A Practical Attack on the MIFARE Classic

Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia

Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
g.t.dekoninggans@student.ru.nl,
jhh@cs.ru.nl,
flaviog@cs.ru.nl

Abstract. The MIFARE Classic is the most widely used contactless smart card in the market. Its design and implementation details are kept secret by its manufacturer. This paper studies the architecture of the card and the communication protocol between card and reader. It reveals command codes and structure that so far were unknown. It also gives a practical, low-cost attack that recovers secret information from the memory of the card. Due to a weakness in the pseudo-random generator we are able to recover the keystream generated by the CRYPTO1 stream cipher. Finally, we exploit the malleability of the stream cipher to read *all* memory blocks of the first sector of the card. Moreover, we are able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector.

1 Introduction

RFID and contactless smart cards have become pervasive technologies nowadays. Over the last few years, more and more systems adopted this technology as replacement for barcodes, magnetic stripe cards and paper tickets for a variety of applications. Contact-less cards consist of a small piece of memory that can be accessed wirelessly, but unlike RFID tags, they also have some computing capabilities. Most of these cards implement some sort of simple symmetric-key cryptography, which makes them suitable for applications that require access control.

A number of high profile applications make use of contactless smart cards for access control. For example, they are used for payment in several public transport systems like the Octopus card[1] in Hong Kong, the Oyster card[2] in London, and the OV-Chipkaart[3] in The Netherlands, among others. Many countries have already incorporated a contactless card in their electronic passports [2] and several car manufacturers have it embedded in their car keys as an anti-theft method. Many office buildings and even secured facilities like airports and military bases, use contactless smart cards for access control.

[1] http://www.octopuscards.com/
[2] http://oyster.tfl.gov.uk
[3] http://www.ov-chipkaart.nl/

On the one hand, the wireless interface has practical advantages: without mechanical components between readers and cards, the system has lower maintenance costs, is more reliable, and has shorter reading times, providing higher throughput. On the other hand, it represents a potential threat to privacy [2] and it is susceptible to relay, replay and skimming attacks that were not possible before.

There is a huge variety of cards on the market. They differ in size, casing, memory and computing power. They also differ in the security features they provide. A well known and widely used system is MIFARE. MIFARE is a product family from NXP semiconductors (formerly Philips). According to NXP there are about 200 million MIFARE cards in use around the world, covering 85% of the contactless smartcard market. The MIFARE family contains four different types of cards: Ultralight, Standard, DESFire and SmartMX. The MIFARE Classic cards come in three different memory sizes: 320B, 1KB and 4KB. The MIFARE Classic is the most widely used contactless card in the market. Throughout this paper we focus on this card. MIFARE Classic cards provides mutual authentication and data secrecy by means of the so called CRYPTO1 stream cipher. This cipher is proprietary of NXP and its design is kept secret.

Nohl and Plötz [5] have recently reverse engineered the hardware of the chip and exposed several weaknesses. Among them, due to a weakness on the pseudo-random generator is the observation that the 32-bit nonces used for authentication have only 16 bits of entropy. They also noticed that the pseudo-random generator is stateless. They claim to have knowledge of the exact encryption algorithm which would facilitate an off-line brute force attack on the 48-bit keys. Such an attack would be feasible, in a reasonable amount of time, especially if dedicated hardware is available.

Our Contribution We used a Proxmark III[4] to analyze MIFARE cards and mount an attack. To do so, we have implemented the ISO 14443-A functionality on the Proxmark, since only ISO 14443-B was implemented at that time. We programmed both processing and generation of reader-to-tag and tag-to-reader communication at physical and higher levels of the protocol. The source code of the firmware is available in the public domain[5]. Concurrently and independently from Nohl and Plötz results, we also noticed a weakness in the pseudo-random generator. Our contribution is threefold: First and foremost, using the weakness of the pseudo-random generator, and given access to a particular MIFARE card, we are able to recover the keystream generated by the CRYPTO1 stream cipher, without knowing the encryption key. Secondly, we reveal the exact details of the communication between reader and card, including command codes and structure that so far were unknown. Finally, we exploit the malleability of the stream cipher to read *all* memory blocks of the first sector (sector zero) of the card (without having access to the secret key). In general, we are able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. After eavesdropping a transaction, we are always able to read the first 6 bytes of every block in that sector, and in most cases also de last 6 bytes. This leaves only 4 unrevealed bytes in those blocks.

[4] http://cq.cx/proxmark3.pl
[5] www.sos.cs.ru.nl/research/mifare

Consequences of our attack Any system using MIFARE Classic cards that relies on information stored on sector zero is now insecure. Our attack recovers, in a few minutes, *all* secret information in that sector. This is also true for most of the data in the remaining sectors, depending on the specific scenario. Besides, it complements Nohl and Plötz results providing the necessary plaintext for a brute force attack on the keys. This is currently work in progress.

Recommendations for improvement Our recommendations for short term improvement are not storing any sensible information in sector zero, such as authentication keys or any private information. Another sensible measure is to use multiple sector authentications, to thwart replay attacks.

As long term improvement we suggest to migrate to more advanced cards such as DESFire or better, to an open design architecture. Security by obscurity has shown several times to be insecure in the long term [4].

Outline of this paper Section 2 describes the architecture of the MIFARE cards and the communication protocol. In Section 5 exposes weaknesses in the design of the cards. In Section 3 we describe the hardware used to mount the attacks which are described in Section 6. Finally in Section 7 we give some concluding remarks and detailed suggestions for improvement.

2 MIFARE Classic

Contactless smartcards are used in many applications nowadays. Contactless cards are based on *radio frequency identification* technology (RFID) [1]. In 1995 NXP, Philips at that time, introduced MIFARE[6]. Some target applications of MIFARE are public transportation, access control and event ticketing. The MIFARE Classic [6] card is a member of the MIFARE product family and is compliant with ISO 14443-A up to part 3. Part 4 defines the high-level protocol and the implementation of NXP differs from the standard. Section 2.1 will discuss the different parts.

2.1 Communication Layer

The communication layer of the MIFARE Classic card is based on the ISO 14443 standard [3]. This ISO standard defines the communication for identification cards, contactless integrated circuit(s) cards and proximity cards. The standard consists of four parts.

Part 1 Physical characteristics
Part 2 Radio frequency interface power and signal interface
Part 3 Initialization and anticollision
Part 4 Transmission protocol

[6] http://www.nxp.com

Part 1 describes the physical characteristics and circumstances under which the card should be able to operate.
Part 2 defines the communication between the reader and card and vice versa. The data can be encoded and modulated in two ways, type A and type B. MIFARE Classic uses type A which defines *Amplitude Shift Keying* (ASK) for reader to card communication. To encode data bits the reader stops to generate a carrier for about 2μs with certain intervals. This corresponds with 100% ASK because there is no amplitude at all in this period. The card to reader communication for type A is done by load modulation. The card will add a subcarrier or not, *On-off Keying* (OOK), to encode data bits. For more detailed information about the communication on RFID we refer to the "RFID Handbook" by Klaus Finkenzeller [1].
Part 3 describes the initialization and anticollision protocol. The *anticollision* is needed to select a particular card when more cards are present within the reading range of the reader. After a successful initialization and anticollision the card is in an active state and ready to receive a command. This state is the starting point for part 4 of the standard and also the point where MIFARE Classic differs from the ISO standard.
The MIFARE Classic data sheets [6] do not mention any commands that could be send on this level nor does it specify answers from the card or the length of the messages. The data sheets does define though the structure of the memory of the card and how to organize it, which is explained in Section 2.2. The modulation of commands is done by the MIFARE Classic reader chip. Knowledge about the actual modulation is therefore not needed.

2.2 Logical Structure

A MIFARE Classic card is in principle a memory card with few extra functionalities. The memory is divided in data blocks of 16 bytes. Those data blocks are grouped into sectors. The MIFARE Classic 1k card has 16 sectors of 4 data blocks each. The first 32 sectors of a MIFARE Classic 4k card consists of 4 data blocks and the remaining 8 sectors consist of 16 data blocks. Every last data block of a sector is called *sector trailer*. A schematic of the memory of a MIFARE Classic 4k card is shown in Figure 1.

Note that block 0 of sector 0 contains special data. The first 4 data bytes contain the unique identifier of the card (UID) followed by its 1-byte *bit count check* (BCC). The bit count check is calculated by successively XOR-ing the separate UID bytes. The remaining bytes are used to store manufacturer data. This data block is set and immediately locked by the manufacturer so its contents cannot longer be modified. The reader needs to authenticate for a sector before any memory operations are allowed. The sector trailer contains the secret keys A and B which are used for authentication. The *access conditions* define which operations are available for this sector. Depending on which key is used for authentication and the access conditions for this key, different restrictions apply to the memory operations.
The sector trailer has special access conditions. Key A cannot be read by a reader. In some configurations key B is readable. In that case the memory is just used for data storage and key B cannot be used as a key for authentication. Besides the access

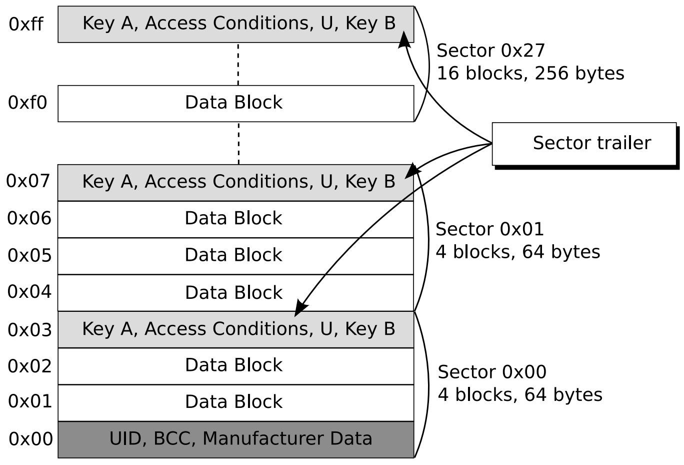
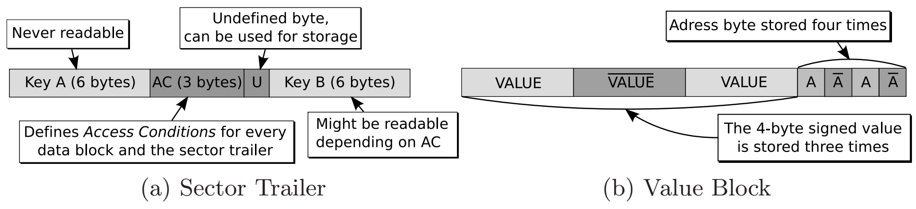


Fig. 1: MIFARE Classic 4k Memory

conditions (AC) and keys, there is one data byte (U) remaining which has no defined purpose. A schematic of the sector trailer is shown in Figure 2a. A data block is used to store arbitrary data or can be configured as a *value block*. When used as a value block a signed 4-byte value is stored twice non-inverted and once inverted. Inverted here means that every bit of the value is XOR-ed with 1. This 4 bytes are stored from the least significant byte on the left to the most significant byte on the right.



Never readable
Undefined byte, can be used for storage
Key A (6 bytes)
AC (3 bytes)
U
Key B (6 bytes)
Defines *Access Conditions* for every data block and the sector trailer
Might be readable depending on AC
(a) Sector Trailer
Adress byte stored four times
VALUE
$\overline{\text{VALUE}}$
VALUE
A
$\overline{\text{A}}$
A
$\overline{\text{A}}$
The 4-byte signed value is stored three times
(b) Value Block

Fig. 2: Block contents

The 4 bytes left are used to store a 1-byte block address and can be useful for back-up management, for instance to restore data blocks to a specific address on the card. The address is stored twice non-inverted and twice inverted. Besides this specific format the access conditions should be configured such that the specific value block commands are allowed for this block. The write command should be used to format a block as a value block.

2.3 Commands

The command set of MIFARE Classic is small. Most commands are related to a data block and require the reader to be authenticated for its containing sector. The access conditions are checked every time a command is executed to determine whether it is allowed or not. A block of data might be configured to be read only. Another example of a restriction might be a value block which can only be decremented.

Read and Write The read and write commands read or write one data block. This is either a data block or a value block. The write command can be used to format a data block as value block or just store arbitrary data.

Decrement, Increment, Restore and Transfer These commands are only allowed on data blocks that are formatted as value blocks. The increment and decrement commands will increment or decrement a value block with a given value and place the result in a memory register. The restore command loads a value into the memory register without any change. Finally the memory register is transferred in the same block or transferred to another block by the transfer command.

2.4 Security Features

The MIFARE Classic card has some built-in security features. The communication is encrypted by the proprietary stream cipher CRYPTO1.

Keys The 48-bit keys used for authentication are stored in the sector trailer of each sector (see section 2.2). MIFARE Classic uses symmetric keys.

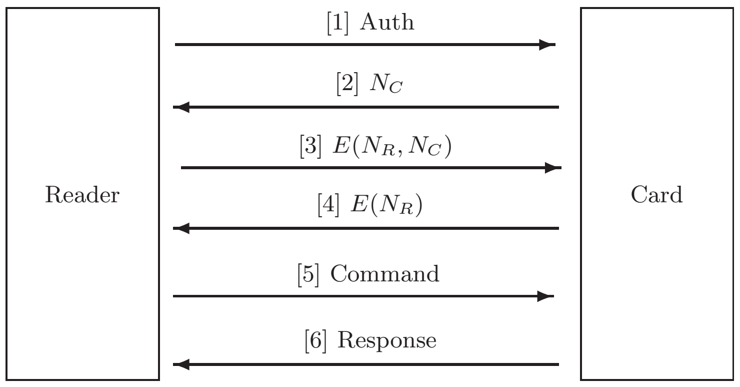


Fig. 3: Authentication followed by a command

Authentication Protocol MIFARE Classic makes use of a mutual three pass authentication protocol that seems to be based on the ISO 9798-2. The authentication scheme is shown in Figure 3. After this mutual authentication both, card and reader, are convinced that they share the same key. The authentication procedure starts after a successful anticollision when the card is in an active state. The reader sends a request for sector authentication and the card will respond with a 32-bit nonce N_C. Then, the reader sends back an encryption of the nonce N_C of the card together with its own nonce N_R and some additional input in an 8-byte response. This answer is the first encrypted message after the start of the authentication procedure. From this point on no messages are send in the clear anymore. The card decrypts the response (#3) and checks if it contains its own nonce N_C and therefore knows whether or not the reader possesses the same key. To show the reader that the card also knows the key, an encryption of N_R is send back (#4). The reader decrypts this response and checks for its own nonce N_C. The data sheets [6] do not give any information on the way these authentication messages are build. It is our believe, based on our experiments, that the above description is not far from the actual authentication protocol. As far as our attack is concerned this description captures all the necessary concepts.

3 Hardware and Software

An RFID system consists of a transponder (card) and a reader [1]. The reader contains a radio frequency module, a control unit and a coupling element to the card. The card contains a coupling element and a microchip. The control unit of a MIFARE Classic enabled reader is typically a MIFARE microchip with a closed design. This microchip communicates with the application software and executes commands from it. Note that the actual modulation of commands is done by this microchip and not by the application software. The design of the microchip of the card is closed and so is the communication protocol between card and reader.

We want to evaluate the security properties of the MIFARE system. Therefore we need hardware to eavesdrop a transaction. It should also be possible to act like a MIFARE reader to communicate with the card. The Proxmark III developed by Jonathan Westhues has these possibilities[7]. Because of its flexible design, it is possible to adjust the Digital Signal Processing to support a specific protocol. This device supports both low frequency (125kHz-134kHz) and high frequency (13.56mHz) signal processing. The signal from the antenna is routed through a Field Programmable Gate Array (FPGA). This FPGA relays the signal to the microcontroller and can be used to perform some filtering operations before relaying. The software implementation allows the Proxmark to eavesdrop communication between an RFID tag and a reader, emulate a tag and a reader. In this case our tag will be the MIFARE Classic card. Despite the basic hardware support for these operations the actual processing of the digitized signal and (de)modulation needs to be programmed for each specific application. The physical layer of the MIFARE Classic card is implemented according to the ISO14443-A standard [3]. We had to implement the ISO14443-A functionality since it was not

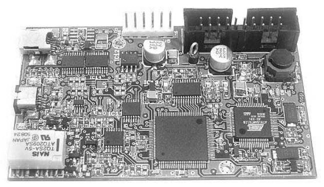


Fig. 4: The Proxmark III

[7] Hardware design and software is publicly available at http://cq.cx/proxmark3.pl

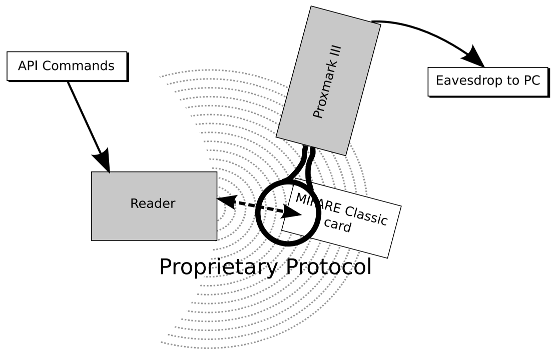


Fig. 5: Experimental Setup

implemented yet. This means we had to program both processing and generation of reader-to-tag and tag-to-reader communication in the physical layer and higher level protocol. To meet the requirements of a replay attack we added the functions 'hi14asnoop' to make traces, 'hi14areader' to act like a reader and 'hi14asim' to simulate a card. We added the possibility to send 'wrong' parity bits. This was necessary because we needed to be able to act like a real MIFARE Classic reader during encrypted communication.

4 Communication Characteristics

To find out what the MIFARE Classic communication looks like we made traces of transactions between MIFARE readers and cards. In this way, we gathered many traces which gave us some insights on the high-level protocol of MIFARE Classic. In this section we explain a trace we recorded as an example, which is shown in Figure 6. This trace contains every part of a transaction. We will refer to the sequence number (SEQ) of the messages we discuss. The messages from the reader are shown as PCD (Proximity Coupling Device) messages and from the card as TAG messages. The time between messages is shown in Elementary Time Units (ETU). One ETU is a quarter of the bit period, which equals 1.18μs. The messages are represented in hexadecimal notation. If the parity bit of a byte is incorrect, this is shown by an exclamation mark. We will discuss only the most significant messages.

Anticollision The reader starts the SELECT procedure. The reader sends `93 20` (#3), on which the card will respond with its unique identifier (#4). The reader sends `93 70` followed by the UID and two CRC bytes (#5) to select the card.

Authentication The card is in the active state and ready to handle any higher layer commands. In Section 2.4 we discussed the authentication protocol. Figure 3 shows the schematic of this mutual three pass authentication protocol. In Figure 6, messages #7 to #10 correspond to authentication.
The authentication request of the reader is `60 04 d1 3d` (#07). The first byte `60` stands for an authentication request with key A. For authentication with key B, the

```
    ETU SEQ  who bytes
---------------------------------------------------------------
      0 : 01 : PCD 26                                  }
     64 : 02 : TAG 04 00                               }
  12097 : 03 : PCD 93 20                               } Anticollision
     64 : 04 : TAG 2a 69 8d 43 8d                      }
  16305 : 05 : PCD 93 70 2a 69 8d 43 8d 52 55          }
     64 : 06 : TAG 08 b6 dd                            }
---------------------------------------------------------------
  16504 : 07 : PCD 60 04 d1 3d                         }
    112 : 08 : TAG 3b ae 03 2d                         } Authentication
   6952 : 09 : PCD c4! 94 a1 d2 6e! 96 86! 42          }
     64 : 10 : TAG 84 66! 05! 9e!                      }
---------------------------------------------------------------
 396196 : 11 : PCD a0 61! d3! e3                       }
    208 : 12 : TAG 0d                                  }
   8442 : 13 : PCD 26 42 ea 1d f1! 68!                 } Increment & Transfer
   5120 : 14 : PCD 8d! ca cd ea                        }
   2816 : 15 : TAG 06!                                 }
---------------------------------------------------------------
1349238 : 16 : PCD 2a 2b 17 97                         }
     72 : 17 : TAG 49! 09! 3b! 4e! 9e! 5e b0 06 d0!    } Read
                   07! 1a! 4a! b4! 5c b0! 4f c8! a4!   }
---------------------------------------------------------------
```

Fig. 6: Trace of a card with default keys, recorded by the Proxmark III

first byte must be `61`. The second byte indicates that the reader wants to authenticate for block 4. Note that block 4 is part of sector 1 and therefore this is an authentication request for sector 1. The last two bytes are CRC bytes.

Encrypted Communication After this successful authentication the card is ready to handle commands for sector 1. The structure of the commands can be recognized clearly. Since we control the MIFARE Classic reader we knew which commands were send. Message #11 to #15 show how an increment is performed. The increment is immediately followed by a read command (#16 and #17).

The MIFARE Classic commands of the higher level protocol consist of 4 bytes of the form `XX YY ZZ ZZ`. The first byte `XX` indicates the command type. The second byte `YY` indicates the memory address on which the command should be executed. A command is not always related to a specific memory address. The halt command (`50 00 57 cd`) illustrates this. The last two bytes `ZZ ZZ` are CRC bytes.

5 Weakness in MIFARE Classic

Nohl and Plötz [5] partially recovered the algorithm that is used to encrypt the communication between card and reader. Furthermore, they discovered that the pseudo-random generator, used to generate the nonces in the authentication, is weak. Also a dependency between the UID of the card and the key used in authentication was revealed.

During our experiments, independently, we also discovered this weakness of the pseudo-random generator by requesting many nonces from the card, at arbitrary times. This experiment showed that a 'random' nonce repeats a few times per hour. This is just by chance because Nohl and Plötz discovered that the nonce is generated by an *Linear Feedback Shift Register* (LFSR) which shifts every 9.44μs. This is exactly one bit period in the communication. Therefore a random nonce could theoretically reappear after 0.618s, if the card is queried at exactly the right time.

Without knowing the cryptographic algorithm, only an online brute force attack can be mounted, trying all possible keys in an actual authentication run between a reader and a card. Because of the communication delay, this would take 5ms for each attempt. An exhaustive key search would then take 16,289,061 days, which equals about 44,627 years.
When the cryptographic algorithm is known, an off-line brute force attack can be mounted using a few eavesdropped traces of an authentication run. Nohl and Plötz state that with dedicated hardware of around $17.000 this would take about 1 hour. For this attack to work, some known plaintext is required. Our analysis provides this plaintext.

It is however possible to attack the MIFARE Classic in another way, that does not require recovering the key. This attack, that we describe here, uses the weakness of the pseudo-random generator to recover the keystream.

6 Keystream Recovery Attack

In Section 5 we discussed over a weakness in the pseudo-random generator of the MIFARE Classic. In this section we deploy a method to recover the keystream that is used in an earlier recorded transaction between a reader and a card. As a result the keystream of the communication will be recovered. For this attack we need to be in possession of the card. The following reasons make this attack interesting:

1. Our attack provides the known plaintext necessary to mount a brute force attack on the key.
2. Using our attack we recovered details about the byte commands.
3. Using the recovered key stream we can *read* card contents without knowing the key.
4. Using the recovered key stream we can also *modify* the contents of the card without knowing the key.
5. We can spoof/clone cards.

6.1 Keystream Recovery

To recover the keystream we exploit the weakness on the pseudo-random generator. As it is this random nonce in combination with only one valid response of the reader that determines the continuation of the keystream. For this attack we need complete control over the reader (Proxmark) and access to a (genuine) card. The attack consists of the following steps:

1. Eavesdrop the communication between a reader and a card. This can be for example be in an access control system or public transport system.
2. Make sure that the card will use the same keystream as in the recorded communication. This is possible because the card repeats the same nonce in reasonable time, and we completely control the reader.
3. Modify the communication under the keystream, so that the card receives a command for which we expect known plaintext in the response.
4. When the plaintext is (partially) known the keystream for that part is recovered. This will reveal data and commands that initially were encrypted.
5. Try recovering more keystream bits by shifting commands.

The plaintext P_1 in the communication is XOR-ed bitwise with a keystream K which gives the encrypted data C_1. When it is possible to use the same keystream on a different plaintext P_2 and either P_1 or P_2 is known, then both P_1 and P_2 are revealed.

$$\left.\begin{array}{l} P_1 \oplus K = C_1 \\ P_2 \oplus K = C_2 \end{array}\right\} C_1 \oplus C_2 \Rightarrow P_1 \oplus P_2 \oplus K \oplus K \Rightarrow P_1 \oplus P_2 \tag{1}$$

The weak pseudo-random generator makes it possible to replay an earlier recorded transaction. We can flip ciphertext bits to try to modify the first command such that it gives another result. Another result gives us another plain text. The attack is based on this principle.

6.2 Keystream Mapping

The data is encrypted bit by bit. When the reader sends or receives a message the keystream is shifted the size of the message sent. This is needed for the card and reader to keep synchronized and use the same keystream bits.

The stream cipher does not use any feedback mechanism. Despite that, when we tried to reveal contents of a message sequence using a known keystream of an earlier trace, something went wrong. We recorded an increment followed by a transfer. We used this trace in a replay and changed the first command to a read command which consists of 4 command bytes and 18 response bytes. Together with the parity bits this makes it a 198 bit stream. It was a read of known plain text and therefore recovered 198 keystream bits. But when we used this key stream and tried to map it on the original trace of the increment (Figure 7) it turned out that the keystream was not in phase after the first command. The reason was the short 4-bit answer of the card.

	INCREMENT	ACK	VALUE	TRANSFER	ACK
Plaintext	c1 04 f6 8b	0a	01 00 00 00 bb 4a	b0 04 ea 62	0a
Ciphertext	4c 88 31 bc!	0a!	e2 79!2a!14 35!6f!	04!81 2d!1e!	0c!

Fig. 7: Recovering the Keystream and Commands

We do not know the internal logic of the card but the following method successfully maps the keystream on another message sequence.
Every time a data bit needs to be encrypted it is XOR-ed with the next bit in the keystream and the keystream is shifted. The last used keystream bit is also stored in register K. The `nrbits_stream` is the number of keystream bits shifted since the first command was sent. The `nrbits_data` is the number of bits sent since the start of the current message. If `nrbits_stream` $+1 \mod 9 = 0$ (8 bits shifted in the key stream), the keystream is shifted an extra bit and this bit is placed in our register K. When the current message needs to encrypt a parity bit it uses the next bit from the keystream without shifting, except when `nrbits_send` $+1 \mod 9 = 0$, then it will use the bit in register K to encrypt the parity bit.

6.3 Authentication Replay

To replay an authentication we first need a trace of a successful authentication between a genuine MIFARE reader and card. An example of an authentication followed by one read command is shown below.

```
1  PCD   60  03  6e  49
2  TAG   e0  92  93  98
3  PCD   ad  e7  96! 48! 20! 22  df  93
4  TAG   bf  06  91! 82
5  PCD   b5! 05! 47  3f
6  TAG   3f  14! 4f  e9! 86  38! 96! 85 3e!
         f3  e3! 3d! eb! 2b! a2  d4  dd 76!
```

Figure 3 shows the schematic of this trace. After this recorded authentication between card and reader, we make sure that the memory of the card is not modified. This ensures that when the memory of the card is read it will return the same plaintext. Now we will act like a MIFARE reader and try to initiate the same authentication. In short:

1. We recorded a trace of a successful authentication between a genuine card and reader.
2. We send authentication requests (#1) until we get a nonce that is equal to the one (#2) in the original trace.
3. We send the recorded response (#3) to this nonce. It consists of a valid response to the challenge nonce and challenge from the reader.
4. We retrieve the response (#4) to the challenge from the card.
5. Now we are at the point we where we could resend the same command (#5) or attempt to modify it.

After step 4 the card is in a state where we have successfully authenticated for (in this case) sector 0 (block 3). Now it expects a command for this sector. If we send the same command we recorded earlier, we get the same encrypted response as in the original trace. Therefore the keystream is the same.

6.4 Reading a Sector

We will show that it is possible to read sector 0 from a card without knowing the key. We only need one transaction between a genuine MIFARE reader and card. Every MIFARE Classic card has some known memory contents. The product information published by NXP [6] gives this information.
When a sector trailer is read the card will return logical '0's instead of key A because key A is not readable. If key B is not readable the card also returns logical '0's. It depends on the access conditions if key B is readable or not. The access conditions

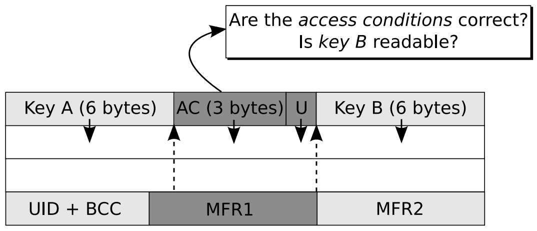


Fig. 8: Recovering Sector 0

can be recovered by using the manufacturer data. Block 0 contains the UID and BCC followed by the manufacturer data. The UID and BCC cover 5 bytes and are known. The remaining 11 bytes are covered by the manufacturer data. Some investigation on different cards (MIFARE Classic 1k and 4k) revealed that the first 5 bytes of the manufacturer data almost never change. These bytes (MFR1) cover the positions of the access conditions (AC) and the unkown byte U, as shown in Figure 8. This means that the keystream can be recovered using the known MFR1 bytes by reading block 0 and block 3 (sector trailer) subsequently. Remember that the access conditions are stored twice in 3 bytes. Once inverted and once non-inverted. This way it is easy to detect if we indeed revealed the access conditions. The unknown byte U can be in any state when the card leaves the manufacturer but appears to be often `00` or `69`.

The access conditions tell us whether key B is readable or not. In many cases key B is not readable. In the Netherlands the MIFARE Classic 4k card is used in the public transport system. The first 5 bytes of the manufacturer data (MFR1 in Figure 8) recovered the access conditions for sector 0. Because the access conditions for the sector trailer define key B as not readable, we know the plaintext is zeros. Hence the whole sector trailer was revealed and therefore the contents of the whole sector 0 were revealed as well.

6.5 Proprietary Commands

We used a card in transport configuration with default keys and empty data blocks to reveal the encrypted commands used in the high-level protocol. All the commands send by the reader consist of a command byte, parameter byte and two CRC bytes. We made several attempts to reveal the command by modifying the ciphertext of this command. The way to do this is to assume we actually know the command. With this 'knowledge' we XOR the ciphertext which gives us the keystream. To check if this is indeed the correct keystream we XOR it with a new command for which we know the response. If we guessed the initial command right the response of the card will be that known response. This method reveiled the commands shown in Figure 9. A logical step would now be to replay the same authentication again and try to execute a command that will return only an ACK or NACK. Because this would result in a shift in the keystream. There will be enough known keystream left to construct a new 'read sector trailer' command. This attempt does only work if a decrement, increment or transfer is allowed. These are the only commands that are shorter than the read. We can only send valid commands because otherwise the protocol aborts. Except from halt, all commands take an address byte of a block as

Authentication			
READER	*CARD*	*READER*	*CARD*
60 YY* Using KeyA	4-byte nonce	8-byte response	4-byte response
61 YY* Using KeyB	4-byte nonce	8-byte response	4-byte response
Data			
READER	*CARD*	*READER*	
30 YY* Read	16 data bytes*		
A0 YY* Write	ACK / NACK	16 data bytes*	
Value blocks			
READER	*CARD*	*READER*	*READER*
C0 YY* Decrement	ACK / NACK	4-byte value*	Transfer
C1 YY* Increment	ACK / NACK	4-byte value*	Transfer
C2 YY* Restore	ACK / NACK	4-byte value*	Transfer
B0 YY* Transfer	ACK / NACK		
Other			
READER			
50 00* Halt			

YY = block address
* = Followed by two CRC bytes

Card responses (ACK / NACK)

A (1010)	ACK
4 (0100)	NACK, not allowed
5 (0101)	NACK, transmission error

Fig. 9: Command set of MIFARE Classic

parameter (Figure 9). The read command returns 16 data bytes and 2 CRC bytes. On a write command the card returns a 4-bit ACK. Then the card is ready to receive 16 data bytes to write and 2 CRC bytes.
The decrement, increment and restore commands all follow the same procedure. The card response is a 4-bit ACK which means it is expecting a 4-byte value (together with 2 CRC bytes). For the restore this value should be send but is not used. Now the value can be send as `YY YY YY YY ZZ ZZ` where `YY` are the value bytes and `ZZ` the CRC bytes.
Finally, a transfer command can be send to transfer the result of one of the previous commands to a memory block.

Of course the card will not reply with an ACK if a command is not allowed. The 4-bit ACK is `a`. When a command is not allowed the card will send `4` and when a transmission error is detected it will send `5`. If the command is of the wrong length the card does not even give a response at all. The protocol stops on every mistake or disallowed command.

7 Conclusions & Recommendations

We have implemented a successful attack to recover the keystream of an earlier recorded transaction between a genuine MIFARE Classic reader and card.

We used a MIFARE Classic reader in combination with a 'blank' card with default keys to recover the byte commands that are used in the proprietary protocol. Knowing the byte commands and a sufficiently long keystream allowed us to perform any operation as if we would be in possession of the secret key.

We managed to read *all* memory blocks of the sector zero of the card, without having access to the secret key. In general, we were able to read *any* sector of the memory of the card, provided that we know *one* memory block within this sector. Moreover, after recording a valid transaction on any sector, we were able to read the first 6 bytes of any block in that sector and also the last 6 bytes if key B is read only.

For short term improvements we recommend not to use sector zero to store secret information. Configure key B as readable and store random information in it. Do not store sensible information in the first 6 bytes of any sector. Use multiple sector authentications in one transaction to thwart attackers in an attempt to recover plaintext. This is only helpful when value block commands are not allowed. Value block commands are shorter than a read command and will enable a shift of the keystream. Another possibility, that might be viable for some applications, is to employ another encryption scheme like AES in the back office, and store only encrypted information on the tags.

On the long term these countermeasures will not be sufficient. The MIFARE Classic card has a closed design. Security by obscurity has shown several times that at some point the details of the system will be revealed compromising security [4]. Therefore we recommend to migrate to more advanced cards with an open design architecture.

References

1. Klaus Finkenzeller. *RFID Handbook*. John Wiley and Sons, 2nd edition, 2003.
2. J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. Wichers Schreur. Crossing borders: Security and privacy issues of the european e-passport. In Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shinichi Kawamura, editors, *Advances in Information and Computer Security. International Workshop on Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 152–167. Springer Verlag, 2006.
3. ISO/IEC 14443. *Identification cards - Contactless integrated circuit(s) cards - Proximity cards*, 2001.
4. Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX, 1983. pp. 5–38, Jan. 1883, and pp. 161–191, Feb. 1883.
5. Karsten Nohl and Henryk Plötz. Mifare, little security, despite obscurity. Presentation on the 24th Congress of the Chaos Computer Club in Berlin, December 2007.
6. NXP Semiconductors. *MIFARE Standard 4kByte Card IC functional specification*, February 2007.