BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

MIFARE CLASSIC EMULATION ON NFC-A TAG PERIPHERAL

EMULACE KARET MIFARE CLASSIC NA NFC-A PERIFERII

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAKUB LUŽNÝ**
AUTOR PRÁCE

SUPERVISOR **Ing. ONDŘEJ HUJŇÁK**
VEDOUCÍ PRÁCE

BRNO 2021

Brno University of Technology
Faculty of Information Technology

Department of Intelligent Systems (DITS) Academic year 2020/2021

Master's Thesis Specification

24139

Student:	**Lužný Jakub, Bc.**
Programme:	Information Technology Field of study: Computer and Embedded Systems
Title:	**MIFARE Classic Emulation on NFC-A Tag Peripheral**
Category:	Security

Assignment:

1. Study RFID technology and NFC standards. Describe MIFARE Classic smartcards and their compliance with these specifications.
2. Propose a way to emulate MIFARE Classic cards using a NFC-A tag peripheral.
3. Implement the proposed solution.
4. Verify the implemented solution (anti-collision, authentication and read, write commands) on Chameleon Mini and other ISO14443 reader. Compare your solution to capabilities of Chameleon, discuss its limits and possible extensions.

Recommended literature:

- Bobčík Martin. *Bezpečnostní analýza karet Mifare Classic.* Bakalářská práce.
- RANKL, Wolfgang a Wolfgang EFFING. *Smart Card Handbook*. Chichester, UK: John Wiley & Sons, 2010. DOI: 10.1002/9780470660911. ISBN 9780470743676.
- PARET, Dominique a Roderick RIESCO. *RFID and contactless smart card applications*. Chichester: John Wiley, 2005, 330 s. ISBN 0-470-01195-5.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Hujňák Ondřej, Ing.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	November 1, 2020
Submission deadline:	May 19, 2021
Approval date:	November 11, 2020

Abstract

The goal of this thesis is to implement emulation of a MIFARE Classic tag on a NFC-A tag peripheral embedded in a microcontroller. To implement the solution, nRF52832 microcontroller from Nordic Semiconductor was used. Basic MIFARE Classic command set was implemented (authentication, read, write). The implementation was tested against multiple readers, using different applications. The result was compared to existing solutions, such as Proxmark and Chameleon Mini.

Abstrakt

Cílem této práce bylo implementovat emulaci tagu MIFARE Classic na NFC-A periferii vestavěné v mikrokontroléru. K implementaci řešení byl použit mikrokontrolér nRF52832 vyráběný firmou Nordic Semiconductor. Byla implementována základní sada příkazů MIFARE Classic (autentizace, čtení, zápis). Implementace byla otestována s několika čtečkami, za pomoci různých aplikací. Výsledné řešení bylo porovnáno s existujícími, jako je Proxmark a Chameleon Mini.

Keywords

RFID, NFC, security, nRF52, MIFARE, contactless cards, smart cards.

Klíčová slova

RFID, NFC, bezpečnost, nRF52, MIFARE, bezkontaktní karty, chytré karty.

Reference

LUŽNÝ, Jakub. *MIFARE Classic Emulation on NFC-A Tag Peripheral.* Brno, 2021. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Ondřej Hujňák

Rozšířený abstrakt

RFID tagy a bezkontaktní karty jsou v dnešní době velmi rychle se rozšiřující technologie, přestože jejich princip je používán již po desítky let. RFID nepředstavuje žádný konkrétní standard, jedná se spíše o obecný termín, který je implementován mnoha různými standardy. Ty se dělí do třech hlavních prodů podle rádiových frekvencí, které používají ke komunikaci. Tagy jsou dělený na nízkofrekvenční (125-134 kHz), vysokofrekvenční (13,56 MHz) a tagy používající velmi vysoké frekvence (860 nebo 960 MHz). Tato práce se zabývá zejména standardy pro komunikaci na vysoké frekvenci, protože karta MIFARE Classic je na nich založená. Jsou popsány tři z nich - ISO 14443 pro komunikaci na blízké vzdálenosti (10 cm), ISO 15693 pro komunikaci na větší vzdálenosti (1 m) a také standard FeliCa, který je alternativou k ISO 14443 a je používán téměr výhradně v Japonsku. Standard ISO 14443 se navíc dělí na typ A a B, které se odlišují jak komunikačním protokolem, tak i antikolizním algoritem, který používají čtecí zařízení pro výběr karty ve svém elektromagnetickém poli, se kterou chtějí komunikovat.

Technologie NFC je standardem pro komunikaci mezi zařízeními na velmi krátké vzdálenosti. Je dnes běžnou součástí chytrých telefonů. Tato technologie je kompatibilní se standardy ISO 14443 a FeliCa, což umožňuje chytrým telefonům komunikovat s RFID tagy a bezkontaktními kartami.

MIFARE Classic je poměrně starý typ bezkontaktní karty, vyráběné firmou NXP od roku 1995. Kommunikace s touto kartou probíhá pomocí protokolu ISO 14443 Typu A, přičemž se nepoužívá standardní transportní protokol, ale je implementován vlastní. Karta v sobě obsahuje paměť EEPROM o velikosti 1 KiB nebo 4 KiB. Tato paměť je rozdělena na sektory, kdy každý sektor má několik datových bloků o velikosti 16 B. Poslední blok sektoru v sobě uchovává dva 48bitové klíče A a B, pomocí kterých se ke kartě autentizuje čtecí zařízení a také příznaky oprávnění, jaké operace je možné nad jednotlivými datovými bloky provádět s těmito klíči. Komunikace se čtecím zařízením je šifrována proudovou šifrou Crypto1, která byla pro tuto kartu speciálně navržena a její princip byl dlouho utajován. To se změnilo v roce 2008, kdy byl šifrovací algoritmus prolomen. Bylo k tomu použito metody zpětné rekonstrukce obvodu pomocí analýzy křemíkových vrstev použitého čipu.

Od té doby se začaly objevovat zařízení schopná emulovat tyto karty. Nejznámější z nich jsou Proxmark a Chameleon Mini. Zatímco Proxmark využívá FPGA k modulaci a demodulaci RFID signálu, v případě Chameleonu Mini je řešení kompletně softwarové, využívající pouze mikrokontroléru a několika diskrétních součástek. Obě řešení následně implementují v mikrokontroléru vyšší vrtsvy ISO 14443 protokolu, na kterém je karta postavena, a samotnou logiku karty MIFARE Classic, včetně šifrovacího algroritmu Crypto1.

Cílem této práce je navrhnout a implementovat emulaci RFID karty typu MIFARE Classic pomocí NFC-A periferie mikrokontroléru. K implementaci byl vybrán mikrokontrolér typu nRF52832 od firmy Nordic Semiconductor, který disponuje touto periferií. Jedná se o čip, který má vestavěné rádio pro technologii Bluetooth Low Energy a je velmi rozšířen v oblastech, kde se tato technologie používá, jako jsou např. chytré hodinky.

Řešení bylo implementováno na vývojovém přípravku nRF52 DK, a to včetně řízení oprávnění k datovým blokům. Implementace byla omezena na variantu MIFARE Classic 1K s 1 KiB paměti, a byly implementovány pouze základní příkazy pro autentizaci, čtení a zápis. Byl kladen důraz na čitelnost kódu a přenositelnost implementace. Proto byla rozdělena do několika modulů, přičemž pouze jeden z nich je specifický pro konkrétní implemenaci NFC periferie.

Výsledná implementace byla otestována se dvěma čtecími zařízeními. Jedním z nich byl chytrý telefon s operačním systémem Android, kde byly k otestování použity dvě různé ap-

likace. Druhým čtecím zařízením byla USB čtečka připojená k počítači, s volně dostupným softwarem. Ve všech případech byla implementace kompatibilní, vyskytly se však problémy s autentizací k více sektorům v rámci jednoho komunikačního sezení. Tyto problémy však nastávaly pouze při komunikaci s chytrým telefonem, nikoliv s USB čtečkou.

Implementované řešení bylo porovnáno s existující konkurencí. Není natolik všestranné, jako vyzrálá konkurence v podobě Chameleonu Mini nebo Proxmarku, je však energeticky úspornější, levnější a vejde se do menších fyzických rozměrů.

Na závěr bylo navrženo několik cest, kterými by se projekt dále mohl vydat. Jednou z nich je rozšíření implementace MIFARE Classic o další příkazy a podporu verze se 4 KiB paměti. Druhou cestou je zprovoznění Bluetooth rádia a vytvoření aplikace pro chytrý telefon, která by umožnila pracovat s obsahem emulované karty. Poslední, třetí cestou, je integrace řešení do nějakého existujícího zařízení, jako jsou například chytré hodinky.

MIFARE Classic Emulation on NFC-A Tag Peripheral

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Ondřej Hujňák. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

....
Jakub Lužný
May 19, 2021

Acknowledgements

I would like to thank my supervisor Ing. Ondřej Hujňák for his guidance, constructive feedback and advices on writing a thesis.

Contents

Chapter 1

Introduction

Contactless cards or tags are used in many areas of everyday life, for example, in public transport, event tickets, building access systems or even for payments. Many of the cards are based on the MIFARE Classic standard developed by NXP. While the cards are small and cheap, they have only limited flexibility in terms of using one card for multiple purposes.

The goal of this thesis is to provide a solution to emulate the MIFARE Classic tag using a microcontroller commonly used in gadgets or wearables, so it can be emulated using these devices without added cost. When the tag is emulated by a smart gadget, many advanced features can be implemented, such as allowing the user to switch between multiple emulated cards, or even switching them automatically based on the location of the user.

MIFARE Classic is already quite an old standard, debuting in 1995. It is proprietary, and no specification that would allow alternative implementation was ever released by the manufacturer. The situation changed in 2008 when researchers reverse-engineered the chip silicon. Since then, alternative implementations started to show up. The most famous ones are in the Proxmark and Chameleon Mini tools for RFID analysis. In the case of Proxmark, an FPGA is used for modulation and demodulation of the signal, which makes the device large and expensive. In the case of Chameleon Mini, it is implemented purely in software with just a few discrete components. In this approach, the software is responsible for the precise timing, making integration into an existing gadget problematic.

In 2016, Nordic Semiconductor released the nRF52 microcontroller series, successor to the very popular nRF51, featuring an ARM core and integrated Bluetooth Low Energy radio. It quickly became popular as well, both in professional gadgets and in the open-source world. One of the new features over the older series is that the NFC-A tag peripheral was added to allow out-of-band Bluetooth pairing with smartphones supporting NFC. In this thesis, the NFC peripheral will be used to implement the emulation of MIFARE Classic.

The thesis will first dive into the depths of RFID cards (chapter 2). Then, in chapter 3, the NFC technology that allows smartphones to interact with RFID tags will be described. Chapter 4 will introduce the proprietary MIFARE standard. The nRF52832 microcontroller and its NFC peripheral will be presented in chapter 5. The steps behind the implementation, its structure and features are in chapter 6. The testing process and its results are evaluated in chapter 7. Finally, the result, its possible extensions and comparison to the existing solutions is discussed in chapter 8.

Chapter 2

RFID

Radio Frequency Identification (RFID) is a technology that uses an electromagnetic field to identify and track tags that contain electronically stored information. The communication between a reader (base station) and a tag (transponder) is done in a wireless, „contactless“ way over short distances.

2.1 History

Although RFID is a technology that undergoes a significant expansion and integration in concurrent applications, its key concepts are in use already for many decades. It is believed that the first use of them was already in the 1940s during World War II with the invention of IFF (Identification of Friend and Foe). A transmitter was put on every British plane. Upon receiving a signal from a radar station, it transmitted the signal back to identify the plane as friendly.

2.2 Standards

RFID is not a name of an exact technical specification; it is more of a term to describe a family of different technical implementations. They also communicate at various frequencies and thus offer various range and communication speed. The most important frequencies are described in table 2.1.

Frequency	125-134 kHz	13.56 MHz	860/960 MHz
Band	LF	HF	UHF
Read distance	10 cm	10 cm - 1 m	1 m - 12 m
Data rate	upto 8 kbps	upto 848 kbps	upto 640 kbps

Table 2.1: RFID frequencies and their properties [20]

In this thesis, only standards operating in the HF band will be covered, as it is the band used by MIFARE Classic and similar tags.

2.2.1 ISO 14443

ISO 14443 (*Identification cards – Contactless integrated circuit cards – Proximity cards*) [5] is the most widespread standard of HF RFID. Its application includes contactless bank cards, public transport passes and door access system cards. The standard is divided into four parts:

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

Part 1 - Physical characteristics

Part 1 defines the physical properties of the card. It should be the same size as ID-1 card defined in ISO 7810. It also defines the operating conditions of the cards, e.g. temperature and electric field immunity. It also defines the naming of the involved devices:

- Proximity card (PICC) for the card
- Proximity coupling device (PCD) for the reader/writer

Part 2 - Radio frequency power and signal interface

The power supply to the card is provided by RF field of the reader at the frequency of 13.56 MHz. In order to receive the energy effectively, the cards have to include a proportionally large antenna.

During the development of the standard, the involved parties were not able to agree on one common communication interface. That led to accepting two different ways of transmission - Type A and Type B. The difference between them will be described and is also summarized in table 2.2. A contactless card has to implement only one of them. A reader should implement both variants of the standard, so it is able to support all cards. To do so, the reader has to switch between the modes periodically while waiting for a card.

Type A cards use 100% ASK modulation with modified Miller coding to transfer data from the reader to the card. The blanking interval (the time when the RF field is turned off to modulate the signal) is just 2-3 μs. To transfer data from the card to the reader, load modulation procedure with a subcarrier is used. Frequency of the subcarrier is defined to 847 kHz (13.56 MHz / 16). The subcarrier is modulated by on/off keying (OOK) of the Manchester coded data stream.

Type B cards use 10% ASK modulation with simple NRZ coding to transfer from the reader to the card. To transmit from the card to the reader, a subcarrier of frequency 847 kHz is also used, but it is modulated by 180° phase shift keying (BPSK) of NRZ coded bitstream.

Type	Type A	Type B
Bit rate during initialisation and anticollision	106 kbps	106 kbps
Modulation (PCD to PICC)	ASK 100%	ASK 10%
Data encoding (PCD to PICC)	Modified Miller	NRZ
Modulation (PICC to PCD)	Load Modulation, OOK	Load Modulation, BPSK
Data encoding (PICC to PCD)	Manchester	NRZ-L

Table 2.2: Card types defined in ISO 14443 Part 2

Part 3 - Initialization and anticollision

Part 3 defines how the initialization of the card is done and how the anticollision procedure works[16]. It is a procedure that allows multiple cards to be present in the operating range and defines an algorithm, how the reader can choose a particular card to communicate with without being interfered from the others. The standard also defines the frame format used and the timing requirements.

As in Part 2, this specification is also split into two card types, Type A and Type B. When a **Type A** card gets into the proximity of a reader and sufficient voltage is available, the card begins to operate. At that point, the card is in the `IDLE` mode. If the card receives a valid `REQA` command (Request-A) in this mode, then an `ATQA` response (Answer to Request A) is sent back to the reader. At this moment, the card gets into the `READY` state. So, now the reader knows that there is at least one card present in its proximity and begins the anticollision procedure. In this case, it is performed by an algorithm called *binary search tree*.

The process is done by the `SELECT` command. It has two attributes - first is the `NVB` (number of valid bits), being actually length of the selected prefix and the second is the actual selected prefix. When the card receives this frame and the prefix of length `NVB` matches its ID, it starts to transmit the rest of its ID, right after the reader has transmitted the prefix. When the reader detects a collision in the received ID, it transmits a new `SELECT` command, with `NVB` set to the first colliding bit and selecting a value of the bit that collided (either 0 or 1).

In practice, the first `SELECT` command is sent with `NVB = 0` and the process continues until a `SELECT` command with `NVB = 40` (the ID is 4 bytes long and there is one byte of checksum appended, thus 40 bits). When a card receives this command with its ID, it confirms its selection by transmitting an `SAK` (SELECT Acknowledge) and transfers to the `ACTIVE` state.

The procedure above works for cards with 4 bytes long UID. However, there are also cards with 7 or 10 bytes long UID. In that case, after being selected in the first round, the card signals in the `SAK` response that the UID is not yet complete and the reader start next round of anticollision with the `SELECT` command. To differ between the rounds, there are actually three different `SELECT` commands with different bit representations.

Type B cards use a different algorithm called *slotted ALOHA procedure*. That is a more complicated approach and will not be described here.

Part 4 - Transmission protocol

Part 4 defines the protocol that is used to communicate with the card after the initialization and anticollision process. In case of a Type A card, additional configuration of the protocol has to be transferred (e.g. possible baud rates). In case of Type B card, this configuration was already transferred during the anticollision phase.

The Type A card may not use this protocol and use a custom one instead, which is the case of the MIFARE Classic card. The `SAK` reply contains information on whether the standard protocol or a proprietary one is used. If the standard protocol is used, the reader sends a `RATS` command (Request to Select Acknowledge). The card then answers with `ATS` (Answer to Select). These messages contain the configuration of the protocol.

2.2.2 ISO 15693

ISO 15693 (*Cards and security devices for personal identification — Contactless vicinity objects*) [6] is another common standard for RFID in the HF band. It specifies *vicinity* cards, which are designed for longer operating distance than *proximity* cards – up to 1 meter.

The standard is split into 3 parts. Part 1 specifies Physical characteristics and is similar to the first part of ISO 14443. Part 2 defines Air interface and initialization and the last part is about Anticollision and transmission protocol.

2.2.3 FeliCa

FeliCa (*Felicity Card*) is another standard for contactless smart cards developed by Sony [15]. It was proposed for ISO 14443 as Type C, but it was rejected. It complies with Japan standard JIS X6319-4.

It is commonly used for payments and public transport in Japan and some other Asian countries, as well as on university campuses in the United States, but it is rare in Europe.

Chapter 3

Near Field Communication (NFC)

Near Field Communication (NFC) is a wireless data transfer technology that enables transfer between two devices, similar to e.g. Bluetooth. However, some of its characteristics also make it related to RFID systems.

The transmission between the devices operates in the HF band at a frequency of 13.56 MHz. The transmission range is typically around 10 cm, hence the name near-field. The NFC interface in the device has both a transmitter and a receiver with a shared antenna, that is alternated between the transmitter and the receiver during operation.

During communication between two NFC devices, each device has its own function – it is either an NFC initiator (master), or NFC target (slave). Communication is always started by the initiator, that generates the RF field that can power a passive target. There are also two communication modes: active mode and passive mode. They are illustrated in figure 3.1.

3.1 Passive mode

In passive mode, the initiator provides the RF field that can power the target. The initiator transmits by modulating data on the field. However, after transmitting its own data block, the initiator leaves the RF field on, so that the target can transfer its data block using load modulation, as in RFID systems.

Using this mode, the NFC device is also able to communicate with compatible transponders (e.g. according to ISO 14443), that the NFC initiator is able to supply with power, and the target is able to communicate back using load modulation. This enables NFC-equipped electronic devices, such as smartphones, to read and write on RFID smart cards. That is why this option is called "reader-emulation mode".

If an NFC-equipped device is put into the proximity of an RFID reader (e.g. ISO 14443 based), the device is also able to adopt the function of an NFC target and perform load modulation on the RF field generated by the reader. When this is performed, the NFC device behaves as a contactless smart card from the reader's perspective, and thus this option is called "card-emulation mode".

3.2 Active mode

Unlike the passive mode, where the initiator and target functions are fixed for the whole communication window, in passive mode, they alternate between the devices. After the

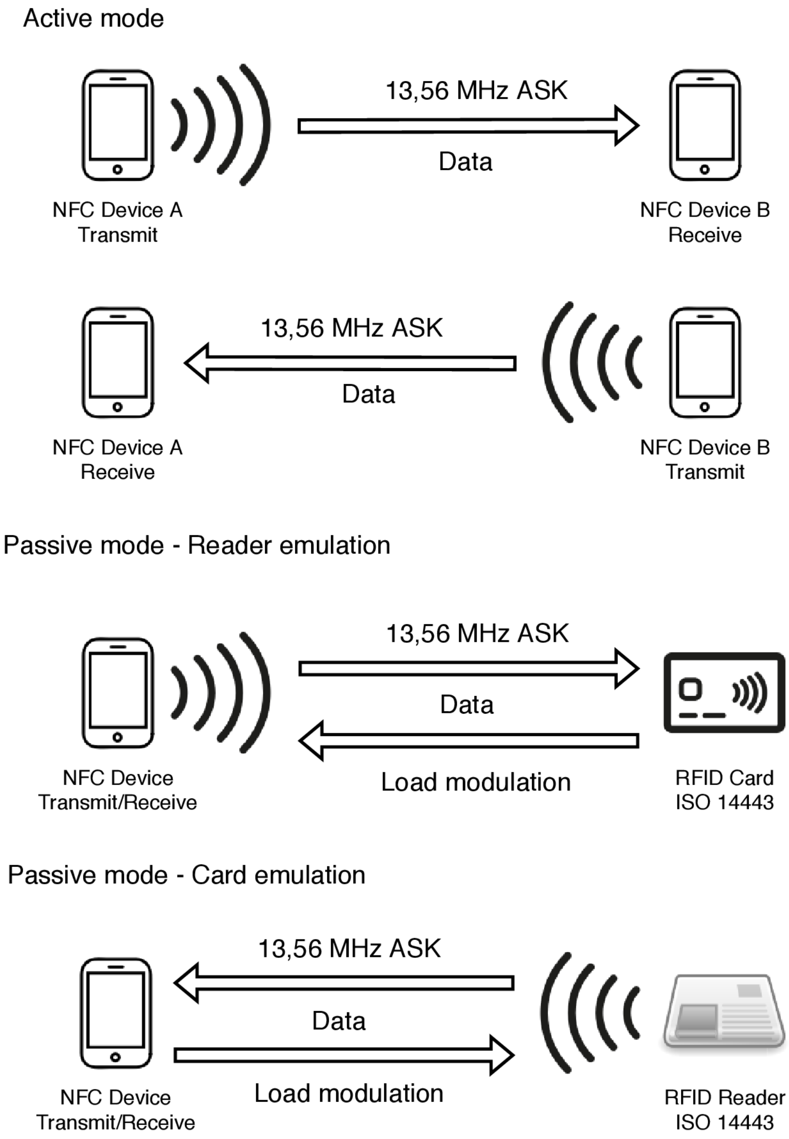


Figure 3.1: NFC communication modes

initiator sends its data block to the target, it starts to act as a target, and the other device starts the transfer as an initiator. That means the currently transmitting device is always generating the RF field and no load modulation is used.

3.3 Standardization

ISO 18092 (*Near Field Communication — Interface and Protocol (NFCIP-1)*) [2] is the underlying standard of NFC. It is based on ISO 14443 and extends it with the Active Communication Mode. It also defines its own transport protocol that can be used instead of ISO 14443-4.

The newer standard ISO 21481 (*Near Field Communication Interface and Protocol - 2 (NFCIP-2)*) [1] adds support for vicinity cards defined in ISO 15693, and defines new naming of the modes:

- NFC mode – device operates as specified in ISO 18092
- PICC mode – device operates as Type A or Type B Proximity Integrated Circuit Card specified in ISO 14443
- PCD mode – device operates as a Proximity Coupling Device specified in ISO 14443
- VCD mode – device operates as a Vicinity Coupling Device specified in ISO 15693

The specifications of the higher-level protocols are maintained by NFC Forum. They include a common data format called NFC Data Exchange Format (NDEF), which can be used to exchange objects between devices or store them in a tag. Objects can range from short data, e.g. URLs, to complete file transfers.

3.4 Tag types

NFC Forum also defines currently five different tag types that can be used to store NDEF messages. They are compared in table 3.1.

Tag Type	Type 1	Type 2	Type 3	Type 4	Type 5
NFC technology	NFC-A	NFC-A	NFC-F	NFC-A, NFC-B	NFC-V
Example product	Broadcom Topaz	NXP MIFARE Ultralight	Sony FeliCa	NXP DESFire	Texas Instruments Tag-it HF-I
Memory size	454 bytes	48 - 1904 bytes	1 - 9 KiB	2 - 144 KiB	32 - 256 bytes
Underling specification	ISO 14443-3A	ISO 14443-3A	JIS 6319-4	ISO 14443-4 A/B	ISO 15693

Table 3.1: NFC tag types

3.5 Smartphones

The first smartphone featuring an NFC interface was Samsung Nexus S in 2010. Nowadays, most of the smartphones on the market support NFC, as it is widely used by customers for contactless payments.

3.5.1 Google Android

Android supports NFC since version 2.3, which was released together with Samsung Nexus S smartphone. In this version, only NFC active mode (to transfer data between NFC devices) and "reader emulation mode" (to read/write tags) were supported.

In Android version 4.4 released in 2013, the Host-based card emulation (HCE) feature was added to allow the "card emulation mode". However, this feature is limited to emulation of cards based on ISO 14443-4 Type A (support for Type B is optional) transport protocol using ISO 7816-4 APDUs. That is sufficient for emulation of contactless bank cards, but does not allow emulation of many common cards not using ISO 14443-4 transport protocol, e.g. MIFARE Classic or MIFARE Ultralight.

Another important limitation of the HCE mode is that the card UID reported to the reader during the anticollision phase is randomly generated every time and that behavior cannot be changed. It is a huge drawback, as many door access systems based on contactless cards use only this UID to authorize the card, and the possibility to change or at least fix the UID would allow usage of the smartphone for access instead of the card.

3.5.2 Apple iOS

Apple's devices got into the NFC world much later. The first iPhone to feature NFC hardware was iPhone 6 released in 2014, together with iOS 8. However, the only application that could use it was Apple Pay, and there was no public API to access the NFC hardware at all.

This changed with the release of iOS 11 in 2017. A new API called Core NFC was introduced, and it allowed to read NDEF formatted NFC tags. It was not possible to write the tags or get any lower level information, like UID of the tag. The API is supported on iPhone 7 and newer.

iOS 13 released in 2019 enabled even more application by allowing to also write the NDEF formatted tags. Furthermore, it also allowed the interaction with protocol specific tags, such as ISO 15693, FeliCa or MIFARE. It also allows reading the UID of the tag.

Currently, there is still no public API to support Host-card emulation. That is probably because Apple does not want to get any competitors of its own Apple Pay application.

Chapter 4

MIFARE

MIFARE (shortcut from Mikron Fare Collection) is a proprietary contactless smart card standard created by an Austrian company called Mikron in 1994. Mikron was later acquired by Philips in 1995 and then spun off into NXP Semiconductors in 2006. It was originally developed for automated fare collection in public transport, but it was later used in many other applications.

There were more types of these card developed since the debut of MIFARE Classic.

MIFARE Ultralight[8], introduced in 2001, is based on Classic, but it has less memory (in the range of 40-128 bytes) and does not offer any cryptographic features. They are so inexpensive that they can be used as disposable tickets for events.

MIFARE Plus[13] was released in 2008 as drop-in replacement for Classic. It supports ISO 14443 Part 4 transport protocol and allows AES encryption, while staying compatible with MIFARE Classic.

MIFARE DESFire[12], introduced in 2002, is a more advanced card, containing a microprocessor and an operating system that allows for simple directory and files structure. It included 3DES encryption from the beginning, with later variants also supporting AES.

4.1 MIFARE Classic

It is basically an EEPROM with 1 or 4 kilobytes of data storage, including the reserved areas for the keys and configuration [7][3] based on ISO 14443 Type A up to Part 3, using its own transport protocol on top of it (incompatible with ISO 14443 Part 4). It uses a proprietary Crypto1 algorithm to secure the communication between the card and the reader. Crypto1 was cracked in 2009 and it is recommended to switch to newer, safer cards by the manufacturer.

4.1.1 Memory structure

The EEPROM memory of the card is split into sectors. Each sector contains 4 or 15 blocks, each of 16 bytes (see figure 4.2). Every block of the sector is used for data, except the last one. The last block of the sector is called the sector trailer and is used to store two 48-bit access keys, key A and key B, and the access condition bits. There are three access condition bits for every block, named `C1-C3`. Every bit is stored twice, in non-inverted and inverted form (see Figure 4.1). In case the inverted and non-inverted form differ, the whole sector is irreversibly blocked.

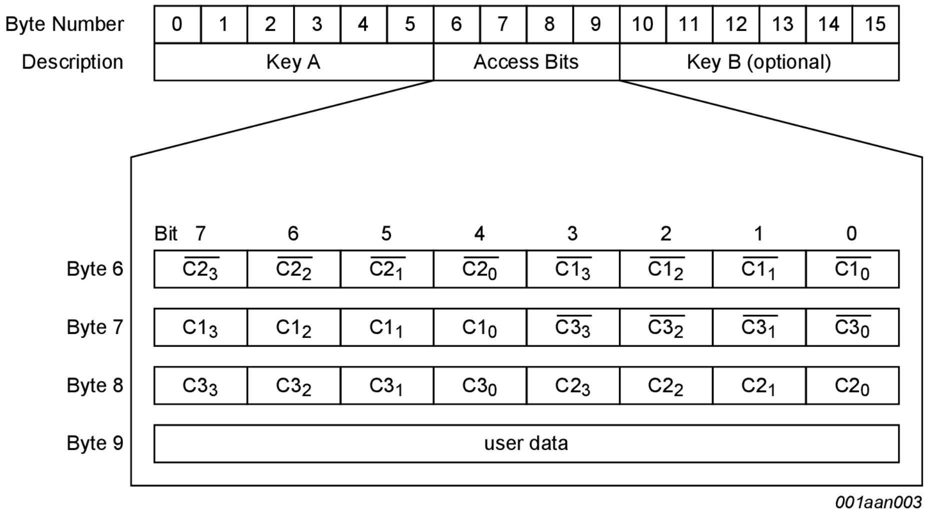


Figure 4.1: Structure of the MIFARE Classic sector trailer block. Taken over from [7]

For data blocks, access conditions define which key and command combinations are allowed on the block (table 4.1). For the sector trailer block, access conditions have different meaning. They define which key is allowed to change the access conditions, or write the key bits (table 4.2). In some cases, key A is allowed to read the key B from the sector trailer. In that case, key B cannot be used for authentication. That allows the key bits to be used to store user data.

Access bits			Access condition for			
C1	C2	C3	Read	Write	Increment	Decrement, transfer, restore
0	0	0	key A|B	key A|B	key A|B	key A|B
0	1	0	key A|B	never	never	never
1	0	0	key A|B	key B	never	never
1	1	0	key A|B	key B	key B	key A|B
0	0	1	key A|B	never	never	key A|B
0	1	1	key B	key B	never	never
1	0	1	key B	never	never	never
1	1	1	never	never	never	never

Table 4.1: Access conditions for data blocks [7]

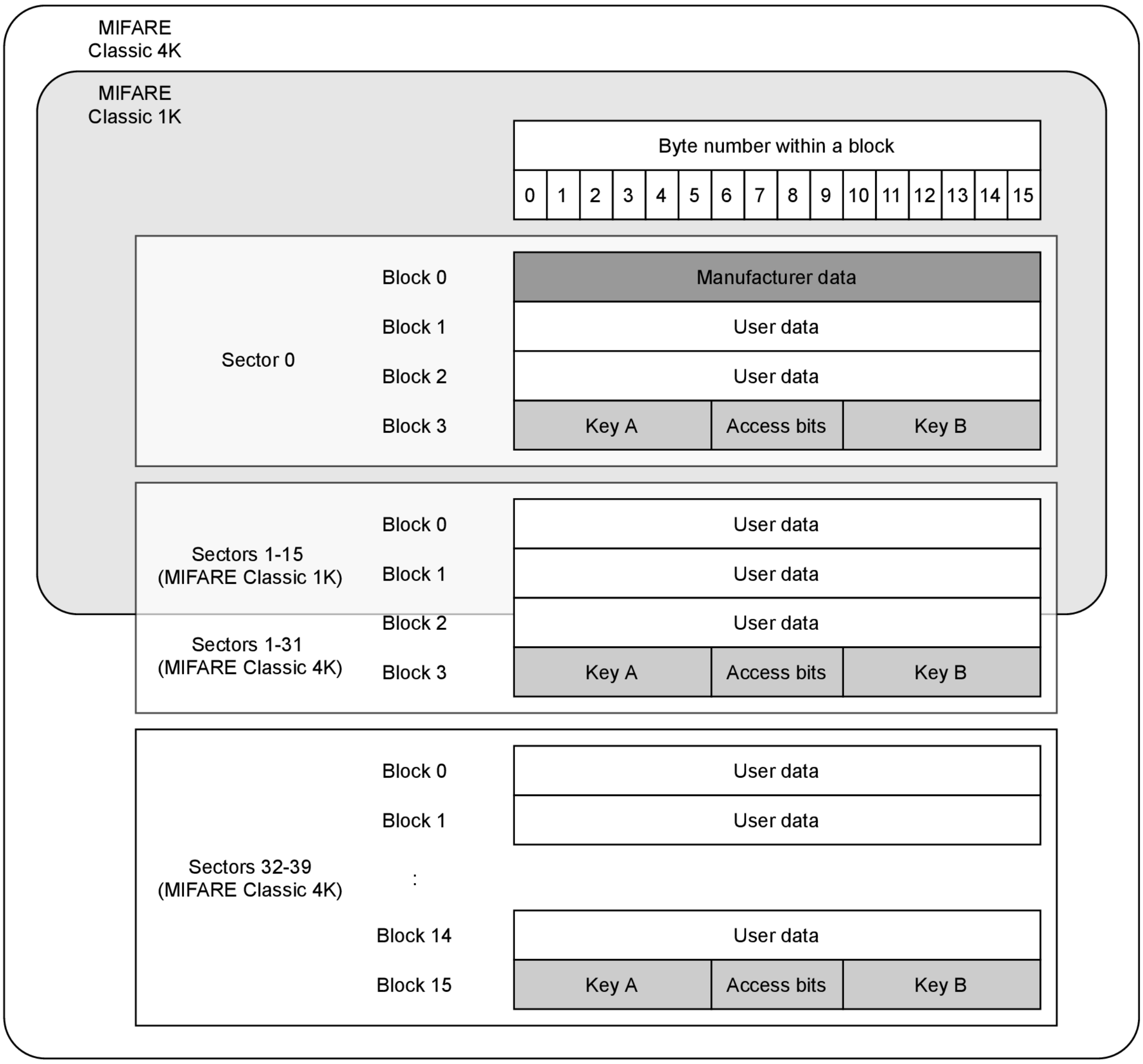


Figure 4.2: MIFARE Classic memory organization [7] [3]

Access bits			Access condition for						Remark
			Key A		Access bits		Key B		
C1	C2	C3	Read	Write	Read	Write	Read	Write	
0	0	0	never	key A	key A	never	key A	key A	Key B may be read
0	1	0	never	never	key A	never	key A	never	Key B may be read
1	0	0	never	key B	key A|B	never	never	key B	
1	1	0	never	never	key A|B	never	never	never	
0	0	1	never	key A	key A	key A	key A	key A	Key B may be read
0	1	1	never	key B	key A|B	key B	never	key B	
1	0	1	never	never	key A|B	key B	never	never	
1	1	1	never	never	key A|B	never	never	never	

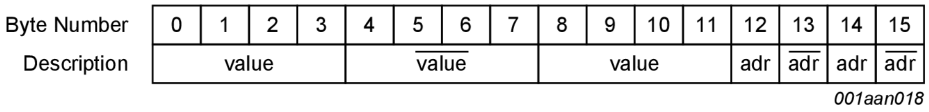
Table 4.2: Access conditions for the sector trailer [7]

In MIFARE Classic 1K variant, the memory is split into 16 sectors. Each sector contains four 16-byte blocks, for total of 1024 bytes of storage, including the keys and access conditions. The first data block of the first sector is used to hold manufacturer data, and it can always be read and never written. That yields 47 blocks usable for user data, with a total size of 752 bytes.

In MIFARE Classic 4K variant, the same memory organization as in the 1K version is used for the first 32 sectors. Starting from sector 32, every sector has 15 data blocks and sector trailer. To reflect this, the access condition bits have different semantics. The first set of bits `C1-C3` controls access to blocks 0-4, second set controls blocks 5-9 and the third set controls blocks 10-14. There are 8 sectors with different organisation, so in total, the card has 215 usable data blocks, resulting in total of 3440 bytes of memory available to the user.

Value blocks

Value block is a specially formatted data block, designed to allow performing electronic purse functions. Value blocks have a fixed data format that allows for error detection, correction and backup management. To format a regular data block, a write command is used. The value block holds a 32-bit signed integer value, that is stored twice in the block, two times in the non-inverted form and once in the inverted form. For backup management, one byte storing the address of the block is also stored, two times in each form (see Figure 4.3).



Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Description	value				$\overline{\text{value}}$				value				adr	$\overline{\text{adr}}$	adr	$\overline{\text{adr}}$

001aan018

Figure 4.3: Format of the value blocks in MIFARE Classic. Taken over from [7]

.

4.1.2 Crypto1 cipher

Crypto1 is an encryption algorithm created specifically for MIFARE Classic cards. The cipher is proprietary and its design was kept secret.

It stayed secret for 14 years, despite more than a billion chips shipped. Then, in 2008, a paper called *Reverse-Engineering a Cryptographic RFID Tag* [19] was published. The authors reverse-engineered the cipher from its silicon implementation. Although it was previously believed that such an approach would be too expensive to do, they have shown that with enough effort, it is doable using pretty common equipment.

First, they used acetone to dissolve the plastic card and isolate the silicon chips. Then they removed each successive layer of the chip through mechanical polishing. The chip contains six layers, the lowest of which holds the transistors. They took pictures of the layers using a standard optical microscope with 500× magnification.

After doing some processing of the captured images, they have found out the images contain about 70 different types of gates. They have built a library of these gates and implemented a template matching algorithm to find the occurrences of every gate of the same kind in the chip. But they still had to do a lot of manual work to identify the connections between the gates.

As their goal was to reveal the cipher and not necessarily the whole card logic, they focused on searching for a 48-bit register with XOR gates around. They found the components in one of the corners of the chip along with a circuit that appeared to be a random number generator, as it had an output but no inputs.

The map of logic gates and the connection between them provided almost enough information to reveal the cipher. But it was still necessary to find out the exact timing and inputs to the cipher. To do so, they analyzed the communication between a MIFARE tag and a reader.

The cipher consists of a 48-bit linear feedback shift register. Twenty state bits of the register are passed through a non-linear filter function f (consisting of f_a, f_b and f_c) to generate one bit of keystream. After that, the LFSR shifts to the left and uses the generating polynomial of $x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^{9} + x^{7} + x^{6} + x^{5} + 1$ to generate a new rightmost bit of the LFSR (function L).

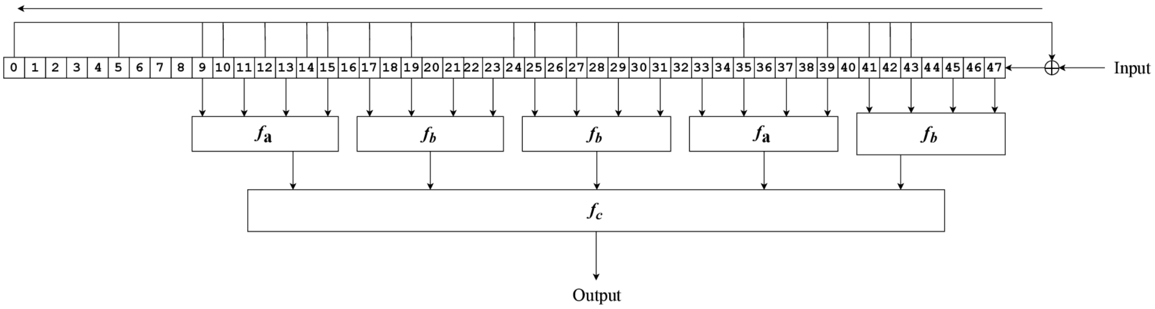


Figure 4.4: Crypto1 (N)LFSR schema

$$\begin{aligned} L(x_0x_1 \dots x_{47}) := {} & x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \\ & \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43} \end{aligned} \tag{4.1}$$

$$\begin{aligned} f(x_0x_1 \dots x_{47}) := {} & f_c(f_a(x_9, x_{11}, x_{13}, x_{15}), f_b(x_{17}, x_{19}, x_{21}, x_{23}), f_b(x_{25}, x_{27}, x_{29}, x_{31}), \\ & f_a(x_{33}, x_{35}, x_{37}, x_{39}), f_b(x_{41}, x_{43}, x_{45}, x_{47})) \end{aligned} \tag{4.2}$$

$$f_a(y_0, y_1, y_2, y_3) := ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3)) \tag{4.3}$$

$$f_b(y_0, y_1, y_2, y_3) := ((y_0 \wedge y_1) \vee y_2) \oplus ((y_0 \oplus y_1) \wedge (y_2 \vee y_3)) \tag{4.4}$$

$$f_c(y_0, y_1, y_2, y_3, y_4) := (y_0 \vee ((y_1 \vee y_4) \wedge (y_3 \oplus y_4))) \oplus ((y_0 \oplus (y_1 \wedge y_3)) \wedge ((y_2 \oplus y_3) \vee (y_1 \wedge y_4))) \tag{4.5}$$

4.1.3 Pseudorandom number generator

To generate a random reader and tag nonces, a pseudorandom number generator (PRNG) is used. Although the nonces are 32-bit long, they are generated using a 16-bit LFSR with generating polynomial $x^{16} + x^{14} + x^{13} + x^{11} + 1$ [17]. The register is clocked at 106 kHz (the bit rate of the protocol). Each clock tick, the LFSR shifts left, and the feedback bit is computed using L_{16}. The register is reset to the initial state every time the tag is powered on. That makes the value in the register depend only on the time passed since the tag was powered on. The *suc* function is used to calculate the next 32-bit sequence, and it will be used later in the authentication protocol. It will be referred to as suc_n, where n says how many iterations of the function is used, always using the output of the previous iteration as input for the next one.

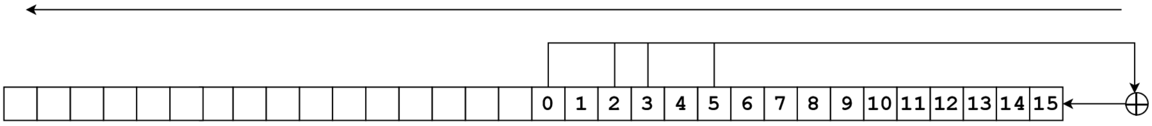


Figure 4.5: Pseudorandom number generator LFSR schema

$$L_{16}(x_0 x_1 \dots x_{15}) := x_0 \oplus x_2 \oplus x_3 \oplus x_5 \tag{4.6}$$

$$suc(x_0 x_1 \dots x_{31}) := x_1 x_2 \dots x_{31} L_{16}(x_{16} x_{17} \dots x_{31}) \tag{4.7}$$

4.1.4 Communication protocol

In order to access the card memory, the card has to be initiated and the reader has to authenticate to the card.

The initialization consists of the anticollision procedure. MIFARE Classic is following the procedure defined in Part 3 of the ISO 14443-A standard, see chapter 2.2.1.

Authentication

After the card has been selected, the reader has to authenticate itself to the card. Command `0x60` is used to authenticate with key A, command `0x61` for key B. As an argument, the block number is given, although the authentication is valid for the whole sector. The authentication command is followed by CRC, the other messages of the authentication protocol are not. The schema of the authentication protocol is in Figure 4.6.

When the authentication command is issued, the internal state of the stream cipher is initialized with the key that is used for the authentication.

The card answers with a random generated 32-bit tag nonce, nT. This nonce is XORed with the UID u of the tag, that was received during the anticollision phase. The result $u \oplus nT$ is fed into the result of the feedback function of the cipher, making it work as an NLFSR. From now on, the following communication is encrypted.

In the next step, the reader answers the tag nonce with its own random generated nonce nR, together with the answer to the tag nonce aR. The nR is also XORed into the feedback function of the cipher, but since the communication is already encrypted, it has to be decrypted first. As it is a stream cipher, the encryption of later bits of nR is influenced by the earlier bits that got XORed into the LFSR feedback. The answer aR is generated using the PRNG LFSR, being defined as $aR = suc_{64}(nT)$.

To finish the authentication, the tag responds with $aT = suc_{96}(nT)$. After that, the reader is authenticated to the card and can issue other commands.

In case the reader wants to authenticate for a different sector now, the procedure is called nested authentication and the protocol is a bit different. In this case, the authentication command has to be sent encrypted (using the cipher state from the previous sector). At

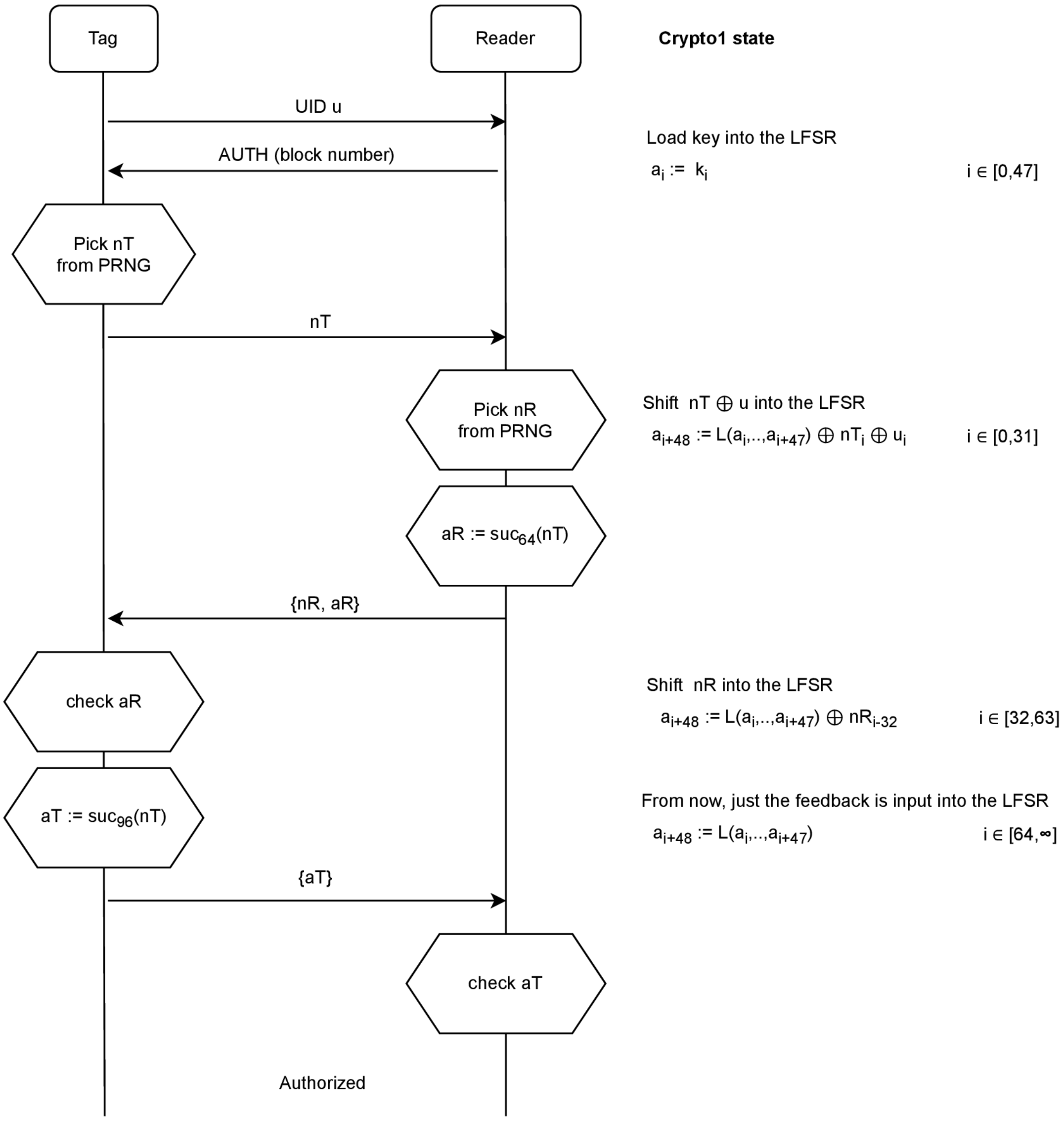


Figure 4.6: Authentication protocol. Symbols denoted by { – } are sent encrypted.

this moment, the cipher is initiated with the key for the new sector and the tag nonce nT is sent already encrypted. The rest of the protocol stays the same.

Read command

The read command requires a block address, and returns one MIFARE Classic block (16 bytes). In case the access bits deny the read using the key that was authorized, `NAK` (Not Acknowledged) is returned by the tag. The communication diagram is shown in Figure 4.7.

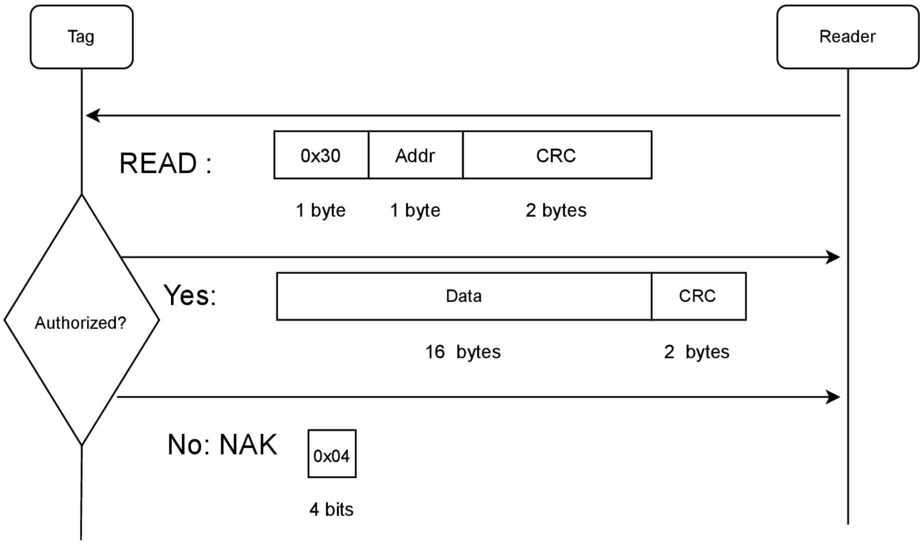


Figure 4.7: MIFARE Classic Read command. In case the CRC or parity is incorrect, a `NAK` with the value of `0x5` is transmitted.

Write command

The writing is split into two parts. First, the reader sends the write command with a block address. After the tag acknowledges, the 16 bytes of a MIFARE Classic block to be written are sent. They are acknowledged by the tag as well. The communication diagram is shown in Figure 4.8.

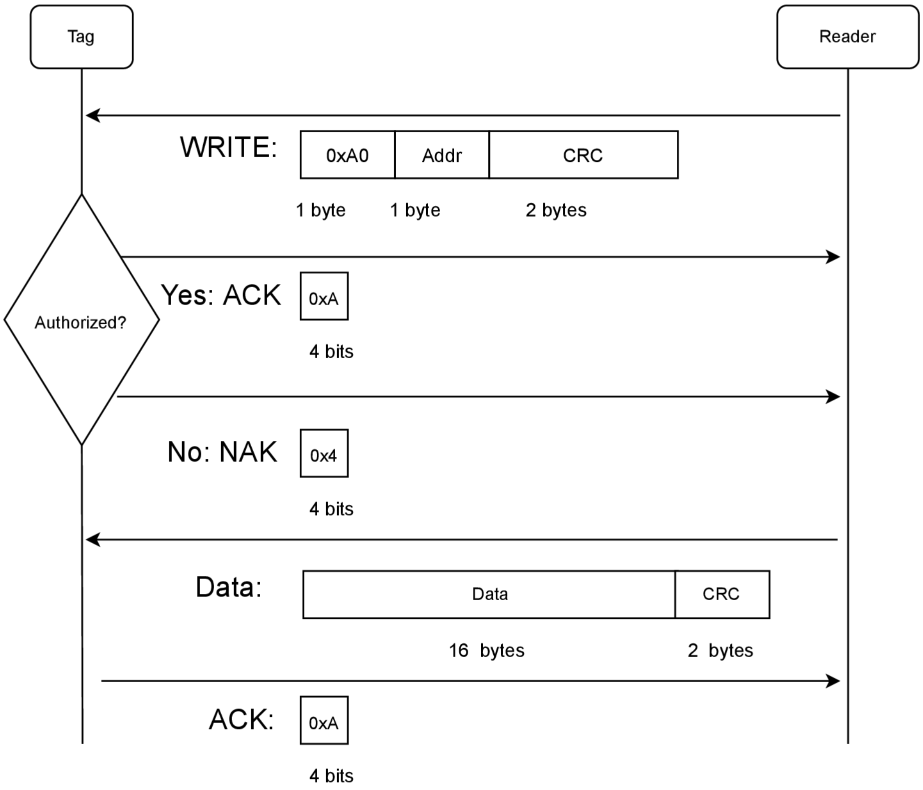


Figure 4.8: MIFARE Classic Write command. In case the CRC or parity is incorrect, a `NAK` with value of `0x5` is transmitted

Other commands

The Increment, Decrement, Restore and Transfer commands can be performed on a value block. The Restore command reads the value from the value block into the internal Transfer Buffer. The value from the Transfer Buffer can be written back into the value block using the Transfer command.

The Increment and Decrement are two-phase commands. First, the command is issued by the reader with the block address as an argument. The tag reads the value from the block into the Transfer Buffer and acknowledges the command. Then, the 4-byte operand is sent to the tag. The tag adds or subtracts the operand from the value in the Transfer Buffer and the operand is acknowledged to the reader.

Chapter 5

nRF52832 microcontroller

nRF52 is a series of microcontrollers manufactured by Nordic Semiconductor. Its main feature is the integrated Bluetooth Low Energy (BLE) radio. It is widely used in areas where low-power operation is required, such as wireless sensors, battery-powered computer peripherals and similar applications.

To increase the security of the Bluetooth pairing process, some microcontrollers from the nRF52 range contain an NFC-A tag peripheral. Its main goal is to serve as a channel for Out-Of-Band (OOB) pairing with devices that support the feature.

nRF52832 [4] is a mid-range chip from the series. It offers one 64 MHz ARM Cortex-M4F core with up to 256 KiB of flash memory and up to 64 KiB of RAM. It also features the NFC-A tag peripheral.

5.1 NFC-A tag peripheral

A unique feature of this microcontroller is the NFC-A tag peripheral. Its main features are:

- 106 kbps bit rate
- can wake up the microcontroller on RF field detection
- frame assembler and disassembler
- programmable timing controller
- hardware automatic collision resolution
- hardware CRC and parity functions

The peripheral contains a 13.56 MHz AM receiver and load modulator, compatible with the NFC-A specification. The integrated frame disassembler can automatically process the received frame, so only the data part is transferred to the RAM. During transmission, the frame assembler can automatically assemble the frame from the data in RAM. Parity and CRC of the frame is automatically checked and removed by the frame disassembler, and calculated and appended by the frame assembler. These features can be disabled.

The hardware anticollision procedure is able to automatically perform all steps until the tag enters the `ACTIVE` state. The UID, `ATAQ` and `SAK` responses are configured by writing into the corresponding registers.

The frame timing controller can be used to maintain the exact time interval between receiving a frame and transmitting a reply. When configured, it holds the frame that is to be transmitted until the configured interval passes.

The detection of the RF field by the peripheral can be used to wake up the microcontroller, even from the deepest power-saving mode called System OFF. In this mode, with the NFC wake-up enabled, the microcontroller consumes only 0.7 µA.

The peripheral is internally controlled by a state machine. The state transfers occur when specific events happen and they can trigger interrupts. Some of them can also be triggered manually by the user. The state machine is described in Figure 5.1.

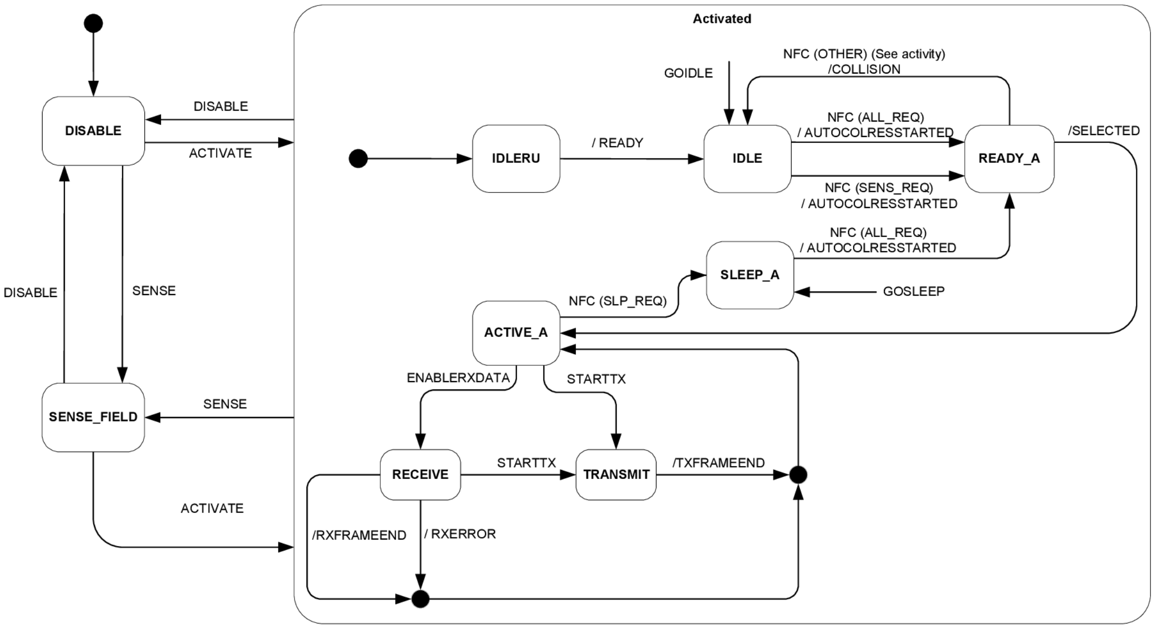


Figure 5.1: State diagram of the NFC-A tag peripheral. The transitions prefixed with / can trigger interrupts. Taken over from [4]

.

5.2 SDK

The nRF5 SDK[10] supplied by Nordic Semiconductor offers a broad selection of drivers, libraries and code examples. Most of it is distributed in the form of source code.

To enable the NFC peripheral, users can choose to emulate either NFC Type 2 or Type 4 tag. However, in both cases, the actual tag implementation is only supplied as a precompiled library. The libraries offer high-level APIs, e.g. for setting the tag memory content, but all the tag logic is hidden in the library.

The only source code related to the NFC peripheral are the wrapper functions. These functions allow the use of the peripheral without accessing the hardware registers directly. They also offer workarounds for some hardware bugs.

The wrapper functions are called internally from the tag emulation libraries, but there are no code examples for using them separately.

5.3 Developer kits

Many developer kits that allow prototyping with the nRF52 series microcontrollers without the need of designing a PCB are available. Two of them, both using the nRF52832 microcontroller, will be described.

5.3.1 nRF52 DK

The nRF52 DK [14] is a versatile single board development kit for the nRF52 microcontroller series. There are multiple variants available with different microcontrollers from the series. For this project, the most interesting one is the one with nRF52832 microcontroller, as it is the smallest one with the NFC tag peripheral. This variant of the kit is also marked as PCA10040.

For debugging, SEGGER J-Link debugger is integrated on the board. It can also be used to debug external chips through a pin header. The kit comes with an external NFC antenna that can be connected to the board.

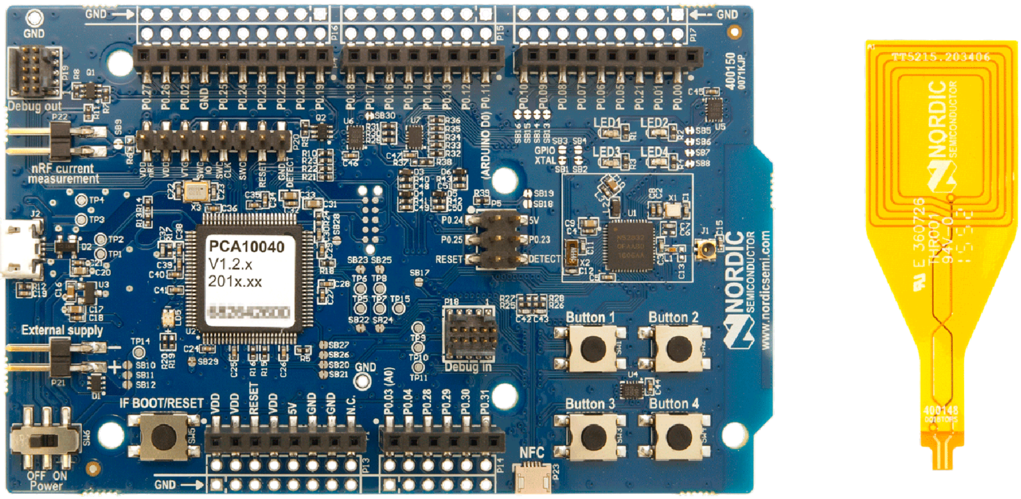


Figure 5.2: nRF52 DK kit contents

5.3.2 Thingy:52

The Nordic Thingy:52 [9] is a compact prototyping platform. Instead of a plain PCB, it is a small plastic box with one push button and an RGB LED. It includes a rechargeable Li-Po battery that can be charged via microUSB connector. NFC antenna is integrated inside the box. There is no debugger included in the kit, but after the removal of the black plastic cover, a pin header to connect with the debugger integrated in nRF52 DK is revealed.

To explore the capabilities of the platform, a smartphone application called *Nordic Thingy* is distributed. It allows to connect to the device over Bluetooth and see the measurements from the sensors integrated into the kit. The integrated sensors include:

- gas sensor

- color sensor
- pressure, altitude and temperature sensor
- motion tracking device
- low power accelerometer
- digital microphone

Figure 5.3: Thingy:52 kit contents

Chapter 6

Implementation

To implement the MIFARE Classic tag emulation, it was decided to use the nRF52832 microcontroller, as it is the smallest one from the nRF52 series featuring the NFC peripheral and also the most common one in the development kits. For development, the nRF52 DK development kit was chosen, as it contains an integrated debug probe that makes the development easier. The software should be easily portable to other boards as well.

In the following sections, the modules of the implementation will be described in the order they evolved. The block diagram of the modules is shown in Figure 6.1.

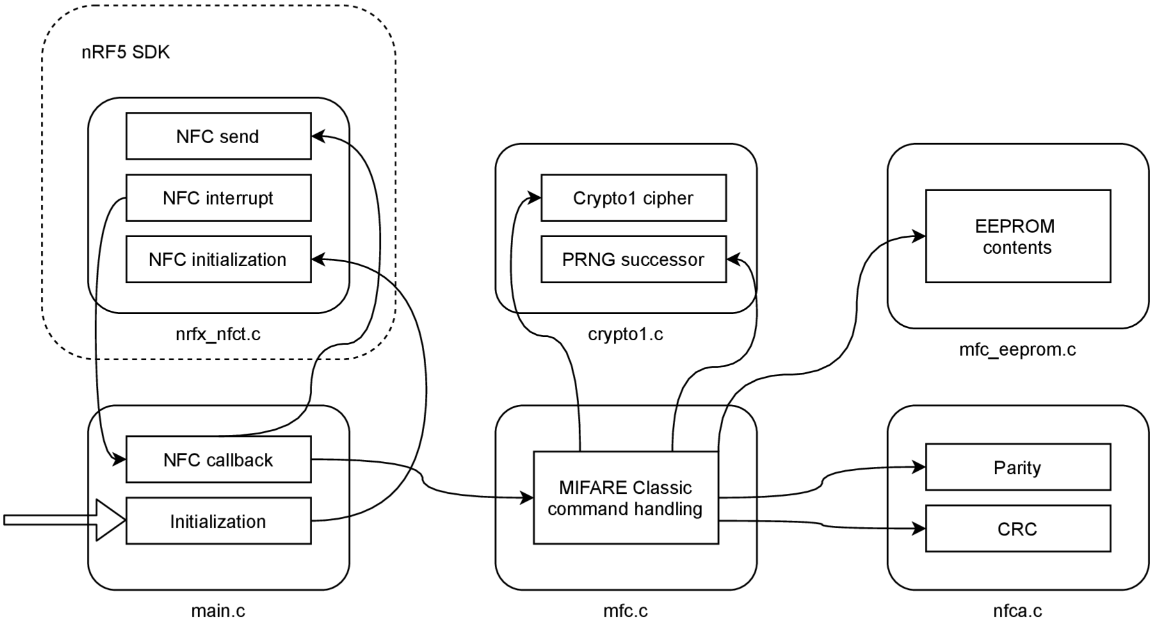


Figure 6.1: Block diagram of the implementation modules. The arrows represent the order the functions are called.

6.1 Main module and NFC peripheral handling

The main module (`main.c`) is the entry point of the code and is responsible for initializing the NFC peripheral and receiving the callback from the NFC driver. It is the only module that makes calls specific to the hardware platform used.

6.1.1 NFC peripheral initialization

The first goal of the implementation was to initialize the NFC tag peripheral, so that the hardware anticollision procedure can send artificial UID, `ATQA` and `SAK` responses to the reader. That would allow the emulated tag to be detected as MIFARE Classic and with the changeable UID, it would already allow to spoof some door access systems, that only use the UID they get from the anticollision phase to authenticate the tag.

While there is a driver for the NFC peripheral in the SDK, it is only used by the proprietary protocol implementations, that are only available as precompiled libraries. There is no code example on using the peripheral alone, nor any documentation on this topic. The only documentation available are the Doxygen comments of the driver code that gave some hints and in conjunction with the reference of the peripheral registers, it was possible to initialize the peripheral.

With the peripheral initialized, it was possible to configure the UID of the emulated tag using the functions of the driver. The tag was then properly detected by a reader. But there are no functions in the driver for setting the `ATQA` and `SAK` responses, so they had to be set by writing directly into the registers of the peripheral. The values kept changing to default after the first anticollision. It was revealed that the driver is resetting the NFC peripheral every time the field from the reader is lost, as a workaround for some hardware bug. So it is necessary to set the register value every time the field lost callback from the driver is called.

6.1.2 NFC send and receive

To figure out how to send and receive data using the NFC peripheral, it was decided to start with a simple MIFARE Ultralight implementation, as the protocol is similar to MIFARE Classic, but there is no authorization or encryption involved. It was found out that it is necessary to enable the reception and set the receive buffer every time the tag is selected, or a frame transmission is finished. A simple `READ` command that returns a constant value was implemented, and it was verified that a reader reads the data properly.

6.2 NFC-A: software CRC and parity calculation

The NFC-A module (`nfca.c`) implements the calculation of the CRC for ISO 14443-A frames and also the packing and unpacking of the parity bits from the frames and their calculation and verification.

As written in chapter 5.1, the NFC peripheral contains a hardware frame assembler and disassembler that can, among other, verify the parity bits and the CRC of the received frames and remove them from the frame on-the-fly as it gets copied to the memory. This was useful for implementing the MIFARE Ultralight tag, as every valid frame of the protocol (except of `ACK`/`NACK`, whose are shorter than one byte and thus do not follow the Standard frame specification) has a CRC appended.

But for MIFARE Classic, the hardware CRC cannot be used, as the nonces sent during the authentication process have no CRC appended. That means the CRC checking and removal in software had to be implemented.

The other characteristic of MIFARE Classic is that when the communication is encrypted, the parity of the frames is not calculated from the ciphertext, but from the plaintext. That makes it impossible to decode or encode the parity of the frames in the hardware, as only the ciphertext is passed to the peripheral. So the parity function had to be also implemented in software.

However, when those operations were moved into the software, the communication started to hang up with the first commands received. It was found out that when the CRC and the parity were removed in hardware, receiving a HALT command put the NFC peripheral into the SLEEP_A state, as stated in the specification. But without the hardware removal of parity bits and CRC, this does not happen and it is not even mentioned in the specification. The issue was solved by forcing the SLEEP_A state manually when receiving a HALT command.

6.3 Crypto1 and PRNG

The Crypto1 module (crypto1.c) implements the cipher and the successor function of the pseudorandom number generator.

During the implementation, the formulas from chapter 4.1.2 were used. As the microcontroller used is fast enough, code readability was the main focus. The 48-bit LFSR of the cipher is simply stored in an uint64_t integer and no manual optimizations of the output or feedback functions were made.

As debugging of the cipher together with the rest of the tag logic would be tricky, the cipher was developed alone and it was verified on traces of the authentication procedure, where the nonces and keys were known. It is worth noting that the nonces are transmitted in little endian byte order, although they are commonly formatted as hexadecimal numbers in big endian byte order by the tools and in the literature.

6.4 MIFARE Classic implementation

The MIFARE Classic module mfc.c implements the actual tag logic, including the authentication procedure and access conditions enforcement.

Most of the logic is hidden in a function that is called from the RX frame end callback in the main module. It is given the receive/transmit buffer pointer and a state structure of the MIFARE Classic. This structure contains the current state of the tag, the cipher and PRNG LFSR. The number of bits to be sent from the same buffer is returned to the main module. The function is implementing the state machine shown in Figure 6.2.

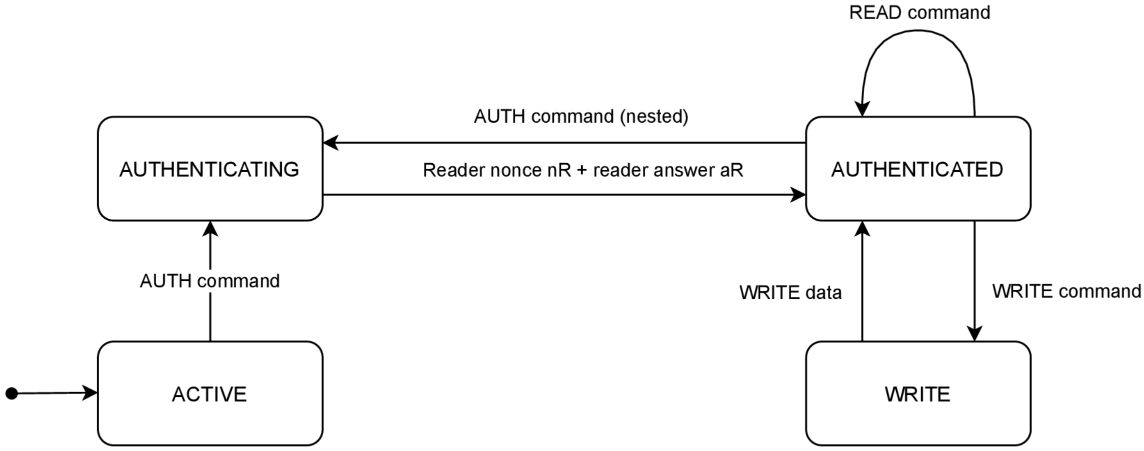


Figure 6.2: State diagram of the MIFARE Classic implementation.

It is able to handle the authentication, read and write commands, and the `HALT` command, that is sent encrypted when the reader is authorized to the tag. The nested authentication procedure is also supported. There is no support for the bigger sectors that are present in MIFARE Classic 4K, so only the 1K version can be emulated.

The access conditions enforcement is done by returning a `NAK` (Not Acknowledge) for the data blocks during read or write. When the sector trailer block is read, the bytes that are not readable are replaced with zeros. During the write into the sector trailer, only the bytes that are allowed to be written are changed in the memory, with the others keeping the original content. This write is always acknowledged, even when no bytes of the block are changed.

Chapter 7

Testing

For testing purposes, the memory of the tag was filled with contents that allow testing as many features as possible. Every data block was filled with ascending sequence of bytes. The value of the first byte of the block was the block number and every following byte of the block had the value increased by one. That would allow to see clearly if the the data got corrupted or a wrong block was read. The pattern can be seen in Figure 7.2.

To test the access condition mechanism, each sector had the key A set to the default value of `0xFFFFFFFFFFFF` and key B to the value of `0xAAAAAAAAAAAA`. Then, various access condition bits were set to the sectors. For compatibility reasons (will be described later), the same access condition bits settings was always used for two succeeding sectors. The settings are described in table 7.1.

Sectors	Block	Access conditions when using the key	
		Key A	Key B
2,3	0	read/write	-
	1	read	-
	2	-	-
	3	write Key A, r/w ACs, r/w Key B	-
4,5	0	-	read/write
	1	-	read
	2	-	-
	3	read ACs	write Key A, r/w ACs, write Key B
6,7	0	read/write	read/write
	1	read	read
	2	-	-
	3	read ACs	write Key A, r/w ACs, write Key B
Others	0	read/write	-
	1	read/write	-
	2	read/write	-
	3	write Key A, r/w ACs, r/w Key B	-

Table 7.1: Access conditions configuration of the tag emulated during testing

To test the implementation, three different readers were used. A Chameleon Mini, an Android phone with two different applications and an USB reader.

7.1 Chameleon Mini

While Chameleon Mini can be used to emulate MIFARE Classic by itself, it does not have support for MIFARE Classic in the reader mode. But it is still able to do the anticollision procedure and retrieve the type of the card and its UID. It also allows to send arbitrary commands to the card and that could be used to work with the MIFARE Classic card, but it would require implementing the Crypto1 cipher and the authentication procedure in software. To run the anticollision procedure and retrieve the transmitted data, there are two commands. `IDENTIFY` that outputs `ATQA`, `SAK` and UID of the card. The other one is `GETUID` that prints only the `UID` of the card. Below is the terminal output of these two commands:

```
IDENTIFY
101:OK WITH TEXT
MIFARE Classic 1k
ATQA:   0400
UID:    6A1346E6
SAK:    08
GETUID
101:OK WITH TEXT
6A1346E6
```

7.2 Android

On Android, two different applications were used. The first was TagInfo[1]. That is a simple proprietary application developed by NXP that is able to show basic info about a tag that was scanned. It can also show the memory contents of a MIFARE Classic tag, if the tag is using the factory default key, or if they key was added to the application by the user. It does not have any tag writing abilities.

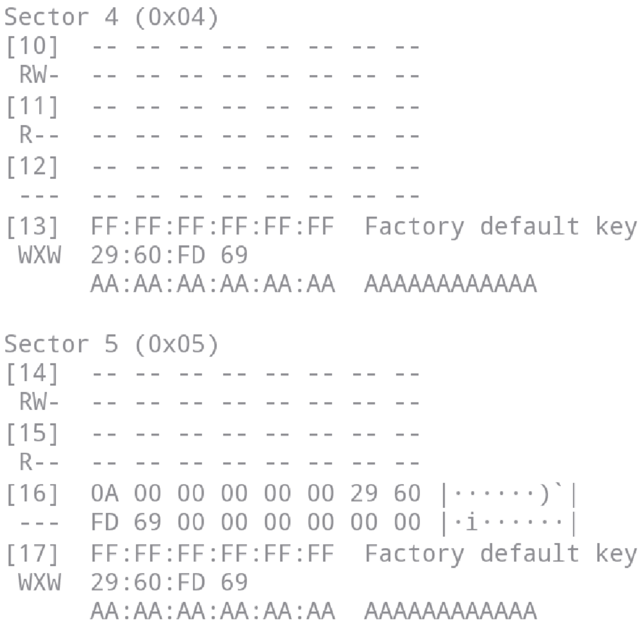
The application reads out the contents of the tag, using the default key A. However, there is no option to force it to use key B. So even though the key used as key B was added to the user keys store of the applications, it does not use it when the authentication with key A was successful, even though no data blocks are readable using key A (sectors 4-5 of the testing tag).

However, TagInfo has an issue in dealing with read-protected blocks. In block 3 of sector 5, it reads out invalid data (see Figure 7.1). This kind of issue started to happen during the development, after the access control enforcement was implemented. But it does not happen on block 3 of sector 4, which has the same access conditions. The issue is not present when using a genuine MIFARE Classic tag nor when using any other application or reader with the emulated tag.

The second was an open-source application called MIFARE Classic Tool [2]. It is a more feature-wise application designed specifically to operate with MIFARE Classic tags. It allows defining key files, which are text files containing one key per line. Then, it allows performing a read operation on the tag, where it can try all the keys on each block, to get the most possible data from the tag. The application also allows to write data to a tag, or even clone the tag. It can even clone block 0 containing the manufacturer data such as

[1] https://play.google.com/store/apps/details?id=com.nxp.taginfolite

[2] https://play.google.com/store/apps/details?id=de.syss.MifareClassicTool



```
Sector 4 (0x04)
[10]  -- -- -- -- -- -- -- --
 RW-  -- -- -- -- -- -- -- --
[11]  -- -- -- -- -- -- -- --
 R--  -- -- -- -- -- -- -- --
[12]  -- -- -- -- -- -- -- --
 ---  -- -- -- -- -- -- -- --
[13]  FF:FF:FF:FF:FF:FF  Factory default key
 WXW  29:60:FD 69
      AA:AA:AA:AA:AA:AA  AAAAAAAAAAAA

Sector 5 (0x05)
[14]  -- -- -- -- -- -- -- --
 RW-  -- -- -- -- -- -- -- --
[15]  -- -- -- -- -- -- -- --
 R--  -- -- -- -- -- -- -- --
[16]  0A 00 00 00 00 00 29 60 |······)`|
 ---  FD 69 00 00 00 00 00 00 |·i······|
[17]  FF:FF:FF:FF:FF:FF  Factory default key
 WXW  29:60:FD 69
      AA:AA:AA:AA:AA:AA  AAAAAAAAAAAA
```

Figure 7.1: Cut from a screenshot of NXP TagInfo application after a „Full scan“ of the emulated tag. Note that in block 0x16, invalid data is read, even though the block is not readable. In sector 0x12 with the same access conditions, this was not happening.

UID, in case it is written to a non-genuine tag that allows UID cloning. The application also offers many useful tools, such as a decoder/encoder for the access condition bits of the tag.

When the emulated tag is read by the application, it is never able to read the odd sectors from the tag (see Figure 7.2). By analyzing the logs from the microcontroller, it was found out that the issue is related to the nested authentication procedure. The application authenticates to the sector 0 using a regular (non-nested) authentication procedure. When it is done with reading the sector, it tries to authenticate to the sector 1 using the nested authentication procedure. But then, after the encrypted nonce nT is sent from the tag, no answer from the reader is received. Instead, the anticollision procedure on the tag is repeated and a regular authentication command is issued, but already on the sector 2. So the sector 1 gets skipped and the same is happening for every other odd sector.

During the analysis of the logs from the microcontroller, it was revealed that TagInfo is also trying and failing to do nested authentication, but unlike MIFARE Classic Tool, it tries again using the regular authentication procedure and succeeds. But the `nfc-mfclassic` tool used with the USB reader in the next section is using nested authentication for every sector following the first one and does not have a problem authenticating to the tag. So it seems that the issue is somehow related to Android, but no details were figured out.

7.2.1 Write tests

As the MIFARE Classic tool is the only application used during the testing that allows writing into single blocks of the tag, instead of just writing the whole tag memory, it was used to test the behavior of the write command. Several tests were performed, to test the access conditions enforcement as well.

To test writes into the data blocks, blocks that cover all combination of access conditions were chosen. During every subsequent test, the write command was issued with all 16 data bytes set to `0xCC`. The tag response was checked to be an acknowledge when the operation is permitted and a non-acknowledge when it is not. Then, the whole tag contents were read and it was verified that the block value is the one that is expected. This verification was skipped for the blocks that are not readable with key A nor key B. In the end, the microcontroller was restarted, to reset the memory of the emulated tag before issuing another test. The test results are in table 7.2.

Sector	Block	Key	Expected result	Result
2	0	A	Pass	Pass
2	1	A	Fail	Fail
2	2	A	Fail	Fail
2	0	B	Fail	Fail
4	0	B	Pass	Pass
4	1	B	Fail	Fail
4	2	B	Fail	Fail
4	0	A	Fail	Fail
6	0	A	Pass	Pass
6	1	A	Fail	Fail
6	2	A	Fail	Fail
6	0	B	Pass	Pass
6	1	B	Fail	Fail
6	2	B	Fail	Fail

Table 7.2: Test protocol of writes into the data blocks

To test the writes into the sector trailer, a sector trailer with the keys A and B swapped and the ACs set to default was used. This modified trailer was written to sectors 2 and 4 using both keys A and B. Then, the whole tag was read using the swapped keys and it was verified that the contents are as expected.

7.3 USB reader

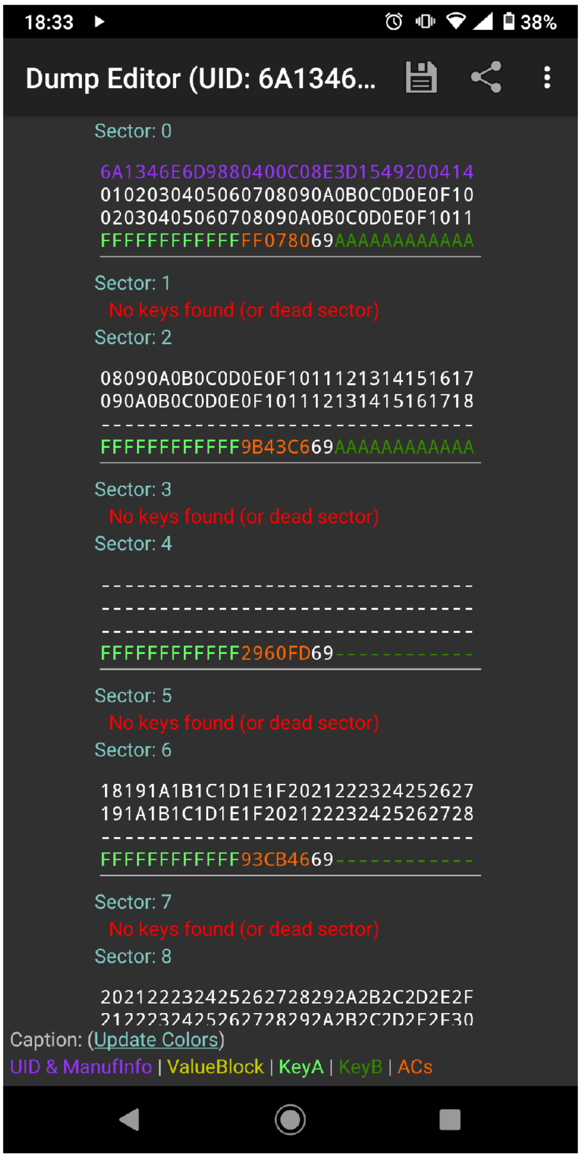
The USB reader used is ACR122U by Advanced Card Systems Ltd. To work with the tag, the `nfc-mfclassic` tool from the `libnfc` library [3] was used. The tool is only able to read and write the whole memory of the tag, it does not have options to work with individual sectors or blocks. To test the compatibility, the option to read the whole card memory into a dump file was used. It was done using the default key A.

The resulting dump named `test.mfd` contains the binary contents of the card and it was read properly. The tool has printed errors during the reading of the blocks that are not readable using key A and the blocks are replaced by zeros in the dump. Interesting behavior is that the key B contained in the sector trailers is always zeroed in the dump. Even when the key is readable and the sector trailer was read from the tag, as the access conditions are present in the dump. The key A is present in the dump. That means its appended by the tool, as it is never readable. In overall, the behavior is the same when

[3] http://www.nfc-tools.org/index.php/Libnfc

reading a genuine MIFARE Classic tag. The command line of the tool and the output is below:

```
    $ nfc-mfclassic r A u test.mfd
NFC reader: ACS / ACR122U PICC Interface opened
Found MIFARE Classic card:
ISO/IEC 14443A (106 kbps) target:
    ATQA (SENS_RES): 00  04
       UID (NFCID1): 6a  13  46  e6
      SAK (SEL_RES): 08
RATS support: no
Guessing size: seems to be a 1024-byte card
Reading out 64 blocks |.................................!
Error: unable to read block 0x1e
xxx.!
Error: unable to read block 0x1a
xxx.!
Error: unable to read block 0x16
xxx.!
Error: unable to read block 0x12
xxx.!
Error: unable to read block 0x0e
xxx.!
Error: unable to read block 0x0a
xxx........|
Done, 46 of 64 blocks read.
Writing data to file: test.mfd ...Done.
```



18:33 38%

Dump Editor (UID: 6A1346...

```
Sector: 0

 6A1346E6D9880400C08E3D1549200414
 0102030405060708090A0B0C0D0E0F10
 02030405060708090A0B0C0D0E0F1011
 FFFFFFFFFFFFFF078069AAAAAAAAAAAA

Sector: 1
  No keys found (or dead sector)
Sector: 2

 08090A0B0C0D0E0F1011121314151617
 090A0B0C0D0E0F101112131415161718
 --------------------------------
 FFFFFFFFFFFF9B43C669AAAAAAAAAAAA

Sector: 3
  No keys found (or dead sector)
Sector: 4

 --------------------------------
 --------------------------------
 --------------------------------
 FFFFFFFFFFFF2960FD69------------

Sector: 5
  No keys found (or dead sector)
Sector: 6

 18191A1B1C1D1E1F2021222324252627
 191A1B1C1D1E1F202122232425262728
 --------------------------------
 FFFFFFFFFFFF93CB4669------------

Sector: 7
  No keys found (or dead sector)
Sector: 8

 202122232425262728292A2B2C2D2E2F
 2122232425262728292A2B2C2D2E2F30
```

Caption: (Update Colors)

UID & ManufInfo | ValueBlock | KeyA | KeyB | ACs

Figure 7.2: Screenshot of MIFARE Classic Tool after reading the emulated tag. Note it is not able to read the odd sectors, although they all have the default keys used.

Chapter 8

Discussion

In this chapter, the implemented solution will be evaluated and compared to the existing alternatives. In the last section, its future extension and possible usages will be discussed.

8.1 Evaluation

The basic command set (authentication, read, write) of MIFARE Classic 1K was implemented. During the read and write operations, the access condition bit settings are evaluated and the access control is enforced. The implementation consists of multiple modules, where most of them are hardware independent, with only one of them being specific to the nRF52832 microcontroller used.

The solution was tested against an Android phone with two different applications and a USB reader connected to a laptop. Testing has shown that the solution is usable in practise, but there is an issue when Android application performs nested authentication on the tag. That happens when the reader is already authenticated to a sector and sends encrypted authentication command to authenticate for another sector. The consequences of the issue depend on how the application handles the authentication. If the application handles the situation correctly, it will reset the tag by toggling the RF field, perform the anticollision procedure again and authenticate using a non-encrypted authentication command. Then, the only visible impact is that the transmission will be slowed down. If the application does not handle the situation properly, some sectors may appear unreadable. The issue was not present when using the USB reader with a laptop. In that case, the nested authentications worked flawlessly.

8.2 Existing alternatives

During the research, two existing solutions for emulating a MIFARE Classic card were found, excluding those based on chips that do the emulation in hardware.

Chameleon Mini[18] is an open-source versatile tool for security analysis of contactless cards. It allows to emulate, clone, read and sniff various types of HF cards with focus on MIFARE. Both the hardware schematics and layout files and the firmware source codes are publicly available in a GitHub repository. It is built on top of Atmel ATXMega128A4U microcontroller and there is no special hardware to handle the NFC protocol - there is just a PCB antenna with a bunch of discrete components connected to the pins of the microcontroller. The modulation, demodulation and all protocol encoding and decoding is done in software. That is an advantage in the terms of hackability, but also a disadvantage in terms of code readability – as the microcontroller is quite slow in today terms, some of the code is highly optimized using inline assembly or precomputed tables. It is sold for about 100 euros.

Proxmark[11] is believed to be today's industry standard tool for RFID Analysis. It uses an FPGA for modulation and demodulation of the signal and supports both LF and HF cards. Protocol encoding and decoding is done in a microcontroller. It is the most capable solution, with support for many different tag types. It is also more expensive – is sells for about 300 euros.

The nRF52-based solution described in this thesis uses a hardware NFC peripheral. The peripheral does not allow reader emulation or sniffing, and it is limited to the NFC-A standard at bit rate of 106 kbps. It is also less hackable, as the lower layers of the protocol are fixed in the hardware and the configuration options are limited. The advantage is that the microcontroller is quite powerful and the precise timing is handled by the hardware, so there is enough power for the actual card logic that can be written in clean, straight-forward way with no optimization. It also features an embedded Bluetooth Low Energy radio that could be used to configure the emulated tag. The possibility to use the RF field detection of the NFC peripheral to wake up the microcontroller could allow the solution to work for months on a single coin-sized battery, if used seldom. Another advantage is the price - both nRF52 DK or Thingy:52 kits sell for about 50 euros.

8.3 Possible extension

The implementation could be extended in multiple ways. The first way is to broaden the range of tags that can be emulated. Any ISO 14443-A tag could be emulated with proper software support. That includes the whole range of MIFARE products. The only limitation of the peripheral in nRF52832 is the bit rate of 106 kbps. At least MIFARE Plus and DESFire support higher bitrates in their current chip revisions. To make the MIFARE Classic support complete, the support for value blocks and the related command set could be implemented, as well as support for the large sectors of MIFARE 4K.

The second way is to enable the Bluetooth Low Energy radio in the microcontroller and design a smartphone application that would be able to communicate with the tag emulation software. The application could alter the memory of the emulated tag, making it possible to switch the tag that is emulated either on user request, or automatically, depending e.g. on the location of the user.

The third way the project could be extended is to port the implementation into an existing device, or create its own PCB. The most interesting device would be a smartwatch or other wearable. It seems that the nRF52832 chip is popular in cheap Chinese smartwatches and fitness trackers. There is a project on GitHub by Curt White[1] that provides support for writing custom software for these smartwatches. However, it seems that none of them contain an NFC antenna. Another interesting device is the open-source rayBeacon[2] device, a coin-sized (25 mmm diameter) development board based on nRF52, featuring a connector for an NFC antenna.

[1] https://github.com/curtpw/nRF5x-device-reverse-engineering

[2] https://bitbucket.org/raytrails/raybeacon/src/master/

Chapter 9

Conclusion

The goal of this thesis was to study the MIFARE Classic smartcards and their relation and compliance with NFC and RFID standards. Then, propose a way to implement the emulation of a MIFARE Classic tag on the NFC-A tag peripheral of a microcontroller and implement the solution.

The solution was successfully implemented on the nRF52 DK development kit, based on the nRF52832 microcontroller from Nordic Semiconductor. The emulation of the card with 1 kilobyte of memory (MIFARE Classic 1K), supporting the basic command set (Authentication, Read, Write) was implemented. The implementation is split into multiple modules, where the hardware-specific code is in its own module, so that the rest of the implementation is portable.

The result was tested against multiple readers, using different applications. It has been proven that it is compatible with all of them and it was verified that the access condition rules of the data blocks are enforced. The capabilities of the implementation were compared to the existing alternatives. It is not as versatile as the more mature competition, but it has advantages in terms of price, form factor and power consumption. Thus, all the goals of the assignment were fulfilled.

The possible extensions were discussed and three ways of future development were proposed. The solution could be extended to support more ISO 14443-A compatible card types, such as the rest of the NXP MIFARE family. The Bluetooth Low Energy radio embedded in the microcontroller could be used to allow communication with a smartphone. This way, the tag emulation could be controlled by an application running on the smartphone. The solution could also be ported to a gadget based on a compatible microcontroller, so that the tag emulation feature could be embedded into a smartwatch or other wearable.

Bibliography

[1] *Information technology — Telecommunications and information exchange between systems — Near Field Communication Interface and Protocol -2 (NFCIP-2).* Standard ISO/IEC 21481:2012. Geneva, CH: International Organization for Standardization, 2012. Available at: https://www.iso.org/standard/56855.html.

[2] *Information technology — Telecommunications and information exchange between systems — Near Field Communication — Interface and Protocol (NFCIP-1).* Standard ISO/IEC 18092:2013. Geneva, CH: International Organization for Standardization, 2013. Available at: https://www.iso.org/standard/56692.html.

[3] *MIFARE Classic EV1 4K - Mainstream contactless smart card IC for fast and easy solution development.* NXP Semiconductors, 2017. Rev. 3.2. Available at: https://www.nxp.com/docs/en/data-sheet/MF1S70YYX_V1.pdf.

[4] *NRF52832 Product Specification.* Nordic Semiconductor, 2017. V1.4. Available at: https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf.

[5] *Cards and security devices for personal identification — Contactless proximity objects.* Standard ISO/IEC 14443-1:2018. Geneva, CH: International Organization for Standardization, 2018. Available at: https://www.iso.org/standard/73596.html.

[6] *Cards and security devices for personal identification — Contactless vicinity objects.* Standard ISO/IEC 15693-1:2018. Geneva, CH: International Organization for Standardization, 2018. Available at: https://www.iso.org/standard/70837.html.

[7] *MIFARE Classic EV1 1K - Mainstream contactless smart card IC for fast and easy solution development.* NXP Semiconductors, 2018. Rev. 3.2. Available at: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf.

[8] *MIFARE Ultralight EV1 - Contactless ticket IC.* NXP Semiconductors, 2019. Rev. 3.3. Available at: https://www.nxp.com/docs/en/data-sheet/MF0ULX1.pdf.

[9] *Nordic Thingy:52 User Guide.* Nordic Semiconductor, 2019. V1.2. Available at: https://infocenter.nordicsemi.com/pdf/Thingy52_UG_v1.2.pdf.

[10] *NRF5 SDK documentation.* Nordic Semiconductor, 2019. V16.0.0. Available at: https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/index.html.

[11] *Proxmark Wiki : Home.* 2019. Revision cee1fb9. Available at: https://github.com/Proxmark/proxmark3/wiki.

[12] *MIFARE DESFire EV3 contactless multi-application IC.* NXP Semiconductors, 2020. Rev. 3.0. Available at: https://www.nxp.com/docs/en/data-sheet/MF3DHx3_SDS.pdf.

[13] *MIFARE Plus EV2*. NXP Semiconductors, 2020. Rev. 3.0. Available at: https://www.nxp.com/docs/en/data-sheet/MF1P(H)x2_SDS.pdf.

[14] *NRF52 DK User Guide*. Nordic Semiconductor, 2020. V1.3.1. Available at: https://infocenter.nordicsemi.com/pdf/nRF52_DK_User_Guide_v1.3.1.pdf.

[15] *FeliCa Card User's Manual Excerpted Edition*. Sony Corporation, 2021. V2.11. Available at: https://www.sony.net/Products/felica/business/tech-support/data/card_usersmanual_2.11e.pdf.

[16] Finkenzeller, K. *RFID handbook : Fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. 3rd ed.th ed. Chichester: Wiley, 2010. ISBN 978-0-470-69506-7.

[17] Garcia, F., Rossum, P. van, Verdult, R. and Schreur, R. Wirelessly Pickpocketing a Mifare Classic Card. In:. May 2009, p. 3–15.

[18] Kasper, T., Von Maurich, I., Oswald, D. and Paar, C. Chameleon: A versatile emulator for contactless smartcards. In:. 2011, p. 189–206. ISBN 9783642242083.

[19] Nohl, K., Evans, D., Starbug, S. and Plötz, H. Reverse-Engineering a Cryptographic RFID Tag. In: *Proceedings of the 17th Conference on Security Symposium*. USA: USENIX Association, 2008, p. 185–193. SS'08.

[20] Paret, D. *RFID and contactless smart card applications*. Chichester: John Wiley and sons, 2005. ISBN 0-470-01195-5.