multivariate public-key cryptosystems [20], the public key is a set of m polynomials $\mathcal{P} = (p_1, \ldots, p_m)$ in variables $\mathbf{w} = (w_1, \ldots, w_n)$ where variables and coefficients are in $\mathbb{K} = \mathrm{GF}(q)$, built via invertible affine maps S and T:

$$\mathcal{P} : \mathbf{w} \in \mathbb{K}^n \overset{S}{\mapsto} \mathbf{x} = \mathrm{M}_S\mathbf{w} + \mathbf{c}_S \overset{\mathcal{Q}}{\mapsto} \mathbf{y} \overset{T}{\mapsto} \mathbf{z} = \mathrm{M}_T\mathbf{y} + \mathbf{c}_T \in \mathbb{K}^m.$$

The central map $\mathcal{Q}$, which can be inverted efficiently, decides the properties of the cryptosystem and S and T make $\mathcal{P}$ appear random.

In a type of $\mathcal{Q}$ called Unbalanced Oil and Vinegar [21], some variables ("oil") never appear in the quadratic terms. We set remaining variables ("vinegar") randomly and solve for oil variables in a linear system to get a digital signature.

For efficiency, we can stack together $m_1 + m_2$ equations in a "2-layer Rainbow:" m_1 equations in a UOV with v vinegar and m_1 oil variables, and the remaining m_2 a UOV with the $v + m_1$ earlier variables as vinegar and the final m_2 as oil.

Making polynomials sparse such that each equation has as many crossterms as vinegar variables, we get "2-stage TTS" with parameters (q, v, m_1, m_2). Here $m = m_1 + m_2$, $n = m + v$. TTS/Rainbow security and implementation has been studied for most of a decade [22]. TTS is very fast both on today's commodity PCs *and* low-end microcontrollers, as shown by the rough speed of a recommended instance TTS(256,18,12,12) in Table 5.1 for four hardware platforms.

- **C2+** Intel Core 2 Quad Q9550;
- **K10+** AMD Phenom II X4 905e;
- **i52** Intel 8052, a 8051 with 256-byte fast RAM, $T = 12$;
- **W58** Winbond W77E58, i52 clone with dual pointers and $T = 4$, T is number of clocks per instruction cycle.

In some delay-insensitive cases, both signature and verification can even be performed on a PCD equipped with 8051-grade microcontroller as they take a fraction of a second each.

Table 5.1: Speed of TTS on Various Platforms

Platform		Signature		Verification	
		cycles	time	cycles	time
C2+	2.83 GHz	36K	12.7 μs	121K	42.7 μs
K10+	2.50 GHz	46K	18.4 μs	155K	54.5 μs
i52	3.57 MHz	101K	341 ms	339K	1139 ms
W58	3.57 MHz	135K	152 ms	452K	506 ms

Furthermore, we can run TTS on FPGA if the PCD has an FPGA, which is a common practice nowadays in many niche-market embedded systems, as FPGAs provide greater cost-effectiveness and flexibility. We can optimizing for minimal area and current like Yang et al. [23] or for time-area product *a la* Bogdanov et al [24]. These designs are scalable, hence we may extrapolate from the earlier enTTS(20,28) and conservatively expect 38,000 gate-equivalents (GEs) for the former, and 89,000 GEs for the latter (in only a few hundred cycles at a few tens of MHz, compared to 10,000 for the former). So even the newer (and larger and slower) TTS should still run efficiently on even the smallest FPGA in most PCDs.

5.2 System Design

Figure 5.1 shows the memory layout of a protected PICC. We combine two sectors into a super sector, which can be used to store three data blocks along with the cryptographic checksum that protects their integrity. The checksum is computed over the entire three data blocks using a strong cryptographic hash function. The hash value is then digitally signed by the PCD using its private key, whose ID is also recorded along with the signature. We note that there is a 32-bit counter stored along with Key ID in the same block that is not shown in Figure 5.1. This counter value is used to prevent replay attacks, which we will discuss in more details below, and hence should be included in checksum computation as well.

Each PCD is equipped with a database of all PCDs' digital certificates. Therefore, each PCD is able to verify whether the data stored on a PICC has been tempered by

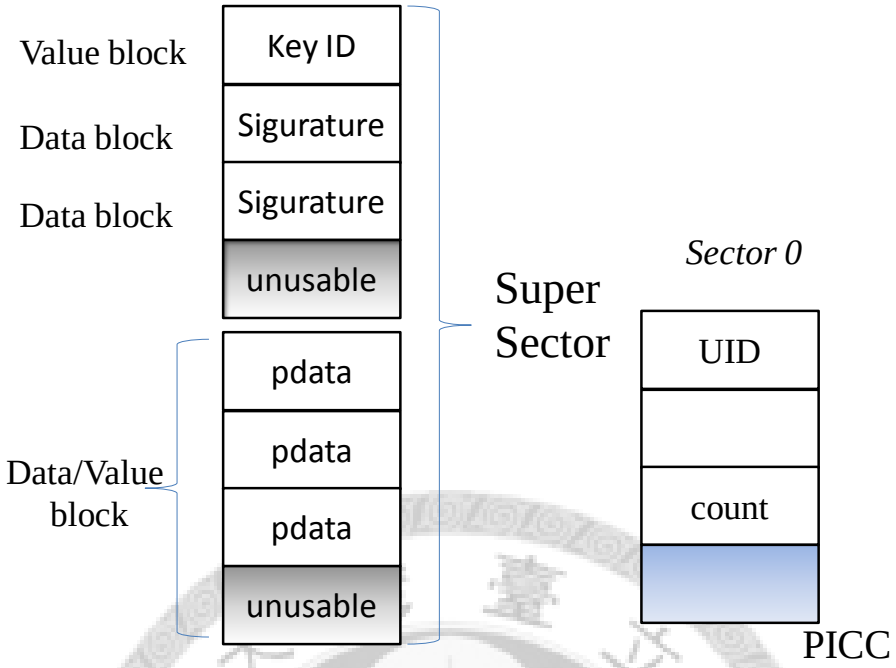

Figure 5.1: Proposed Defense Mechanism on PICC

an adversary when necessary. This prevents the attacker from altering the content of a PICC as long as he can not forge a valid signature. However, this does not prevent replay attacks, in which the attacker simply replace the content of a PICC with a previously valid value, resulting in what is called "double spending" in the literature of electronic payment systems.

Replay attacks can be prevented using a blacklisting mechanism on PCD, with which each PCD can check against whether a PICC has been compromised before conducting any transaction with it. Figure 5.2 shows the software modification needed on PCD in order to support data integrity check and blacklisting. The basic idea is to keep track of how many times a PICC has been accessed across the entire system. Conceptually, this per-PICC state is kept in the backend system, so breaking MIFARE Classic PICC does not give the adversary any benefits. However, to reduce communication overhead, we store this value on the sector 0 of each PICC so that each PCD can get this number locally when communicating with a PICC. To tie this count value with a transaction, we require that PCD include this value in cryptographic checksum calculation, as mentioned previously.

Each PCD keeps a transaction log for all the transactions it has done and synchronizes

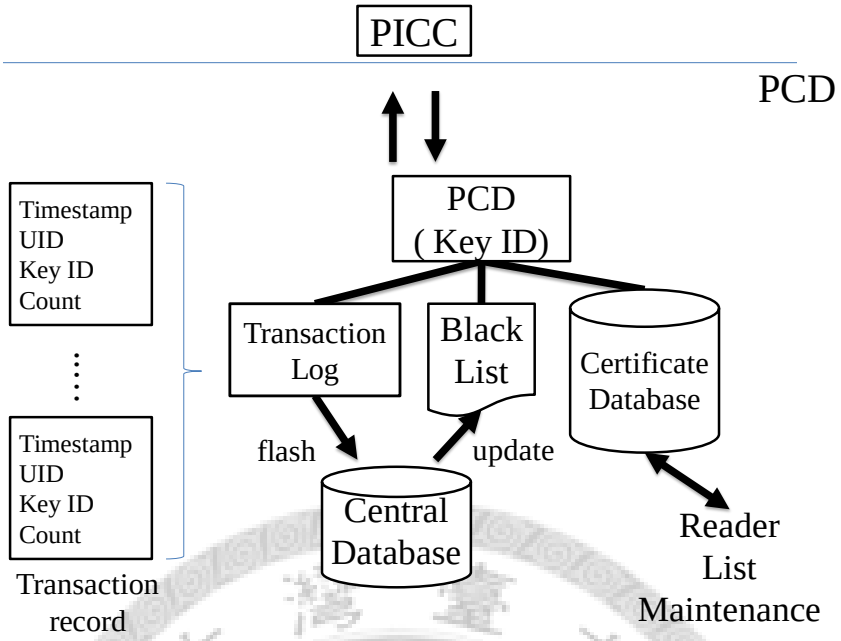

Figure 5.2: Proposed Defense Mechanism on PCD

with a central database periodically. For online PCDs, we could in theory synchronize on a per-transaction basis; however doing so can induce long latency and extra load on the central database and hence should be avoided.

The central database server checks the logs on a regular basis. Besides on verifying that each transaction is indeed conducted by some legitimate PCD, it can also detect double spending by noticing whether a valid PICC state has shown up twice or more in the collective log from all PCDs in the system. Upon detection of double spending, the server then records the UID of this PICC to a blacklist that is distributed periodically to all PCDs in the system.

Finally, we give a performance estimate for the required computational resources to implement the proposed defense mechanism. Assuming we have a busy system deployed in a metropolitan area with around 10 million transactions per day. Each transaction log has a 16-byte header containing timestamp, UID, Key ID, and counter value. In addition, it also contains the 48-byte data blocks under protection and a 32-byte signature. Together the size of these logs is less than one gigabyte per day spread across maybe thousands of PCDs in the city, which should be easy to store and transport with today's hardware.

The load at central database server is also reasonably light. With 10 million transactions each taking around 50 μs to verify using one single core of the processor on a commodity PC, the overall log checking time is about 500 seconds per day. This amount of time does not include the overhead of transferring the logs into the server and other housekeeping computation, which could add up to more than the time spent in cryptographic verification but is still easy to handle with today's servers.

Chapter 6

Conclusion

In this thesis, we have reported our experiment experiences attacking a real MIFARE Classic system. We have also presented our ideas how to defend against most attacks using practical mechanisms that do not require any hardware changes. Our proposed mechanisms can be easily implemented on a variety of MIFARE Classic readers on the market and only require commodity PCs be used in the backend system with intermittent network connectivity. Such mechanisms can be used as an interim solution before the deployment of next-generation, secure RFID ticketing systems to replace MIFARE Classic.

Lastly, it appears that it is still a common practice in the security industry to adopt proprietary ciphers and base the security of a system on the secrecy of the cipher design, despite the fact that history repeatedly shows the ineffectiveness of such an approach. It is the duty of the cryptographic community to educate the general public and demonstrate the danger of this mindset by cryptanalytic works of more practical security systems.

Bibliography

[1] F. Garcia, G. de Koning Gans, R. Muijrers, P. van Rossum, R. Verdult, R. Schreur, and B. Jacobs, "Dismantling MIFARE Classic," in *Computer Security — ESORICS 2008*, pp. 97–114.

[2] M. Hutter, J.-M. Schmidt, and T. Plos, "RFID and its vulnerability to faults," in *Cryptographic Hardware and Embedded Systems — CHES 2008*, pp. 363–379.

[3] S. K. Nohl, D. Evans, and H. Plotz, "Reverse engineering a cryptographic RFID tag," in *USENIX Security Symposium 2008*.

[4] G. de Koning Gans, J.-H. Hoepman, and F. Garcia, "A practical attack on the MIFARE Classic," in *Smart Card Research and Advanced Applications — IFIP WG 8.8/11.2*, 2008, pp. 267–282.

[5] F. D. Garcia, P. van Rossum, R. Verdult, and R. W. Schreur, "Wirelessly pickpocketing a MIFARE Classic card," in *IEEE Symposium on Security and Privacy*, 2009, pp. 3–15.

[6] C. Ming-Yang, S. Jie-Ren, Y. Bo-Yin, D. Jintai, and C. Chen-Mou, in *Cryptology and Information Security Conference 2010*.

[7] "COPACOBANA — special-purpose hardware for code-breaking," http://www.copacobana.org.

[8] K. Finkenzeller, *RFID handbook: Fundamentals and applications in contactless smart cards and identification*, 2nd ed. Wiley and Sons Ltd, 2003.

[9] *ISO/IEC 14443. identification cards — contactless integrated circuit(s) cards — proximity card*, 2001.

[10] D. L. Cook, J. Ioannidis, A. D. Keromytis, and J. Luck, "CryptoGraphics: Secret key cryptography using graphics cards," in *RSA Conference, Cryptographer's Track (CT-RSA)*, 2005, pp. 334–350.

[11] D. L. Cook and A. D. Keromytis, *CryptoGraphics: Exploiting Graphics Cards For Security*. Springer, 2006.

[12] A. Moss and N. P. Smart, "Toward acceleration of RSA using 3D graphics hardware," in *11th IMA International Conference on Cryptography and Coding*, December 2007.

[13] D. Bernstein, T.-R. Chen, C.-M. Cheng, T. Lange, and B.-Y. Yang, "ECM on graphics cards," in *Advances in Cryptology — EUROCRYPT 2009*, pp. 483–501.

[14] D. V. Bailey, L. Batina, D. J. Bernstein, P. Birkner, J. W. Bos, H.-C. Chen, C.-M. Cheng, G. van Damme, G. de Meulenaer, L. J. D. Perez, J. Fan, T. Güneysu, F. Gurkaynak, T. Kleinjung, T. Lange, N. Mentens, R. Niederhagen, C. Paar, F. Regazzoni, P. Schwabe, L. Uhsadel, A. Van Herrewege, and B.-Y. Yang. (2009, Nov) Breaking ECC2K-130. Cryptology ePrint Archive, Report 2009/541.

[15] "GNU radio," http://gnuradio.org.

[16] *MF1ICS50 Functional specification*, NXP, December 2009.

[17] *MF1ICS70 Functional specification*, NXP, December 2009.

[18] Ettus Research, "The universal software radio peripheral," http://www.ettus.com.

[19] "Proxmark III, a test instrument for HF/LF RFID," http://proxmark3.com.

[20] D. J. Bernstein, J. Buchmann, and E. Dahmen, Eds., *Post-Quantum Cryptography*. Springer-Verlag Berlin, February 2009, iSBN: 978-3-540-88701-0, e-ISBN: 978-3-540-88702-7.

[21] A. Kipnis, J. Patarin, and L. Goubin, "Unbalanced Oil and Vinegar signature schemes," in *Advances in Cryptology — EUROCRYPT 1999*, pp. 206–222.

[22] A. I.-T. Chen, C.-H. O. Chen, M.-S. Chen, C.-M. Cheng, and B.-Y. Yang, "Practical-sized instances of multivariate PKCs: Rainbow, TTS, and ℓIC-derivatives," in *PQCrypto*, 2008, pp. 95–108.

[23] B.-Y. Yang, C.-M. Cheng, B.-R. Chen, and J.-M. Chen, "Implementing minimized multivariate public-key cryptosystems on low-resource embedded systems," in *SPC 2006*, pp. 73–88.

[24] A. Bogdanov, T. Eisenbarth, A. Rupp, and C. Wolf, "Time-area optimized public-key engines: Cryptosystems as replacement for elliptic curves?" in *Cryptographic Hardware and Embedded Systems — CHES 2008*, pp. 45–61.