

## Wirelessly Pickpocketing a Mifare Classic Card

Flavio D. Garcia    Peter van Rossum    Roel Verdult    Ronny Wichers Schreur

Radboud University Nijmegen, The Netherlands

{flaviog,petervr,rverdult,ronny}@cs.ru.nl

### Abstract

*The Mifare Classic is the most widely used contactless smartcard on the market. The stream cipher CRYPTO1 used by the Classic has recently been reverse engineered and serious attacks have been proposed. The most serious of them retrieves a secret key in under a second. In order to clone a card, previously proposed attacks require that the adversary either has access to an eavesdropped communication session or executes a message-by-message man-in-the-middle attack between the victim and a legitimate reader. Although this is already disastrous from a cryptographic point of view, system integrators maintain that these attacks cannot be performed undetected.*

*This paper proposes four attacks that can be executed by an adversary having only wireless access to just a card (and not to a legitimate reader). The most serious of them recovers a secret key in less than a second on ordinary hardware. Besides the cryptographic weaknesses, we exploit other weaknesses in the protocol stack. A vulnerability in the computation of parity bits allows an adversary to establish a side channel. Another vulnerability regarding nested authentications provides enough plaintext for a speedy known-plaintext attack.*

### 1. Introduction

With more than one billion cards sold, the Mifare Classic covers more than 70% of the contactless smartcard market<sup>1</sup>. Such cards contain a slightly more powerful IC than classical RFID chips (developed for identification only), equipping them with modest computational power and making them suitable for applications beyond identification, such as access control and ticketing systems.

The Mifare Classic is widely used in public transport payment systems such as the Oyster card<sup>2</sup> in London,

1. <http://www.nxp.com>

2. <http://oyster.tfl.gov.uk>

the Charlie Card in Boston<sup>3</sup>, the SmartRider in Australia<sup>4</sup>, EasyCard in Taiwan<sup>5</sup>, and the OV-chipkaart<sup>6</sup> in The Netherlands. It is also widely used for access control in office and governmental buildings and military objects.

According to [MFS08] the Mifare Classic complies with parts 1 to 3 of the ISO standard 14443-A [ISO01], specifying the physical characteristics, the radio frequency interface, and the anti-collision protocol. The Mifare Classic does not implement part 4 of the standard, describing the transmission protocol, but instead uses its own secure communication layer. In this layer, the Mifare Classic uses the proprietary stream cipher CRYPTO1 to provide data confidentiality and mutual authentication between card and reader. This cipher has recently been reversed engineered [NESP08], [GKM<sup>+</sup>08].

In this paper, we show serious vulnerabilities of the Mifare Classic that enable an attacker to retrieve all cryptographic keys of a card, just by wirelessly communicating with it. Thus, the potential impact is much larger than that of the problems previously reported in [GKM<sup>+</sup>08], [CNO08], [KHG08], [Noh08], where the attacker either needs to have access to a legitimate reader or an eavesdropped communication session. The attacks described in this paper are fast enough to allow an attacker to wirelessly ‘pickpocket’ a victim’s Mifare Classic card, i.e., to clone it immediately.

**Vulnerabilities.** The vulnerabilities we discovered concern the handling of parity bits and nested authentications.

- The Mifare Classic sends a parity bit for each byte that is transmitted. Violating the standard, the Mifare Classic mixes the data link layer and secure communication layer: parity bits are computed over the plaintext instead of over the

3. [http://www.mbta.com/fares\\_and\\_passes/charlie](http://www.mbta.com/fares_and_passes/charlie)

4. <http://www.transperth.wa.gov.au>

5. <http://www.easycard.com.tw>

6. <http://www.ov-chipkaart.nl>

bits that are actually sent, i.e., the ciphertext. This is, in fact, authenticate-then-encrypt which is generically insecure [Kra01].

Furthermore, parity bits are encrypted with the same bit of keystream that encrypts the first bit of the next byte of plaintext. During the authentication protocol, if the reader sends wrong parity bits, the card stops communicating. However, if the reader sends correct parity bits, but wrong authentication data, the card responds with an (encrypted) error code. This breaks the confidentiality of the cipher, enabling an attacker to establish a side channel.

- The memory of the Mifare Classic is divided into sectors, each of them having its own 48-bit secret key. To perform an operation on a specific sector, the reader must first authenticate using the corresponding key. When an attacker has already authenticated for one sector (knowing the key for that sector) and subsequently attempts to authenticate for another sector (without knowing the key for this sector), that attempt leaks 32 bits of information about the secret key of that sector.

**Attacks.** We describe four attacks exploiting these vulnerabilities to recover the cryptographic keys from a Mifare Classic card having only contactless communication with it (and not with a legitimate reader). These attacks make different trade-offs between online communication time (the time an attacker needs to communicate with a card), offline computation time (the time it takes to compute the cryptographic key using the data gathered from the card), precomputation time (one-time generation time of static tables), disk space usage (of the static tables) and special assumptions (whether the attacker has already one sector key or not).

- The first attack exploits the weakness of the parity bits to mount an offline brute-force attack on the 48-bit key space. The attacker only needs to try to authenticate approximately 1500 times (which takes under a second).
- The second attack also exploits the weakness of the parity bits but this time the attacker mounts an adaptive chosen ciphertext attack. The attacker needs approximately 28500 authentication attempts. In this attack, she needs to make sure that the challenge nonce of the card is constant, which is why this takes approximately fifteen minutes. During these authentication attempts, the attacker adaptively chooses her challenge to the card, ultimately obtaining a challenge that guarantees that there are only 436 possibilities

for the odd-numbered bits of the internal state of the cipher. This reduces the offline search space to approximately 33 bits. On a standard desktop computer this search takes about one minute.

- In the third attack the attacker keeps her own challenge constant, but varies the challenge of the tag, again ultimately obtaining a special internal state of the cipher. These special states have to be precomputed and stored in a 384 GB table. This attack requires on average  $2^{12} = 4096$  authentication attempts, which could in principle be done in about two minutes. A few extra authentication attempts allow efficient lookup in the table.
- The fourth attack assumes that the attacker has already recovered at least one sector key. When the attacker first authenticates for this sector and then for another sector, the authentication protocol is slightly different, viz., the challenge nonce of the tag is not sent in the clear, but encrypted with the key of the new sector. Because the random number generator has only a 16-bit state, because parity bits leak three bits of information, and because the tag's random number generator runs in sync with the communication timing, this allows an attacker to guess the plaintext tag nonce and hence 32 bits of keystream. Due to weaknesses in the cipher [GKM<sup>+</sup>08], we can use these 32 bits of keystream to compute approximately  $2^{16}$  candidate keys. These can then be checked offline using another authentication attempt. Since this attack only requires three authentication attempts, the online time is negligible. The offline search takes under a second on ordinary hardware.

**Related work.** De Koning Gans et al. [KHG08] have proposed an attack on a Mifare Classic tag that exploits the malleability of the CRYPTO1 stream cipher to read partial information from a tag, without even knowing the encryption algorithm. By slicing a Mifare Classic chip and taking pictures with a microscope, the cipher was reverse engineered by Nohl et al. [NESP08]. Courtois et al. claim in [CNO08] that the CRYPTO1 cipher is susceptible to algebraic attacks and Nohl shows a statistical weakness of the cipher in [Noh08]. A full description of the cipher was given by Garcia et al. in [GKM<sup>+</sup>08], together with a reverse engineered authentication protocol. They also describe an attack with which an attacker can recover a sector key by communicating with a genuine reader or by eavesdropping a successful authentication.

All attacks described in these papers have in common that they need access to a legitimate reader or intercepted communication. In contrast, the attacks

described in our paper only need access to a card.

**Impact.** The implications of the attacks described in this paper are vast.

Many ticketing and payment systems using the Mifare Classic sequentially authenticate for several sectors verifying the data in the card. In case of invalid data, the protocol aborts. With previous attacks, this means that an attacker has to either eavesdrop a full trace or walk from the reader to the card holder several times, executing a message-by-message man-in-the-middle attack. In practice, both options are hard to accomplish undetected. Furthermore, there is no guarantee that this allows an attacker to recover all useful data in the card, since some sectors might not be read in this particular instance. Our attacks always enable an attacker to retrieve all data from the card.

Our fourth attack, where the attacker already knows a single key, is extremely fast (less than one second per key on ordinary hardware). The first key can be retrieved using one of our first three attacks, but in many situations this is not even necessary. Most deployed systems leave default keys for unused sectors or do not diversify keys at all. Nearly all deployed systems that do diversify have at least one sector key that is not diversified, namely for storing the diversification information. This is even specified in NXP’s guideline for system integrators [MAD07]. This means that it is possible for an adversary to recover all keys necessary to read and write the sixteen sectors of a Mifare Classic 1k tag in less than sixteen seconds.

**Overview.** We start by gathering the relevant information that is already known about the Mifare Classic in Section 2: its logical structure, the encryption algorithm, the authentication protocol and the initialization of the stream cipher, how to undo the initialization of the stream cipher, and information about how the tag generates its random numbers. In Section 3, we continue with a precise description of the discovered weaknesses in the handling of the parity bits and nested authentications. In Section 4, we show how these weaknesses can be exploited to recover a sector key by communication with just a card. Section 5 gives some concluding remarks.

## 2. Background

### 2.1. Communication

The physical layer and data link layer of the Mifare family of cards are described in the ISO standard

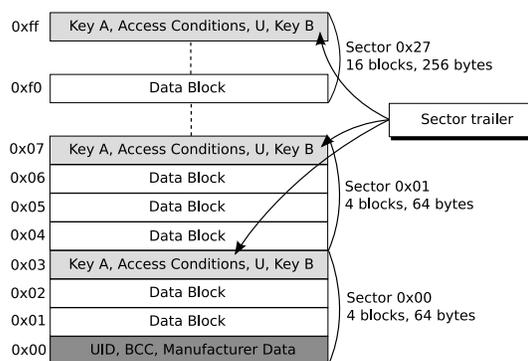


Figure 2.1. Memory layout of the Mifare Classic

14443-A. We have used the Proxmark III<sup>7</sup> for communication; this device implements, among others, these two layers of this standard and can emulate both a card and a reader.

Using information from [KHG08] about the command codes of the Mifare Classic and from [GKM<sup>+</sup>08], [NESPO8] about the cryptographic aspects of the Mifare Classic, we implemented the functionality of a Mifare Classic reader on the Proxmark. Note that we can observe a tag’s communication at the data link level, implying that we can observe the parity bits as well. Furthermore, we have the freedom to send arbitrary parity bits, which is not possible using stock commercial Mifare Classic readers. However, many newer NFC readers can be used to communicate with a Mifare Classic card as well and are capable of sending and receiving arbitrary parity bits.<sup>8</sup> We have also executed the attacks described in this paper using an inexpensive (30 USD) stock commercial NFC reader. However, these readers are typically connected to a host PC using USB and it is harder to obtain accurate communication timing.

### 2.2. Memory structure of the Mifare Classic

The Mifare Classic tag is essentially a memory chip with secure wireless communication capabilities. The memory of the tag is divided into sectors, each of which is further divided into blocks of sixteen bytes each. The last block of each sector is the sector trailer and stores two secret keys and the access conditions for that sector.

To perform an operation on a specific block, the reader must first authenticate for the sector containing

7. <http://www.proxmark.org/>

8. <http://www.libnfc.org/>

that block. The access conditions determine which of the two keys must be used. See Figure 2.1 for an overview of the memory of a Mifare Classic tag.

### 2.3. CRYPTO1

After authentication, the communication between tag and reader is encrypted with the CRYPTO1 stream cipher. This cipher consists of a 48-bit linear feedback shift register (LFSR) with generating polynomial  $x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1$  and a non-linear filter function  $f$  [NESP08]. Each clock tick, twenty bits of the LFSR are put through the filter function, generating one bit of keystream. Then the LFSR shifts one bit to the left, using the generating polynomial to generate a new bit on the right. See Figure 2.2 for a schematic representation.

We let  $\mathbb{F}_2 = \{0, 1\}$  the field of two elements (or the set of Booleans). The symbol  $\oplus$  denotes addition (XOR).

**Definition 2.1.** The feedback function  $L: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$  is defined by  $L(x_0x_1 \dots x_{47}) := x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43}$ .

The specifics of the filter function are taken from [GKM<sup>+</sup>08].

**Definition 2.2.** The filter function  $f: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$  is defined by

$$\begin{aligned} f(x_0x_1 \dots x_{47}) &:= f_c(f_a(x_9, x_{11}, x_{13}, x_{15}), \\ &f_b(x_{17}, x_{19}, x_{21}, x_{23}), f_b(x_{25}, x_{27}, x_{29}, x_{31}), \\ &f_a(x_{33}, x_{35}, x_{37}, x_{39}), f_b(x_{41}, x_{43}, x_{45}, x_{47})). \end{aligned}$$

Here  $f_a, f_b: \mathbb{F}_2^4 \rightarrow \mathbb{F}_2$  and  $f_c: \mathbb{F}_2^5 \rightarrow \mathbb{F}_2$  are defined by  $f_a(y_0, y_1, y_2, y_3) := ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))$ ,  $f_b(y_0, y_1, y_2, y_3) := ((y_0 \wedge y_1) \vee y_2) \oplus ((y_0 \oplus y_1) \wedge (y_2 \vee y_3))$ , and  $f_c(y_0, y_1, y_2, y_3, y_4) := (y_0 \vee ((y_1 \vee y_4) \wedge (y_3 \oplus y_4))) \oplus ((y_0 \oplus (y_1 \wedge y_3)) \wedge ((y_2 \oplus y_3) \vee (y_1 \wedge y_4)))$ . Because  $f(x_0x_1 \dots x_{47})$  only depends on  $x_9, x_{11}, \dots, x_{47}$ , we shall overload notation and see  $f$  as a function  $\mathbb{F}_2^{20} \rightarrow \mathbb{F}_2$ , writing  $f(x_0x_1 \dots x_{47})$  as  $f(x_9, x_{11}, \dots, x_{47})$ .

Note that  $f_a$  and  $f_b$  here are negated when compared to [GKM<sup>+</sup>08] and  $f_c$  is changed accordingly. The expressions for  $f_a$ ,  $f_b$ , and  $f_c$  given here have the minimal number of logical operators in  $\{\wedge, \vee, \oplus, \neg\}$ ; in practice, this allows for a fast bitsliced implementation of  $f$  [Bih97].

For future reference, note that each of the building blocks of  $f$  (and hence  $f$  itself) have the property that

it gives zero for half of the possible inputs (respectively one).

**Theorem 2.3.** Let  $Y_0, Y_1, \dots, Y_4$  be independent uniformly distributed variables over  $\mathbb{F}_2$ . Then

$$\begin{aligned} P[f_a(Y_0, Y_1, Y_2, Y_3) = 0] &= 1/2 \\ P[f_b(Y_0, Y_1, Y_2, Y_3) = 0] &= 1/2 \\ P[f_c(Y_0, Y_1, Y_2, Y_3, Y_4) = 0] &= 1/2. \end{aligned}$$

**Proof.** By inspection.  $\square$

### 2.4. Tag nonces

For use in the authentication protocol, described in Section 2.5 below, Mifare Classic tags possess a pseudo-random generator. In [NP07] it was revealed that the 32-bit tag nonces are generated by a 16-bit LFSR with generating polynomial  $x^{16} + x^{14} + x^{13} + x^{11} + 1$ . Every clock tick the LFSR shifts to the left and the feedback bit is computed using  $L_{16}$ .

**Definition 2.4.** The feedback function  $L_{16}: \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2$  of the pseudo-random generator is defined by

$$L_{16}(x_0x_1 \dots x_{15}) := x_0 \oplus x_2 \oplus x_3 \oplus x_5.$$

Let us define the function  $\text{suc}$  that computes the next 32-bit LFSR sequence of the 16-bit LFSR. This function is used later on in Section 2.5 in the authentication protocol.

**Definition 2.5.** The successor function  $\text{suc}: \mathbb{F}_2^{32} \rightarrow \mathbb{F}_2^{32}$  is defined by

$$\text{suc}(x_0x_1 \dots x_{31}) := x_1x_2 \dots x_{31}L_{16}(x_{16}x_{17} \dots x_{31}).$$

Because the period of the pseudo-random generator is only 65535 and because it shifts every  $9.44\mu\text{s}$ , it cycles in  $618\text{ms}$ .

Under similar physical conditions (i.e., do not move the tag or the reader), the challenge nonce that the tag generates only depends on the time between the moment the reader switches on the electromagnetic field and the moment it sends the authentication request. In practice, this means that an attacker who has physical control of the tag, can get the tag to send the same nonce every time. To do so, the attacker just has to drop the field (for approximately  $30\mu\text{s}$ ) to discharge all capacitors in the tag, switch the field back on, and wait for a constant amount of time before authenticating.

Alternatively, by waiting exactly the right amount of time before authenticating again, the attacker can control the challenge nonce that the tag will send. This works whenever the tag does not leave the electromagnetic field in the mean time. On average, this takes  $618\text{ms}/2 = 309\text{ms}$ .

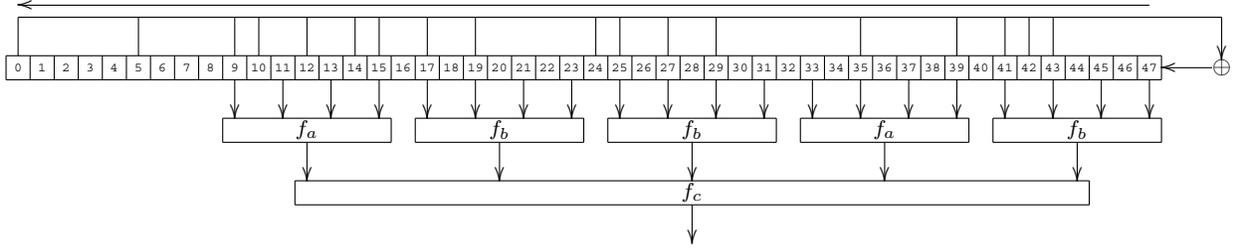


Figure 2.2. Structure of the CRYPTO1 stream cipher

## 2.5. Authentication protocol and initialization

The authentication protocol was reverse engineered in [GKM<sup>+</sup>08]. During the anti-collision phase, the tag sends its uid  $u$  to the reader. The reader then asks to authenticate for a specific sector. The tag sends a challenge  $n_T$ . From this point on, communication is encrypted, i.e., XOR-ed with the keystream. The reader responds with its own challenge  $n_R$  and the answer  $a_R := \text{suc}^{64}(n_T)$  to the challenge of the tag; the tag finishes with its answer  $a_T := \text{suc}^{96}(n_T)$  to the challenge of the reader. See Figure 2.3. Note that later on we will send messages  $a_R$  that deviate from this protocol; this will be explained in Section 4.

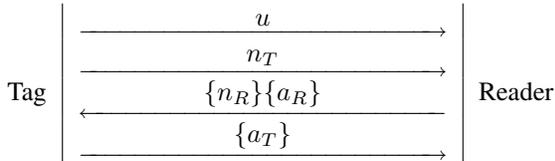


Figure 2.3. Authentication protocol

During the authentication protocol, the internal state of the stream cipher is initialized. It starts out as the sector key  $k$ , then  $n_T \oplus u$  is shifted in, then  $n_R$  is shifted in. Because communication is encrypted from  $n_R$  onwards, the encryption of the later bits of  $n_R$  is influenced by the earlier bits of  $n_R$ . Authentication is achieved by reaching the same internal state of the cipher after shifting in  $n_R$ .

The following precisely defines the initialization of the cipher and the generation of the LFSR-stream  $a_0a_1\dots$  and the keystream  $b_0b_1\dots$ .

**Definition 2.6.** Given a key  $k = k_0k_1\dots k_{47} \in \mathbb{F}_2^{48}$ , a tag nonce  $n_T = n_{T,0}n_{T,1}\dots n_{T,31} \in \mathbb{F}_2^{32}$ , a uid  $u = u_0u_1\dots u_{31} \in \mathbb{F}_2^{32}$ , and a reader nonce  $n_R = n_{R,0}n_{R,1}\dots n_{R,31} \in \mathbb{F}_2^{32}$ , the internal state of the cipher at time  $i$  is  $\alpha_i := a_i a_{i+1} \dots a_{i+47} \in \mathbb{F}_2^{48}$ .

Here the  $a_i \in \mathbb{F}_2$  are given by

$$\begin{aligned} a_i &:= k_i & \forall i \in [0, 47] \\ a_{48+i} &:= L(a_i, \dots, a_{47+i}) \oplus n_{T,i} \oplus u_i & \forall i \in [0, 31] \\ a_{80+i} &:= L(a_{32+i}, \dots, a_{79+i}) \oplus n_{R,i} & \forall i \in [0, 31] \\ a_{112+i} &:= L(a_{64+i}, \dots, a_{111+i}) & \forall i \in \mathbb{N}. \end{aligned}$$

Furthermore, we define the keystream bit  $b_i \in \mathbb{F}_2$  at time  $i$  by

$$b_i := f(a_i a_{i+1} \dots a_{47+i}) \quad \forall i \in \mathbb{N}.$$

We denote encryptions by  $\{-\}$  and define  $\{n_{R,i}\}, \{a_{R,i}\} \in \mathbb{F}_2$  by

$$\begin{aligned} \{n_{R,i}\} &:= n_{R,i} \oplus b_{32+i} & \forall i \in [0, 31] \\ \{a_{R,i}\} &:= a_{R,i} \oplus b_{64+i} & \forall i \in [0, 31]. \end{aligned}$$

Note that the  $a_i$ ,  $\alpha_i$ ,  $b_i$ ,  $\{n_{R,i}\}$ , and  $\{a_{R,i}\}$  are formally functions of  $k$ ,  $n_T$ ,  $u$ , and  $n_R$ . Instead of making this explicit by writing, e.g.,  $a_i(k, n_T, u, n_R)$ , we just write  $a_i$  where  $k$ ,  $n_T$ ,  $u$ , and  $n_R$  are clear from the context.

## 2.6. Rollback

For our attacks it is important to realize that to recover the key, it is sufficient to learn the internal state of the cipher  $\alpha_i$  at any point  $i$  in time. Since an attacker knows  $u$ ,  $n_T$ , and  $\{n_R\}$ , the LFSR can then be rolled back to time zero. This is explained in Section 6.2 of [GKM<sup>+</sup>08]; below we show their method translated into our notation.

**Definition 2.7.** The rollback function  $R: \mathbb{F}_2^{48} \rightarrow \mathbb{F}_2$  is defined by  $R(x_1x_2\dots x_{48}) := x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{48}$ .

If one first shifts the LFSR left using  $L$  to generate a new bit on the right, then  $R$  recovers the bit that dropped out on the left, i.e.,

$$R(x_1x_2\dots x_{47} L(x_0x_1\dots x_{47})) = x_0. \quad (1)$$

**Theorem 2.8.** *In the situation from Definition 2.6, we have*

$$\begin{aligned} a_{64+i} &= R(a_{65+i} \dots a_{112+i}) & \forall i \in \mathbb{N} \\ a_{32+i} &= R(a_{33+i} \dots a_{80+i}) \oplus \{n_{R,i}\} \oplus \\ & \quad f(0 \ a_{33+i} \dots a_{79+i}) & \forall i \in [0, 31] \\ a_i &= R(a_{1+i} \dots a_{48+i}) \oplus n_{T,i} \oplus u_i & \forall i \in [0, 31]. \end{aligned}$$

**Proof.** Straightforward, using Definition 2.6 and Equation (1). For the second equation, note that  $f$  does not depend on its leftmost input. Therefore  $f(0 \ a_{33+i} \dots a_{79+i}) = f(a_{32+i} \dots a_{79+i}) = b_{32+i}$  and hence  $\{n_{R,i}\} \oplus f(0 \ a_{33+i} \dots a_{79+i}) = n_{R,i}$ .  $\square$

Consequently, if an attacker somehow recovers the internal state of the LFSR  $\alpha_i = a_i a_{i+1} \dots a_{i+47}$  at some time  $i$ , then she can repeatedly apply Theorem 2.8 to recover  $\alpha_0 = a_0 a_1 \dots a_{47}$ , which is the sector key.

### 3. Weaknesses

This section describes weaknesses in the design of the Mifare Classic. We first treat weaknesses in the way the Mifare Classic handles parity bits and then the ones concerning nested authentications. These weaknesses will be exploited in Section 4.

#### 3.1. Parity weaknesses

The ISO standard 14443-A [ISO01] specifies that every byte sent is followed by a parity bit. The Mifare Classic computes parity bits over the plaintext instead of over the ciphertext. Additionally, the bit of keystream used to encrypt the parity bits is reused to encrypt the next bit of plaintext.

This already breaks the confidentiality of the encryption scheme. In this paper we shall only be concerned with the four parity bits of  $n_T$ ,  $n_R$ , and  $a_R$ . The ISO standard specifies odd parity, hence the “ $\oplus 1$ ” in the definition below.

**Definition 3.1.** In the situation from Definition 2.6, we define the parity bits  $p_j \in \mathbb{F}_2$  by

$$\begin{aligned} p_j &:= n_{T,8j} \oplus n_{T,8j+1} \oplus \dots \oplus n_{T,8j+7} \oplus 1 \\ p_{j+4} &:= n_{R,8j} \oplus n_{R,8j+1} \oplus \dots \oplus n_{R,8j+7} \oplus 1 \\ p_{j+8} &:= a_{R,8j} \oplus a_{R,8j+1} \oplus \dots \oplus a_{R,8j+7} \oplus 1 \\ & \quad \forall j \in [0, 3] \end{aligned}$$

and the encryptions  $\{p_j\}$  of these by

$$\{p_j\} := p_j \oplus b_{8+8j} \quad \forall j \in [0, 11].$$

There is a further weakness concerning the parity bits. During the authentication protocol, when the

reader sends  $\{n_R\}$  and  $\{a_R\}$ , the tag checks the parity bits before the answer of the reader. If at least one of the eight parity bits is wrong, the tag does not respond. If all eight parity bits are correct, but the answer  $a_R$  is wrong, the tag responds with the 4-bit error code  $0 \times 5$  signifying failed authentication, called ‘transmission error’ in [KHG08]. If all eight parity bits are correct and the answer  $a_R$  is also correct, the tag responds, of course, with its answer  $a_T$ . Furthermore, in case the reader sends the correct parity, but the wrong answer, the 4-bit error code  $0 \times 5$  is sent encrypted. This happens even though the reader has not authenticated itself and hence cannot be assumed to be able to decrypt.

Figure 3.1 shows an authentication trace where the attacker sends incorrect authentication data but correct parity bits. The exclamation marks represent parity bits that deviate from what is specified in the standard. The final message of this trace is the encrypted error message  $0 \times 5$ .

#### 3.2. Nested authentications

Once an attacker knows a single sector key of a Mifare Classic, there is a vulnerability that allows an adversary to recover more keys. When a reader is already communicating (encrypted) with a tag, a subsequent authentication command for a new sector also has to be sent encrypted. After this authentication command, the internal state of the cipher is set to the key for the new sector and the authentication protocol from Section 2.5 starts again. This time, however, the challenge of the tag is also sent encrypted. Because there are only  $2^{16}$  possible nonces, an attacker can simply try to guess a nonce to recover 32 bits of keystream.

Also here, the information that leaks through the parity bits can be used to speed up the attack. Although there are  $2^{16}$  tag nonces, we show below that the parity bits sent with the encrypted tag nonce leak three bits of information, so that there are only  $2^{13}$  tag nonces possible.

**Definition 3.2.** In the situation from Definition 2.6, we define  $\{n_{T,i}\} \in \mathbb{F}_2$  by

$$\{n_{T,i}\} := n_{T,i} \oplus b_i \quad \forall i \in [0, 31].$$

**Theorem 3.3.** *For every  $j \in \{0, 1, 2\}$  we have*

$$\begin{aligned} n_{T,8j} \oplus n_{T,8j+1} \oplus \dots \oplus n_{T,8j+7} \oplus n_{T,8j+8} \\ = \{p_j\} \oplus \{n_{T,8j+8}\} \oplus 1 \end{aligned}$$

Reader	26	req type A
Tag	02 00	answer req
Reader	93 20	select
Tag	c1 08 41 6a e2	uid, bcc
Reader	93 70 c1 08 41 6a e2 e4 7c	select(uid)
Tag	18 37 cd	Mifare Classic 4k
Reader	60 00 f5 7b	auth(block 0)
Tag	ab cd 19 49	$n_T$
Reader	59! d5 92 0f! 15 b9 d5! 53!	$\{n_R\}\{a_R\}$
Tag	a	$\{5\}$

Figure 3.1. Trace of a failed authentication attempt

**Proof.** We compute as follows.

$$\begin{aligned}
& n_{T,8j} \oplus n_{T,8j+1} \oplus \dots \oplus n_{T,8j+7} \oplus n_{T,8j+8} \\
&= p_j \oplus 1 \oplus n_{T,8j+8} \quad (\text{by Dfn. 3.1}) \\
&= p_j \oplus b_{8+8j} \oplus n_{T,8j+8} \oplus b_{8+8j} \oplus 1 \\
&= \{p_j\} \oplus \{n_{T,8j+8}\} \oplus 1 \quad (\text{by Dfns. 3.1 and 3.2})
\end{aligned}$$

□

Since the attacker can observe  $\{p_j\}$  and  $\{n_{T,8j+8}\}$ , this theorem gives an attacker three bits of information about  $n_T$ .

In practice, timing information between the first and second authentication attempt leaks so much additional information that the attacker can accurately predict what the challenge nonce will be.

It turns out that the distance between the tag nonces used in consecutive authentication attempts strongly depends on the time between those attempts. Here distance is defined as follows.

**Definition 3.4.** Let  $n_T$  and  $n'_T$  be two tag nonces. We define the distance between  $n_T$  and  $n'_T$  as

$$d(n_T, n'_T) := \min_{i \in \mathbb{N}} \text{suc}^i(n_T) = n'_T.$$

## 4. Attacks

This section shows how the weaknesses described in the previous section can be exploited.

### 4.1. Brute-force attack

The attacker plays the role of a reader and tries to authenticate for a sector of her choice. She answers the challenge of the tag with eight random bytes (and eight random parity bits) for  $\{n_R\}$  and  $\{a_R\}$ . With probability  $1/256$ , the parity bits are correct and the tag responds with the encrypted 4-bit error code. A success leaks 12 bits of entropy (out of 48).

Repeating the above procedure sufficiently many times (in practice six is enough) uniquely determines the key. Since the key length is only 48 bits, the attacker can now brute force the key: she can just check which of the  $2^{48}$  keys produces all six times the correct parity bits and received response. In practice, gathering those six authentication sessions with correct parity bits only takes on average  $6 \cdot 256 = 1536$  authentication attempts which can be done in less than one second. The time it takes to perform the offline brute-force attack of course is strongly dependent on the resources the attacker has at her disposal. We give an estimate based on the performance of COPACOBANA [KPP<sup>+</sup>06]; this is a code-cracker built from off-the-shelf hardware costing approximately 10000 USD. Based on the fact that COPACOBANA finds a 56-bit DES key in on average 6.4 days, pessimistically assuming that one can fit the same number of CRYPTO1 checks on an FPGA as DES-decryptions, and realizing that the search space is a factor of 256 smaller, we estimate that this takes on average  $6.4 \text{ days}/256 = 36 \text{ min}$ .

In Sections 4.2 and 4.3 the same idea is exploited in a different way, trading online communication for computation time.

### 4.2. Varying the reader nonce

This section shows how an attacker can mount a chosen ciphertext attack by adaptively varying the encryption of  $n_R$ . We assume that the attacker can control the power up timing of the tag, thereby causing the tag to produce the same  $n_T$  every time.

We first give the idea of the attack. The attacker runs authentication sessions until she guesses the correct parity bits. The internal state of the stream cipher just after feeding in  $n_R$  is  $\alpha_{64}$ . She then runs another authentication session, keeping the first 31 bits of  $\{n_R\}$  (and the three parity bits) the same, flipping the last

bit of  $\{n_R\}$  (and randomly picking the rest until the parity is ok). Now the state of the stream cipher just after feeding in the reader nonce is  $\alpha_{64} \oplus 1$ , i.e.,  $\alpha_{64}$  with the last bit flipped. Since the parity of the last byte of  $n_R$  changed (since the attacker flipped just the last bit), and since its parity in the first run is encrypted with  $f(\alpha_{64})$  and in the second run with  $f(\alpha_{64} \oplus 1)$ , she can deduce whether or not the last bit of  $n_R$  influences the encryption of the next bit, i.e., whether or not  $f(\alpha_{64}) = f(\alpha_{64} \oplus 1)$ . Approx. 9.4% of the possible  $\alpha_{64}$ 's has  $f(\alpha_{64}) \neq f(\alpha_{64} \oplus 1)$  and they can easily be generated since only the twenty bits that are input to  $f$  are relevant. By repeating this, the attacker eventually (on average after 10.6 tries) finds an instance in which  $\alpha_{64}$  is in those 9.4% and then she only has to search, offline, 9.4% of all possible states.

We now make this idea precise and at the same time generalize it to the last bit of each of the four bytes in the reader nonce. The following definition says that a reader nonce has property  $F_j$  (for  $j \in \{0, 1, 2, 3\}$ ) if flipping the last bit of the  $(j+1)$ th byte of the reader nonce changes the encryption of the next bit.

**Definition 4.1.** Let  $j \in \{0, 1, 2, 3\}$  and let  $n_R$  and  $n'_R$  be reader nonces with the property that  $n'_{R,8j+7} = \overline{n_{R,8j+7}}$  and  $n'_{R,i} = n_{R,i}$  for all  $i < 8j+7$  (and no restrictions on  $n_{R,i}$  and  $n'_{R,i}$  for  $i > 8j+7$ ). We say that  $n_R$  has property  $F_j$  if  $b_{8j+40} \neq b'_{8j+40}$ .

Formally this is not just a property of  $n_R$ , but also of  $k$ ,  $n_T$ , and  $u$ . Now  $k$  and  $u$  of course do not vary, so we ignore that here. Furthermore, when deciding whether or not  $n_R$  has property  $F_j$  in Protocol 4.2 below, the attacker also keeps  $n_T$  constant.

The attacker does change the reader nonce. We use  $a'_i$  to refer to the bits of the LFSR-stream where the reader nonce  $n'_R$  is used and similarly for  $\alpha'_i, b'_i$ , etc. I.e.,  $a'_i$  denotes  $a_i(k, n_T, n'_R)$ .

Note that  $\alpha_{8j+40}$  (resp.  $\alpha'_{8j+40}$ ) is the internal state of the cipher just after feeding in  $(j+1)$ th byte of  $n_R$  (resp.  $n'_R$ ) and  $b_{8j+40} = f(\alpha_{8j+40})$  (resp.  $b'_{8j+40} = f(\alpha'_{8j+40})$ ), so that  $F_j$  does not depend on  $n_{R,i}$  and  $n'_{R,i}$  for  $i > 8j+7$ . Also observe that  $\alpha'_{8j+40} = a_{8j+40} \dots a_{8j+86} a'_{8j+87}$ , i.e.,  $\alpha_{8j+40}$  and  $\alpha'_{8j+40}$  only differ in the last position.

The crucial idea is that an attacker can decide whether or not  $n_R$  has property  $F_j$ , only knowing  $\{n_R\}$ . (In practice, the attacker of course chooses  $\{n_R\}$ .)

**Protocol 4.2.** Given  $\{n_R\}$ , an attacker can decide as follows whether or not  $n_R$  has property  $F_j$ . She first chooses  $\{a_R\}$  arbitrary. She then starts, consecutively, several authentication sessions with the tag.

After the tags sends its challenge  $n_T$ , the attacker answers  $\{n_R\}, \{a_R\}$ . Inside this answer, the attacker also has to send the (encryptions of) the parity bits:  $\{p_4\}, \dots, \{p_{11}\}$ . For these, she tries all 256 possibilities. After on average 128 authentication sessions, and after at most 256, with different choices for the  $\{p_i\}$ , the parity bits are correct and the attacker recognizes this because the tag responds with an error code.

Now the attacker defines  $\{n'_{R,8j+7}\} := \overline{\{n_{R,8j+7}\}}$ , i.e., she changes the last bit of the  $j$ th byte of  $\{n_R\}$ . The earlier bits of  $\{n'_R\}$  she chooses the same as those of  $\{n_R\}$ ; the later bits of  $\{n'_R\}$  and  $\{a'_R\}$  the attacker chooses arbitrarily. Again, the attacker repeatedly tries to authenticate to find the correct parity bits  $\{p'_i\}$  to send. Note that necessarily  $\{p'_i\} = \{p_i\}$  for  $i \in \{4, \dots, j+3\}$ , so this takes on average  $2^{7-j}$  authentication attempts and at most  $2^{8-j}$ .

Now  $n_R$  has property  $F_j$  if and only if  $\{p_{j+4}\} \neq \{p'_{j+4}\}$ .

**Proof.** Because the attacker modified the ciphertext of the last bit of the  $j$ th byte of  $n_R$ , the last bit of the plaintext of this byte also changes:  $n'_{R,8j+7} = \{n'_{R,8j+7}\} \oplus b'_{8j+39} = \{n'_{R,8j+7}\} \oplus b'_{8j+39} = \overline{\{n_{R,8j+7}\}} \oplus b_{8j+39} = \overline{n_{R,8j+7}} \oplus b_{8j+39} \oplus b_{8j+39} = \overline{n_{R,8j+7}}$ . Hence, the parity of this byte changes:  $p'_{j+4} = n'_{R,8j} \oplus \dots \oplus n'_{R,8j+6} \oplus n'_{R,8j+7} \oplus 1 = n_{R,8j} \oplus \dots \oplus n_{R,8j+6} \oplus \overline{n_{R,8j+7}} \oplus 1 = \overline{p_{j+4}}$ .

Now  $\{p_{j+4}\} \oplus \{p'_{j+4}\} = p_{j+4} \oplus b_{8j+40} \oplus p'_{j+4} \oplus b'_{8j+40} = p_{j+4} \oplus b_{8j+40} \oplus \overline{p_{j+4}} \oplus b'_{8j+40} = b_{8j+40} \oplus b'_{8j+40}$ . Hence  $\{p_{j+4}\} = \{p'_{j+4}\}$  if and only if  $b_{8j+40} = b'_{8j+40}$ , i.e.,  $\{p_{j+4}\} \neq \{p'_{j+4}\}$  if and only if  $n_R$  has property  $F_j$ .  $\square$

The theorem below shows that the probability that  $n_R$  has the property  $F_j$  is approximately 9.4%.

**Lemma 4.3.** Let  $Y_0, \dots, Y_4$  be independent uniformly distributed random variables over  $\mathbb{F}_2$ . Then

$$P[f_b(Y_0, Y_1, Y_2, Y_3) \neq f_b(Y_0, Y_1, Y_2, \overline{Y_3})] = \frac{1}{4}$$

$$P[f_c(Y_0, Y_1, Y_2, Y_3, Y_4) \neq f_c(Y_0, Y_1, Y_2, Y_3, \overline{Y_4})] = \frac{3}{8}.$$

**Proof.** By inspection.  $\square$

**Theorem 4.4.** Let  $Y_0, Y_1, \dots, Y_{18}, Y_{19}$  be independent uniformly distributed random variables over  $\mathbb{F}_2$ . Then

$$P[f(Y_0, \dots, Y_{18}, Y_{19}) \neq f(Y_0, \dots, Y_{18}, \overline{Y_{19}})] = \frac{3}{32}.$$

**Proof.** Write  $Z_0 := f_a(Y_0, \dots, Y_3)$ ,  $Z_1 := f_b(Y_4, \dots, Y_7)$ ,  $Z_2 := f_b(Y_8, \dots, Y_{11})$ ,  $Z_3 := f_a(Y_{12}, \dots, Y_{15})$ , and  $Z_4 := f_b(Y_{16}, \dots, Y_{19})$ . Furthermore, write  $Z'_4 := f_b(Y_{16}, \dots, Y_{18}, \overline{Y_{19}})$ . Note that  $Z_0, \dots, Z_4$  are independent and, by Theorem 2.3,

uniformly distributed over  $\mathbb{F}_2$ . Then

$$\begin{aligned}
& P[f(Y_0, Y_1, \dots, Y_{18}, Y_{19}) \neq f(Y_0, Y_1, \dots, Y_{18}, \overline{Y_{19}})] \\
&= P[f_c(Z_0, \dots, Z_4) \neq f_c(Z_0, \dots, Z_3, Z'_4)] \\
&= P[f_c(Z_0, \dots, Z_4) \neq f_c(Z_0, \dots, Z'_4) | Z_4 \neq Z'_4] \\
&\quad \cdot P[Z_4 \neq Z'_4] \\
&= P[f_c(Z_0, \dots, Z_3, 0) \neq f_c(Z_0, \dots, Z_3, 1)] \\
&\quad \cdot P[f_a(Y_{16}, \dots, Y_{18}, 0) \neq f_a(Y_{16}, \dots, Y_{18}, 1)] \\
&= \frac{3}{8} \cdot \frac{1}{4} \quad (\text{by Lemma 4.3}) \\
&= \frac{3}{32}.
\end{aligned}$$

Alternatively, one can also obtain this result by simply checking all  $2^{20}$  possibilities.  $\square$

We now describe how an attacker can find an  $\{n_R\}$  such that  $n_R$  has all four properties  $F_j$ . Recall that these properties also depend on  $n_T$  and it is possible that for a fixed  $n_T$  no  $n_R$  has all four properties. In that case, as is explained in the protocol below, the attacker makes the tag generate a different  $n_T$  and starts the search again.

**Protocol 4.5.** An attacker can find  $\{n_R\}$  such that  $n_R$  has properties  $F_0, F_1, F_2, F_3$  in a backtracking fashion. She first loops over all possibilities for the first byte of  $\{n_R\}$  (taking the other bytes of  $\{n_R\}$  arbitrary). Using Protocol 4.2, the attacker decides if  $n_R$  has property  $F_0$  (which only depends on the first byte). If it has, she continues with the second byte of  $\{n_R\}$ , looping over all possibilities for the second byte of  $\{n_R\}$  while keeping the first byte fixed, trying to find  $\{n_R\}$  such that  $n_R$  also has property  $F_1$ . She repeats this for the third and fourth byte of  $\{n_R\}$ . If at some stage no possible byte has property  $F_j$ , the search backtracks to the previous stage. It fails at the first stage, the attacker has to try a different tag nonce.

By simulating this protocol (for a random key and random uid, and a random tag nonce in every outer loop of the search), we can estimate the number of authentication attempts needed to find a reader nonce having all four properties  $F_j$ .

**Observation 4.6.** *The expected number of authentication attempts needed to find an  $n_R$  which has all four properties  $F_j$  is approximately 28500.*

Once the attacker has found an  $n_R$  having all four properties  $F_j$ , the number of possibilities for the internal state of the cipher after feeding in this particular  $n_R$  is seriously restricted. The following theorem states how many possibilities there still are.

**Theorem 4.7.** *Suppose that  $n_R$  has properties  $F_0, F_1, F_2$ , and  $F_3$ . Then there are only 436 possibilities*

0x000041414110	0x000041414140	0x000141414110	0x000141414140	0x000441414110	0x000441414140
0x000441414140	0x001441414110	0x001441414140	0x001541414110	0x001541414140	0x005141414110
0x005141414110	0x005141414140	0x00A441414110	0x00A441414140	0x00F441414110	0x00F441414140
0x010441414110	0x010441414140	0x011441414110	0x011441414140	0x014441414110	0x014441414140
0x014441414140	0x014441414110	0x01A441414110	0x01A441414140	0x01F441414110	0x01F441414140
0x020441414110	0x020441414140	0x024041414110	0x024041414140	0x028041414110	0x028041414140
0x028041414140	0x028041414110	0x02C041414110	0x02C041414140	0x030041414110	0x030041414140
0x030041414140	0x030041414110	0x034041414110	0x034041414140	0x038041414110	0x038041414140
0x038041414140	0x038041414110	0x03C041414110	0x03C041414140	0x040041414110	0x040041414140
0x040041414140	0x040041414110	0x044041414110	0x044041414140	0x048041414110	0x048041414140
0x048041414140	0x048041414110	0x04C041414110	0x04C041414140	0x050041414110	0x050041414140
0x050041414140	0x050041414110	0x054041414110	0x054041414140	0x058041414110	0x058041414140
0x058041414140	0x058041414110	0x05C041414110	0x05C041414140	0x060041414110	0x060041414140
0x060041414140	0x060041414110	0x064041414110	0x064041414140	0x068041414110	0x068041414140
0x068041414140	0x068041414110	0x06C041414110	0x06C041414140	0x070041414110	0x070041414140
0x070041414140	0x070041414110	0x074041414110	0x074041414140	0x078041414110	0x078041414140
0x078041414140	0x078041414110	0x07C041414110	0x07C041414140	0x080041414110	0x080041414140
0x080041414140	0x080041414110	0x084041414110	0x084041414140	0x088041414110	0x088041414140
0x088041414140	0x088041414110	0x08C041414110	0x08C041414140	0x090041414110	0x090041414140
0x090041414140	0x090041414110	0x094041414110	0x094041414140	0x098041414110	0x098041414140
0x098041414140	0x098041414110	0x09C041414110	0x09C041414140	0x0A0041414110	0x0A0041414140
0x0A0041414140	0x0A0041414110	0x0A4041414110	0x0A4041414140	0x0A8041414110	0x0A8041414140
0x0A8041414140	0x0A8041414110	0x0AC041414110	0x0AC041414140	0x0B0041414110	0x0B0041414140
0x0B0041414140	0x0B0041414110	0x0B4041414110	0x0B4041414140	0x0B8041414110	0x0B8041414140
0x0B8041414140	0x0B8041414110	0x0BC041414110	0x0BC041414140	0x0C0041414110	0x0C0041414140
0x0C0041414140	0x0C0041414110	0x0C4041414110	0x0C4041414140	0x0C8041414110	0x0C8041414140
0x0C8041414140	0x0C8041414110	0x0CC041414110	0x0CC041414140	0x0D0041414110	0x0D0041414140
0x0D0041414140	0x0D0041414110	0x0D4041414110	0x0D4041414140	0x0D8041414110	0x0D8041414140
0x0D8041414140	0x0D8041414110	0x0DC041414110	0x0DC041414140	0x0E0041414110	0x0E0041414140
0x0E0041414140	0x0E0041414110	0x0E4041414110	0x0E4041414140	0x0E8041414110	0x0E8041414140
0x0E8041414140	0x0E8041414110	0x0EC041414110	0x0EC041414140	0x0F0041414110	0x0F0041414140
0x0F0041414140	0x0F0041414110	0x0F4041414110	0x0F4041414140	0x0F8041414110	0x0F8041414140
0x0F8041414140	0x0F8041414110	0x0FC041414110	0x0FC041414140	0x100041414110	0x100041414140
0x100041414140	0x100041414110	0x104041414110	0x104041414140	0x108041414110	0x108041414140
0x108041414140	0x108041414110	0x10C041414110	0x10C041414140	0x110041414110	0x110041414140
0x110041414140	0x110041414110	0x114041414110	0x114041414140	0x118041414110	0x118041414140
0x118041414140	0x118041414110	0x11C041414110	0x11C041414140	0x120041414110	0x120041414140
0x120041414140	0x120041414110	0x124041414110	0x124041414140	0x128041414110	0x128041414140
0x128041414140	0x128041414110	0x12C041414110	0x12C041414140	0x130041414110	0x130041414140
0x130041414140	0x130041414110	0x134041414110	0x134041414140	0x138041414110	0x138041414140
0x138041414140	0x138041414110	0x13C041414110	0x13C041414140	0x140041414110	0x140041414140
0x140041414140	0x140041414110	0x144041414110	0x144041414140	0x148041414110	0x148041414140
0x148041414140	0x148041414110	0x14C041414110	0x14C041414140	0x150041414110	0x150041414140
0x150041414140	0x150041414110	0x154041414110	0x154041414140	0x158041414110	0x158041414140
0x158041414140	0x158041414110	0x15C041414110	0x15C041414140	0x160041414110	0x160041414140
0x160041414140	0x160041414110	0x164041414110	0x164041414140	0x168041414110	0x168041414140
0x168041414140	0x168041414110	0x16C041414110	0x16C041414140	0x170041414110	0x170041414140
0x170041414140	0x170041414110	0x174041414110	0x174041414140	0x178041414110	0x178041414140
0x178041414140	0x178041414110	0x17C041414110	0x17C041414140	0x180041414110	0x180041414140
0x180041414140	0x180041414110	0x184041414110	0x184041414140	0x188041414110	0x188041414140
0x188041414140	0x188041414110	0x18C041414110	0x18C041414140	0x190041414110	0x190041414140
0x190041414140	0x190041414110	0x194041414110	0x194041414140	0x198041414110	0x198041414140
0x198041414140	0x198041414110	0x19C041414110	0x19C041414140	0x1A0041414110	0x1A0041414140
0x1A0041414140	0x1A0041414110	0x1A4041414110	0x1A4041414140	0x1A8041414110	0x1A8041414140
0x1A8041414140	0x1A8041414110	0x1AC041414110	0x1AC041414140	0x1B0041414110	0x1B0041414140
0x1B0041414140	0x1B0041414110	0x1B4041414110	0x1B4041414140	0x1B8041414110	0x1B8041414140
0x1B8041414140	0x1B8041414110	0x1BC041414110	0x1BC041414140	0x1C0041414110	0x1C0041414140
0x1C0041414140	0x1C0041414110	0x1C4041414110	0x1C4041414140	0x1C8041414110	0x1C8041414140
0x1C8041414140	0x1C8041414110	0x1CC041414110	0x1CC041414140	0x1D0041414110	0x1D0041414140
0x1D0041414140	0x1D0041414110	0x1D4041414110	0x1D4041414140	0x1D8041414110	0x1D8041414140
0x1D8041414140	0x1D8041414110	0x1DC041414110	0x1DC041414140	0x1E0041414110	0x1E0041414140
0x1E0041414140	0x1E0041414110	0x1E4041414110	0x1E4041414140	0x1E8041414110	0x1E8041414140
0x1E8041414140	0x1E8041414110	0x1EC041414110	0x1EC041414140	0x1F0041414110	0x1F0041414140
0x1F0041414140	0x1F0041414110	0x1F4041414110	0x1F4041414140	0x1F8041414110	0x1F8041414140
0x1F8041414140	0x1F8041414110	0x1FC041414110	0x1FC041414140	0x200041414110	0x200041414140
0x200041414140	0x200041414110	0x204041414110	0x204041414140	0x208041414110	0x208041414140
0x208041414140	0x208041414110	0x20C041414110	0x20C041414140	0x210041414110	0x210041414140
0x210041414140	0x210041414110	0x214041414110	0x214041414140	0x218041414110	0x218041414140
0x218041414140	0x218041414110	0x21C041414110	0x21C041414140	0x220041414110	0x220041414140
0x220041414140	0x220041414110	0x224041414110	0x224041414140	0x228041414110	0x228041414140
0x228041414140	0x228041414110	0x22C041414110	0x22C041414140	0x230041414110	0x230041414140
0x230041414140	0x230041414110	0x234041414110	0x234041414140	0x238041414110	0x238041414140
0x238041414140	0x238041414110	0x23C041414110	0x23C041414140	0x240041414110	0x240041414140
0x240041414140	0x240041414110	0x244041414110	0x244041414140	0x248041414110	0x248041414140
0x248041414140	0x248041414110	0x24C041414110	0x24C041414140	0x250041414110	0x250041414140
0x250041414140	0x250041414110	0x254041414110	0x254041414140	0x258041414110	0x258041414140
0x258041414140	0x258041414110	0x25C041414110	0x25C041414140	0x260041414110	0x260041414140
0x260041414140	0x260041414110	0x264041414110	0x264041414140	0x268041414110	0x268041414140
0x268041414140	0x268041414110	0x26C041414110	0x26C041414140	0x270041414110	0x270041414140
0x270041414140	0x270041414110	0x274041414110	0x274041414140	0x278041414110	0x278041414140
0x278041414140	0x278041414110	0x27C041414110	0x27C041414140	0x280041414110	0x280041414140
0x280041414140	0x280041414110	0x284041414110	0x284041414140	0x288041414110	0x288041414140
0x288041414140	0x288041414110	0x28C041414110	0x28C041414140	0x290041414110	0x290041414140
0x290041414140	0x290041414110	0x294041414110	0x294041414140	0x298041414110	0x298041414140
0x298041414140	0x298041414110	0x29C041414110	0x29C041414140	0x2A0041414110	0x2A0041414140
0x2A0041414140	0x2A0041414110	0x2A4041414110	0x2A4041414140	0x2A8041414110	0x2A8041414140
0x2A8041414140	0x2A8041414110	0x2AC041414110	0x2AC041414140	0x2B0041414110	0x2B0041414140
0x2B0041414140	0x2B0041414110	0x2B4041414110	0x2B4041414140	0x2B8041414110	0x2B8041414140
0x2B8041414140	0x2B8041414110	0x2BC041414110	0x2BC041414140	0x2C0041414110	0x2C0041414140
0x2C0041414140	0x2C0041414110	0x2C4041414110	0x2C4041414140	0x2C8041414110	0x2C8041414140
0x2C8041414140	0x2C8041414110	0x2CC041414110	0x2CC041414140	0x2D0041414110	0x2D0041414140
0x2D0041414140	0x2D0041414110	0x2D4041414110	0x2D4041414140	0x2D8041414110	0x2D8041414140
0x2D8041414140	0x2D8041414110	0x2DC041414110	0x2DC041414140	0x2E0041414110	0x2E0041414140

```

0x0000004d4d1f 0x0000012d7b8b 0x000001513ca3 0x0000049e0e78 0x000004cafec1
0x000006f945be 0x000007089ea5 0x0000072b67df 0x000008e79d8e 0x00000a137cd9
0x00000aed7467 0x00000b92342b 0x00000c6db6a0 0x00000cbd2daa 0x00000cda7817
0x00000d0cb27 0x00000e98af03 0x00001089393d 0x0000129d78db 0x000012f4cde6
0x000015382c19 0x000016a7a95c 0x0000172bebc6 0x0000173f2299 0x00001821aa0a
0x000018769666 0x00001a6d513e 0x00001b1c2ff7 0x00001c259261 0x00001c46edf7
0x00001c5a3fde 0x00001c97ee44 0x00001f19da5e 0x00001fe9ec2 0x000022ce6797
0x000023a396ce 0x000023a92baa 0x000026bc6e18 0x0000278a7954 ...

```

Table 4.2. Excerpt from table  $T_{0xa04}$  of internal cipher states  $\alpha_{32}$  at index  $0xa04$

she gained knowledge about the internal cipher state. Now the attacker does the opposite: she keeps  $\{n_R\}$  (and  $\{a_R\}$  and the  $\{p_i\}$  as well) constant, but varies  $n_T$  instead. As before, the attacker waits for the tag to respond; when this happens, she gains knowledge about the internal state of the cipher.

**Protocol 4.8.** The attacker repeatedly tries to authenticate to the tag, every time with a different tag nonce  $n_T$  and sending all zeros as its response (including the encrypted parity bits), i.e.,  $\{n_R\} = 0$ ,  $\{a_R\} = 0$ ,  $\{p_4\} = \dots = \{p_{11}\} = 0$ . She waits for an  $n_T$  such that the tag actually responds (i.e., the parity bits are the correct parity bits) and where the encrypted error code is  $0x5$  (i.e.,  $b_{96} = b_{97} = b_{98} = b_{99} = 0$ ).

Note that twelve bits have to be ‘correct’ (the eight parity bits and the four keystream bits), so this will take on average  $2^{12} = 4096$  authentication attempts.

The following defines a large table that needs to be precomputed.

**Definition 4.9.**

$$T := \{\alpha_{32} \in \mathbb{F}_2^{48} \mid \{n_R\} = \{a_R\} = 0 \Rightarrow \{p_4\} = \dots = \{p_{11}\} = b_{96} = \dots = b_{99} = 0\}.$$

So the attacker knows that after the tag sends the challenge  $n_T$  found in Protocol 4.8, the current state of the cipher,  $\alpha_{32}$ , appears in  $T$ . Now  $T$  can be precomputed; one would expect it to contain  $2^{48}/2^{12} = 2^{36}$  elements; in fact, it contains 0.82% fewer elements due to a small bias in the cipher. In principle, the attacker could now use Theorem 2.8 to roll back each of the LFSRs in the table to find candidate keys and check each of these keys against a few other attempted authentication sessions.

In practice, searching through  $T$  takes about one day, which is undesirable. The attacker can shrink the search space by splitting  $T$  as follows.

**Protocol 4.10.** After finding  $n_T$  in Protocol 4.8, the attacker again repeatedly tries to authenticate to the tag, every time with the tag nonce  $n_T$  she just found. Instead of zeros, she now sends ones for the response and this time she tries all possibilities for the encrypted

parity bits until the tag responds with an encrypted error code. I.e.,  $\{n_R\} = 0xffffffff$  and  $\{a_R\} = 0xffffffff$  and successively tries all possibilities for  $\{p_4\}, \dots, \{p_{11}\}$  until one is correct.

This time, because eight bits have to be ‘correct’, on average 128 authentication attempts are needed.

The table  $T$  can be split in  $2^{12} = 4096$  parts indexed by the eight encrypted parity bits and four keystream bits that encrypt the error code.

**Definition 4.11.** For every  $\gamma = \gamma_0 \dots \gamma_{11} \in \mathbb{F}_2^{12}$  we define

$$T_\gamma := \{\alpha_{32} \in T \mid \{n_R\} = \{a_R\} = 0xffffffff \Rightarrow \{p_4\} = \gamma_0 \wedge \dots \wedge \{p_{11}\} = \gamma_7 \wedge b_{96} = \gamma_8 \wedge \dots \wedge b_{99} = \gamma_{11}\}.$$

So instead of storing  $T$  as one big table, during precomputation the attacker creates the 4096 tables  $T_\gamma$ . Taking  $\gamma := \{p_4\} \dots \{p_{11}\} b_{96} \dots b_{99}$  at the end of Protocol 4.10, the attacker knows that  $\alpha_{32}$  must be an element of  $T_\gamma$ . Now  $T_\gamma$  contains only approximately  $2^{24}$  entries, so this can easily be read from disk to generate  $2^{24}$  candidate keys and check them against a few other authentication sessions. Table 4.2 shows, as an example, the first part of  $T_\gamma$  for  $\gamma = 0xa04 = 1010\ 0000\ 0100$ .

#### 4.4. Nested authentication attack

We now assume that the attacker already knows at least one sector key; let us call this sector the exploit sector.

The time between two consecutive authentication attempts might vary from card to card, although it is quite constant for a specific card. Therefore, an attacker can first estimate this time by authenticating two times for the exploit sector. In this way the attacker can estimate the distance  $\delta$  between the first and the second tag nonce.

As explained in Section 3.2, the attacker can now authenticate for the exploit sector and subsequently for another sector. In the authentication for the exploit sector the tag nonce  $n_T^0$  is sent in the clear; during the second authentication the tag nonce  $n_T$  is sent encrypted as  $\{n_T\}$ . By computing  $\text{suc}^i(n_T^0)$  for  $i$  close to  $\delta$ , the adversary has a small number of guesses for  $n_T$ . The adversary can further narrow the possibilities for  $n_T$  using the three bits of information from the parity bits (Theorem 3.3). In this way the adversary can accurately guess  $n_T$  and hence recover the first 32 bits of keystream,  $b_0 b_1 \dots b_{31}$ .

We shall show how a variant of the attack of Section 6.3 of [GKM<sup>+</sup>08] can be used to recover

approximately  $2^{16}$  possible candidate keys. By doing this procedure two or three times, the attacker can recover the key for the second sector as well by taking the intersection of the two or three sets of candidate keys.

The crucial ingredient in the attack is the fact that the inputs to the filter function are only on odd-numbered places of the LFSR. This makes it possible to compute separately all possibilities for the odd-numbered bits of the LFSR-stream and the even-numbered bits of the LFSR-stream that are compatible with the keystream.

**Definition 4.12.** We define the odd tables  $T_i^O$  by

$$T_0^O := \{x_9x_{11} \dots x_{45}x_{47} \in \mathbb{F}_2^{20} \mid f(x_9x_{11} \dots x_{45}x_{47}) = b_0\}$$

and for  $i \in \{1, \dots, 15\}$

$$T_i^O := \{x_9x_{11} \dots x_{45+2i}x_{47+2i} \in \mathbb{F}_2^{20+i} \mid x_9x_{11} \dots x_{45+2i} \in T_{i-1}^O \wedge f(x_{9+2i}x_{11+2i} \dots x_{45+2i}x_{47+2i}) = b_{2i}\}.$$

Symmetrically, we define the even tables  $T_i^E$  by

$$T_0^E := \{x_{10}x_{12} \dots x_{46}x_{48} \in \mathbb{F}_2^{20} \mid f(x_{10}x_{12} \dots x_{46}x_{48}) = b_1\}$$

and for  $i \in \{1, \dots, 15\}$

$$T_i^E := \{x_{10}x_{12} \dots x_{46+2i}x_{48+2i} \in \mathbb{F}_2^{20+i} \mid x_{10}x_{12} \dots x_{46+2i} \in T_{i-1}^E \wedge f(x_{10+2i}x_{12+2i} \dots x_{46+2i}x_{48+2i}) = b_{2i+1}\}.$$

We write  $T^O := T_{15}^O$  and  $T^E := T_{15}^E$ .

Because of the structure of the filter function  $f$ ,  $T_0^O$  and  $T_0^E$  are exactly of size  $2^{19}$  (Theorem 2.3). The other tables are approximately of this size as well. An entry  $x_9x_{11} \dots x_{45+2i}$  of  $T_{i-1}^O$  leads to four different possibilities in  $T_i^O$ : it can appear in  $T_i^O$  extended with 0 and with 1; it can appear extended only with 0; it can appear extended only with 1; or it can not appear at all. Overall, these possibilities are equally likely, and hence  $T_i^O$  has, on average, the same size as  $T_{i-1}^O$  (and similarly for  $T^E$ ).

The feedback function  $L$  can also be split in an even and an odd part.

**Definition 4.13.** We define the odd part of the feedback function,  $L^O: \mathbb{F}_2^{24} \rightarrow \mathbb{F}_2$ , by  $L^O(x_1x_3 \dots x_{47}) := x_5 \oplus x_9 \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{43}$  and the even part of the feedback function,  $L^E: \mathbb{F}_2^{24} \rightarrow \mathbb{F}_2$ , by  $L^E(x_0x_2 \dots x_{46}) := x_0 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{24} \oplus x_{42}$ .

Note that  $L^E$  and  $L^O$  combine to give  $L$ , in the sense that

$$L(x_0x_1x_2 \dots x_{47}) = L^E(x_0x_2 \dots x_{46}) \oplus L^O(x_1x_2 \dots x_{47}). \quad (2)$$

As the  $a_9a_{10} \dots a_{77}a_{78}$  are being shifted through the LFSR, the uid  $u$  and the tag nonce  $n_T$  are shifted in as well. In the following definition we compute the 22 bits of feedback from the LFSR from time 9 to time 31, taking care of the shifting in of  $u \oplus n_T$ , and also splitting the contribution from the odd- and even-numbered bits of the LFSR. At this point, the situation in [GKM<sup>+</sup>08] is slightly simpler. There, the attacker tries to find the state of the LFSR after initialization, so nothing is being shifted in.

**Definition 4.14.** We define the contribution of the entries of the odd table to the feedback,  $\psi^O: T^O \rightarrow \mathbb{F}_2^{22}$ , by

$$\begin{aligned} \psi^O(x_9x_{11} \dots x_{77}) := & (L^E(x_{9+2i}x_{11+2i} \dots x_{55+2i}) \oplus n_{T,9+2i} \oplus u_{9+2i}, \\ & L^O(x_{11+2i}x_{13+2i} \dots x_{57+2i}) \oplus n_{T,10+2i} \\ & \oplus u_{10+2i})_{i \in [0,10]} \end{aligned}$$

and we define the contribution of the entries of the even table to the feedback,  $\psi^E: T^E \rightarrow \mathbb{F}_2^{22}$ , by

$$\begin{aligned} \psi^E(x_{10}x_{12} \dots x_{78}) := & (L^O(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{57+2i}, \\ & L^E(x_{10+2i}x_{12+2i} \dots x_{56+2i}) \oplus x_{58+2i})_{i \in [0,10]}. \end{aligned}$$

**Definition 4.15.** We define the combined table  $T^C$  as follows.

$$T^C := \{x_9x_{10}x_{11} \dots x_{78} \in \mathbb{F}_2^{70} \mid x_9x_{11} \dots x_{77} \in T^O \wedge x_{10}x_{12} \dots x_{78} \in T^E \wedge \psi^O(x_9x_{11} \dots x_{77}) = \psi^E(x_{10}x_{12} \dots x_{78})\}.$$

Note that  $T^C$  can easily be computed by first sorting  $T^O$  by  $\psi^O$  and  $T^E$  by  $\psi^E$ .

The crucial point is the following theorem; it shows that the actual LFSR-stream of the tag under attack is in the table  $T^C$ .

**Theorem 4.16.**  $a_9a_{10}a_{11} \dots a_{78} \in T^C$ .

**Proof.** By definition of  $T^O$  and  $T^E$ ,  $a_9a_{11} \dots a_{77} \in T^O$  and  $a_{10}a_{12} \dots a_{78} \in T^E$ . We only have to check that the sequence  $a_9a_{10}a_{11} \dots a_{78}$  satisfies the con-

straint defining  $T^C$ . For this, we have

$$\begin{aligned}
& \psi^O(a_9 a_{11} \dots a_{77}) \oplus \psi^E(a_{10} a_{12} \dots a_{78}) \\
&= (L^E(x_{9+2i} x_{11+2i} \dots x_{55+2i}) \oplus n_{T,9+2i} \oplus u_{9+2i} \\
&\quad \oplus L^O(x_{10+2i} x_{12+2i} \dots x_{56+2i}) \oplus x_{57+2i}), \\
&\quad L^O(x_{11+2i} x_{13+2i} \dots x_{57+2i}) \oplus n_{T,10+2i} \oplus u_{10+2i} \\
&\quad \oplus L^E(x_{10+2i} x_{12+2i} \dots x_{56+2i}) \oplus x_{58+2i})_{i \in [0,10]} \\
&\hspace{15em} \text{(by Dfn. 4.14)} \\
&= (L(x_{9+2i} x_{10+2i} \dots x_{56+2i}) \\
&\quad \oplus n_{T,9+2i} \oplus u_{9+2i} \oplus x_{57+2i}, \\
&\quad L(x_{10+2i} x_{11+2i} \dots x_{57+2i}) \\
&\quad \oplus n_{T,10+2i} \oplus u_{10+2i} \oplus x_{58+2i})_{i \in [0,10]} \\
&\hspace{15em} \text{(by Eqn. (2))} \\
&= (0, 0)_{i \in [0,10]}, \hspace{15em} \text{(by Dfn. 2.6)}
\end{aligned}$$

as required.  $\square$

Taking the first 48 bits of every entry of  $T^C$ , the attacker can apply Theorem 2.8 nine times for every entry, obtaining one candidate key for every entry of  $T^C$ . Because we have used 32 bits of keystream and the key is 48 bits, on average there will be  $2^{16}$  candidate keys. Doing this procedure once more gives another set of approximately  $2^{16}$  candidate keys; the actual key must be in the intersection. In practice, most of the time the intersection only contains a single key; occasionally it contains two keys and then a third run of this whole procedure can be used to determine the key (or both candidate keys can just be tested online, of course).

## 5. Conclusions

We have found serious ‘textbook’ vulnerabilities in the Mifare Classic tag. In particular, the Mifare Classic mixes two layers of the protocol stack and reuses a one-time pad for the encryption of the parity bits. It also sends encrypted error messages before a successful authentication. These weaknesses allow an adversary to recover a secret key within seconds. Moreover, tag nonces are predictable which, besides allowing replays, provides known plaintext for our nested authentication attack. We have executed these attacks in practice and retrieved all secret keys from a number of cards, including cards used in large access control and public transport ticketing systems.

To slightly hamper an adversary, system integrators could consider the following countermeasures:

- diversify all keys in the card;
- cryptographically bind the contents of the card to the uid, for instance by including a MAC;

- perform regular integrity checks in the back office.

For the time being, the second countermeasure prevents an attacker from cloning a card onto a blank one. However, this does not stop an attacker from emulating that card with an emulator like the Proxmark.

Early on we have notified the manufacturer NXP of these vulnerabilities. Since the protocol is implemented in hardware, we do not foresee any definitive countermeasure to these attacks that does not require replacing the entire infrastructure. However, NXP is currently developing a backwards compatible successor to the Mifare Classic, the Mifare Plus. We are collaborating with NXP, providing feedback to help them improving the security of their new prototypes, given the limitations of the backwards compatibility mode.

## Acknowledgments

We are grateful to our faculty’s computer department (C&CZ) for providing us with computing power and to Ben Polman in particular for his assistance.

## References

- [Bih97] Eli Biham. A fast new DES implementation in software. In *Fast Software Encryption (FSE '97)*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272, 1997.
- [CNO08] Nicolas T. Courtois, Karsten Nohl, and Sean O’Neil. Algebraic attacks on the Crypto-1 stream cipher in Mifare Classic and Oyster Cards. *Cryptology ePrint Archive*, Report 2008/166, 2008.
- [GKM<sup>+</sup>08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In Sushil Jajodia and Javier Lopez, editors, *European Symposium on Research in Computer Security (ESORICS '08)*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer, 2008.
- [ISO01] Identification cards — contactless integrated circuit cards — proximity cards (ISO/IEC 14443), 2001.
- [KHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A practical attack on the MIFARE Classic. In Gilles Grimaud and Francois-Xavier Standaert, editors, *Smart Card Research and Advanced Application (CARDIS '08)*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer, 2008.

- [KPP<sup>+</sup>06] Sandeep Kumar, Christof Paar, Jan Pelzl, Gerd Pfeiffer, and Manfred Schimmler. Breaking ciphers with COPACOBANA - a cost-optimized parallel code breaker. In *Cryptographic Hardware and Embedded Systems (CHES '06)*, volume 4249 of *Lecture Notes in Computer Science*, pages 101–118. Springer, 2006.
- [Kra01] Hugo Krawczyk. The order of encryption and authentication for protecting communications (or: How secure is SSL?). In *Advances in Cryptology (CRYPTO '01)*, pages 310–331. Springer, 2001.
- [MAD07] Mifare application directory. [http://www.nxp.com/acrobat\\_download/other/identification/M001830.pdf](http://www.nxp.com/acrobat_download/other/identification/M001830.pdf), May 2007.
- [MFS08] MF1ICS50 functional specification. [http://www.nxp.com/acrobat/other/identification/M001053\\_MF1ICS50\\_rev5\\_3.pdf](http://www.nxp.com/acrobat/other/identification/M001053_MF1ICS50_rev5_3.pdf), January 2008.
- [NESP08] Karsten Nohl, David Evans, Starbug, and Henryk Plötz. Reverse-engineering a cryptographic RFID tag. In *USENIX Security 2008*, pages 185–193, 2008.
- [Noh08] Karsten Nohl. Cryptanalysis of Crypto-1. <http://www.cs.virginia.edu/~kn5f/Mifare.Cryptanalysis.htm>, 2008.
- [NP07] Karsten Nohl and Henryk Plötz. Mifare, little security despite obscurity. Presentation at Chaos Computer Congress, 2007.