Security analysis of RFID tags

Roel Verdult

June 25, 2008

Abstract

Usage of Radio Frequency Identification is winning ground everywhere. Advantages of contactless communication compared to chips with contact are transaction speed, durability and ease to use. A major disadvantage is that messages can be intercepted from a distance by a malicious user. Eavesdropping of unsecured transmissions can be a serious security risc. This research describes a way to intercept this information. Furthermore, it shows the vulnerabilities in different major RFID systems and demonstrates how to exploit them.

Supervisors:
Flavio D. Garcia
Peter van Rossum

version: 1.00

Preface

The process during my master thesis was a experience I will never forget. The hard work of developing an embedded device almost let me decide to stop the project after the first months. I can still remember when my supervisor proposed the project, which was back then, still a theoretical idea of how we wanted to investigate contactless smartcards. At the start of my thesis there was within our university not much knowledge availble about hardware development that could supported me. But when I finally managed to communicate with the device I immidiatly started working on my case studies. It was nice to have a fellow student Gerhard de Koning Gans working next to me starting to do a similar project during his master thesis. We helped each other on various grounds during our development.

The fuss about OV-chipkaart suspended my graduation for some weeks, but it was very interesting to see how a topic like security suddenly gets so much media attention. The awareness that is invoked by our statements about the OV-chipkaart helps the people to understand their need for privacy and security.

Finally I want to thank Ravindra Kali, Vinesh Kali for there technical support during my research.

Contents

1 Introduction

Radio-frequency identification (RFID) is an automatic identification method, used for remotely storing and retrieving data. RFID can be used to transmit contactless small amounts of data over a distance. This identification technique is widely used to replace legacy systems like bar codes, entrance tickets and personal passes. There are low, high and ultra high frequency standards. A few ISO standards describe the details of these techniques. The most widely used technique is the High Frequency proximity identification described in the ISO 14443[1] standard[1]. This standard is used in most contact less smart cards.

RFID systems are used in different environments. Each application has its own security requirements. A simple product identification system clearly needs less security than an access control gateway. The focus of this research is on systems that need at least some kind of protection.

The ISO standards for RFID systems provide no security features like authentication, integrity, authorization or availability. Though it is possible to implement a secure communication layer on top of the default transmission layer.

The question that raised here is: if there is no standard for this security, does this mean that every company need to invent its own layer. What we see in practice is that one company designs a system which is adapted by many other companies to keep production costs low and to provide (some) compatibility with other parties.

To investigate the security of RFID systems it is required to know the features, limitations and processes used during the communication. These subjects will be explained in chapters 3,4 and 5. A reader that is already familiar with this information can skip these chapters.

To be able to investigate different systems on a very low-level, special hardware is required. This hardware was not available for an affordable price at the beginning of this research. The solution was to in-house develop the required hard- and software. The result a device called the Ghost a fully working RFID eavesdropper and tag emulator. Since this was a substantial part of this research, this thesis will describe and explain parts of the design decisions that were made during this process.

The field tests performed during this research are described in the case studies presented in the latter chapters of this document. One of these case studies led to a publication in a top Security conference Esorics[4]. The Ghost played a major role in reverse engineering a proprietary cryptographic algorithm of widely used RFID tags.

[1]Two methods of modulation are described in this standard, variant A and B. The research in this thesis focuses on the modulation of variant A. Reference to this part of the ISO is given by 14443A.

2 Research Question

What are the security features of different RFID systems which are using the most widely used ISO-14443A(1-3) standard?

Security features concerning RFID devices are very important. Because of the wireless interface it does not require direct contact it is vulnerable for unnoticed communication between a tag and a malicious reader. Nowadays there are billions of tags sold that are based on the ISO-14443A standard. More than 70 percent of them are based on the Mifare Ultralight or Classic technology. They are used in systems like public transport and access control systems. The impact is very high when the security features are very weak. For example, access cards and tickets which are linked to personal bank-accounts could be faked in such a way that they are not distinctuishable by a system in any way. This is the main reason why the focus of this research focuses on this particular type of RFID systems.

3 Hardware

The usage of ISO-14443A RFID devices requires two different hardware parts. The reader, in the ISO referred to as proximity coupling device (PCD). This reader is an embedded device and contains an antenna to communicate at the frequency of 13.56 Mhz. The reader creates a electronic field which is used as a power source by a transponder tag, in the ISO referred to as proximity integrated circuit card (PICC). An example of a reader and a tag are shown in *Figure 1*.

Figure 1: Reader and Tag

In addition to these two standard hardware parts I have developed the firmware and software for a third device, the Ghost. With this Ghost I was able to analyse and test several systems at a very low-level of communication.

3.1 Reader

The reader can be connected to a computer or used as a standalone device. When it is controlled by a computer it often only acts as an antenna which is used to communicate to a tag. The requests performed by the reader are often very simple and is working on a fast operation speed. Cheaper solutions often use no security layer at all at the transmission level.

The reader uses the electronic field to communicate to the tag by dropping the field for 2.28μs. The drops are at different time intervals which modulates the data transmission to the tag. The modulation is done according to the Modified Miller encoding technique. *Figure 2* shows a modulation example. The modulation of one bit takes 9.44μs, this is called a bitperiod. The bitperiods are seperated by vertical lines in the figure. To transmit a *zero* there are two options, which is determined by the last bitperiod. Drop the field at the start of the bitperiod when the last transmitted bit was also a *zero*. Do not drop the field when the last transmitted bit was a *one*. The modulation of a *one* is always the same, drop the field in the middle of the bitperiod.

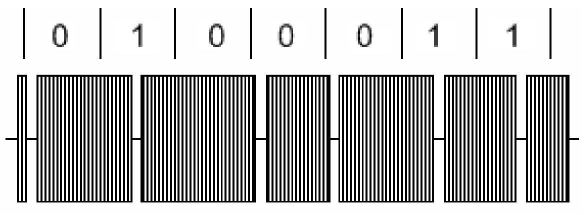


Figure 2: Example of modified miller encoding

Because the electronic field will only drop for a few micro-seconds a simple capacitor in the tag can overcome power interruption. The maximum range of this field is about four inches.

When the reader tries to find a tag nearby it will send continuously a welcome command for a tag that could be near. When the tag is in the field it will respond as specified in the ISO-14443A. From this moment a communication session is started. If there are multiple tags in the range of the reader, it will try to select each one in turn so it will be able to handle all the available tags.

3.2 Tag

There are two different kind of tags: passive and active ones. The passive tags are cheaper ones and do not have any power supply, so they completely rely on the electronic field of the reader. This limits the features of the tag. Active tags possess a small battery as a power supply. They often work with a longer range and do more complex computations. Most of the ISO-14443A tags are passive, since the reception range is small and their applications are often very simple. I mostly worked with tags that look like a credit card. They have the antenna embedded into the border of the card. The more expensive tags are often of the same material as a credit card, while the cheaper, disposable tags, are just simply maded out of paper. The tag will use a different way for communication. The tag can not drop down the electronic field like the reader, in stead, it will set up some resistancy in this field that can be detected by the reader. This resistancy is active in particular time frames which modulates the data. For this the Manchester encoding is selected and is woven into the electronic field that is created by the reader by using the subcarier frequency 847.5 Khz. This frequency is a divisor of the main 13.56 Mhz frequency.

Figure 3 shows the modulation of a few bits using the manchester encoding. The Manchester encoding is easier to understand than the Modified Miller encoding. The bitperiod is split up into two parts. When a *zero* is transmitted the first half of the carier wave stays intact while the second half is inteferred by a small field resistance. Modulation of a *one* is the complement of the modulation used during the transmission of a *zero*.

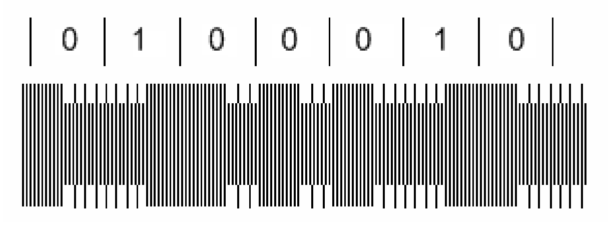


Figure 3: Example of manchester encoding

3.3 Ghost device

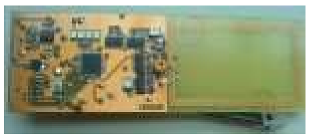


Figure 4: Ghost

The Ghost, showed in *Figure 4*, is actually a simple programmable RFID tag. It can communicate with the reader the same way a tag does. The major difference though is that the microchip on the device is programmable. I have written a firmware that provides you with features that control the complete communication bytes between the reader and the tag. This way it is possible to eavesdrop information or impersonate tags. It has a RS232 interface which can be connected to the serial port of the computer. This is useful for logging transferred frames and for updating the configuration of the ghost. The Ghost has its own 9V battery. Because of this it is possible to let it work standalone. When the messages are known beforehand by the user, so only a reproduction is needed, no computer interface is required. This way it is compact and easy to hide from human observers, which could be useful in some cases. Blueprints of the Ghost are available in the Appendix B of this document.

4 Related work

There are a few projects covering similar subjects. The master thesis Embedded Security Analysis of RFID Devices[5] written by Timo Kasper describes a project which is closely related to this research. Kasper focuses more on the development of the custom hardware which can do a different type of analysis. Technical details about the encoding and decoding techniques are very good described in this paper. He successfully tested the hardware he developed on the World Championship Soccer entrance tickets.

Kirschenbaum and Wool developed a low cost Extended-Range RFID Skimmer[6]. This article shows a very easy way to increase the eavesdropping range.

Ross and Goto developed a very primitive device [7] that communicates with low frequency RFID devices. They were able to trick a specific access control system and grant themself unauthorized access without the original tag. Heydt-Benjamin and his team were able to compromise the security of the first credit cards containing contactless features[8].

Gerhard Hancke developed a device which applied a succesful relay attack[9]. This is a man in the middle attack where the original tag is replaced by a tag-emulator. This emulator gathers the requests and is connected via a wireless connection to a mallicious reader. This reader communicates with the original tag, sends the requests and gathers the answers. For this he used one of the widely used Mifare Classic tags. The communication between the tag and the reader is encrypted, the encryption is not harmed because the hardware only records the communication waves and plays them through an emulator back to the reader.

5 ISO14443-A Protocol

The ISO 14443A[1] is the most widely used RFID standard in the world. This is the main reason why this research is focused on this ISO standard. Though results can be generalized to similar proximity 13.56 MHz systems that use the second modulation variant ISO 14443B[1] standard. In addition they can be roughly applied to similar systems operating at the 125 kHz frequency according to the ISO 15693[2] and the 13.56 Mhz Vicinity cards with a longer communication range as described in the ISO 11785[3].

The features, timing and messages of reader and tag are specified in the ISO 14443-3 standard. This chapter explains the features of the protocol very briefly. This is necessary to understand the communication between reader and tag.

5.1 Overview

This section introduces the basics of the protocol which are related to this research. The tag must implement some elementary actions like the anti-collision and halt command. The anti-collision provides a way to get the unique identifier (UID) of a tag even if there are more tags in the field. The halt command disables a tag. After disabling, the tag will not respond anymore unless it is waked up again.

The features from a tag can be detected in the anti-collision. The reader selects and process the tags that are compliant to its system. The scheme in *Figure 5* shows a some paths that could be taken during a tag-processing cycle. The horizontal layers annotate examples of systems. The example systems are explained more in detail during the case studies.

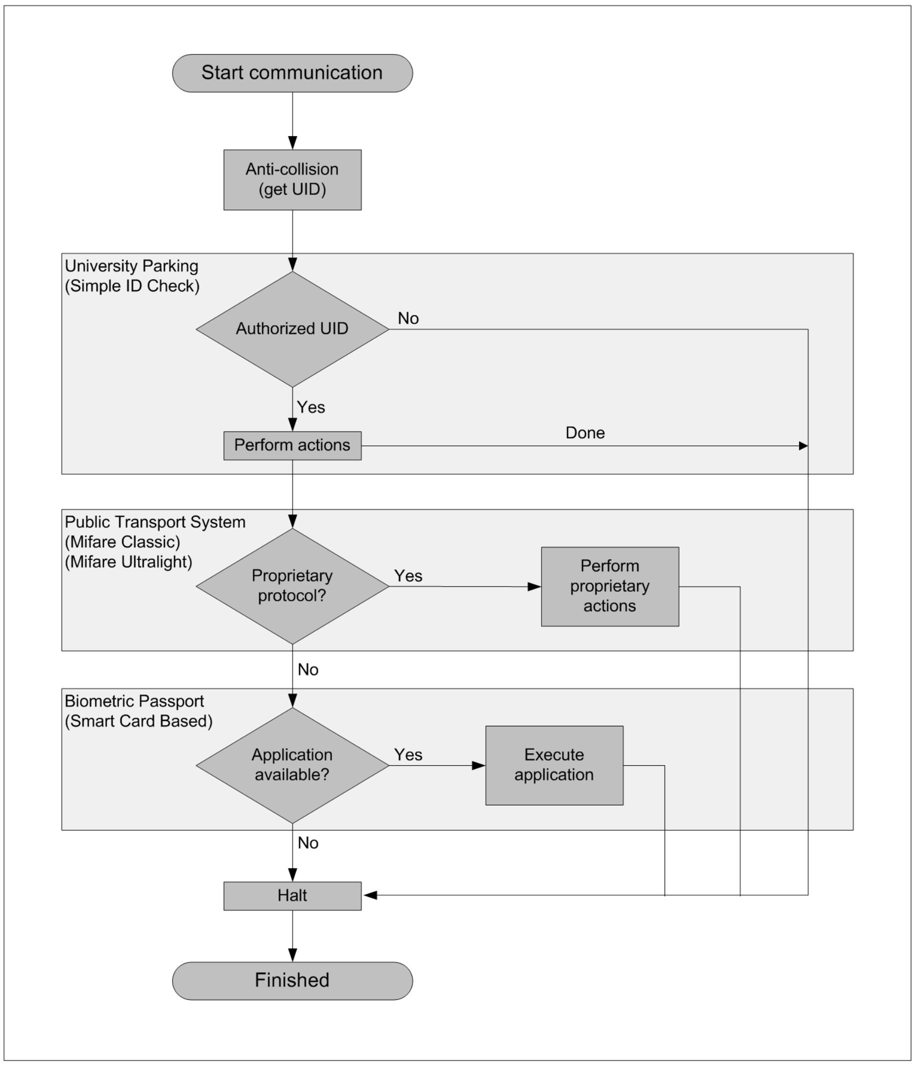


Figure 5: RFID example system processing a tag

5.2 Anti-collision

The anti-collision procedure is always performed as startup communication between a reader and the tag. The anti-collision is required to detect which nearby tags are available. Every tag has a different unique identifier (UID). To avoid any collisions in the communication, the ISO standard defines the anti-collision protocol. *Figure 6* presents the schematic overview of an anti-collision select sequence.

The first action comes from the reader (PCD). It probes for any tag (PICC) that is within reading distance. The probing can be done in two ways. It can send a REQA or a WUPA. For these messages a 7-bits commands are transferred. To distiguish from other communications the command is 1 bit shorter than any other command. The REQA (request) command requests all the tags to respond in order and let the reader know of their existence. The WUPA (wakeup) command wakes up tags that are in the field but disabled earlier, this means they are not active at the moment.

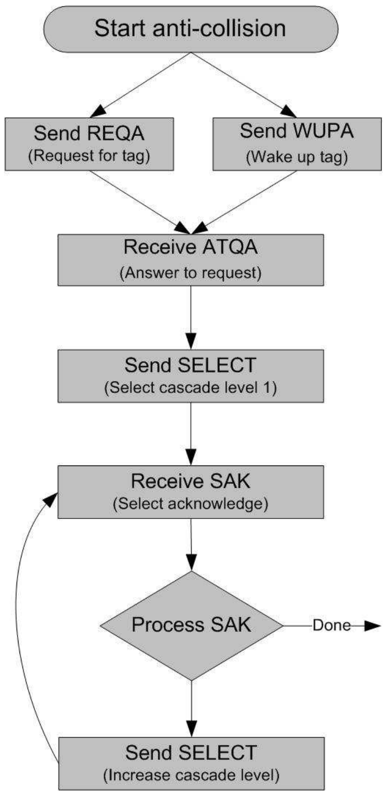


Figure 6:
Anti-collision sequence

When no tag responds, the REQA command is send over and over again. The delay between commands is proposed but in practice implemented differently by every manufacturer. Experiments show that some readers use one second while others use just a few milliseconds. As expected, the smaller the interval, the faster a tag can be moved through the field. This interval is not the only aspect that influence the speed, but it seems to be a rather large factor. Apart from this the speed also depends on the length, number and intelligence of the frames transferred between reader and tag.

A tag in the field will respond on a REQA or WUPA command with the ATQA block (answer to request). This will initiate the anti-collision procedure in the reader. The reader will try to find all the tags in the field (this could be multiple tags). Every tag contains a UID (unique identifier) which offers a distinction of between tags. The UID could exist of 4, 7 and 10 bytes. The ATQA will supply the bit-length of the UID. In the anti-collision process the reader uses a binary search to detect multiple tags. With the retrieved overview the reader can filter out the tags that are not compliant to the system.

After the reader has received the ATQA block, it will send a SELECT (select) command with the valid UID starting bit(s) responding to the current request of the binary search. When a UID of the tag matches on the prefix of these bits it will respond with its complete UID. If multiple tags are responding simultaneously on the SELECT command, a stricter prefix with more specified starting bits is send by the reader. As search area gets smaller ultimately an individual transponder can be identified. This process is visually displayed in *Figure7*.

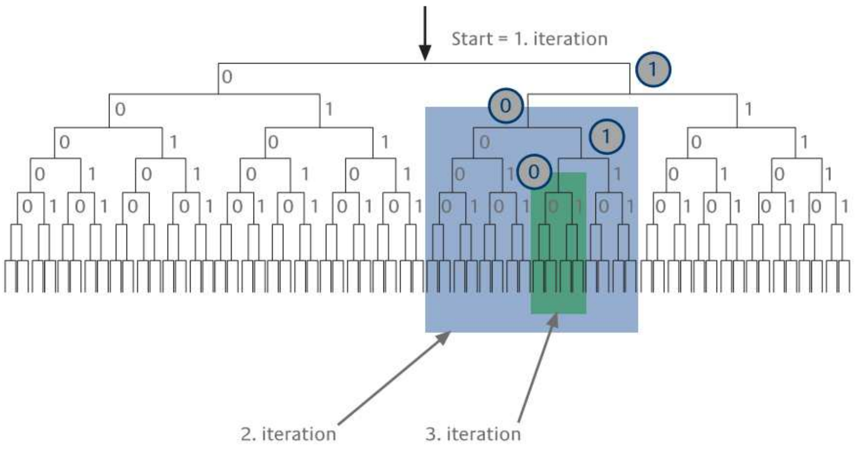


Figure 7: Binary search tree

If the mask singles out only one tag, the reader sends a new SELECT command with the specified UID of this tag. The tag responds with a SAK (select acknowledge) command. After the SAK command this cascade level is completed, but there could be multiple levels. For 7 UID bytes, the SEL command will be transferred two times and for 10 bytes it is transmitted three times. When the SAK describes no more UID bytes are available the anti-collision ends and the tag will turn to active state. In this state the tag processes all commands until a HALT (disable) command is received from the reader.

The anti-collision as defined in the ISO is always using non-encrypted data. This results in that it is vulnerable to several attacks like replay, relay and forging.

The next trace is observed from the communication between the reader and the tag during the anti-collision. For this example a tag is used with an UID length of 7 bytes. This means that two cascade levels are used during the anti-collision. The communication is observed from the reader side.

```
write len=1, data= 26                        => Welcome (REQA)
read: len=2 val= 44 00: OK                   => Respond (ATQA)
write len=2, data= 93 20                     => Select cascade 1 (SEL)
read: len=5 val= 88 04 f2 52 2c: OK          => CT, UID, BCC
write len=7, data= 93 70 88 04 f2 52 2c => Select available tag (SEL)
```

```
read: len=1 val= 04: OK                => Select Acknowledge (SAK)
write len=2, data= 95 20               => Select cascade 2 (SEL)
read: len=5 val= b1 ec 02 80 df: OK    => UID, BCC
write len=7, data= 95 70 b1 ec 02 80 df => Finish select (SEL)
read: len=1 val= 00: OK                => SAK without cascade bit set
Layer 2 success (ISO 14443-3 A)        => UID = 04 f2 52 b1 ec 02 80

CT  => Cascade tag byte (88), signals that the UID is not complete yet
BCC => Checkbyte, calculated as exclusive-or over 4 previous bytes
```

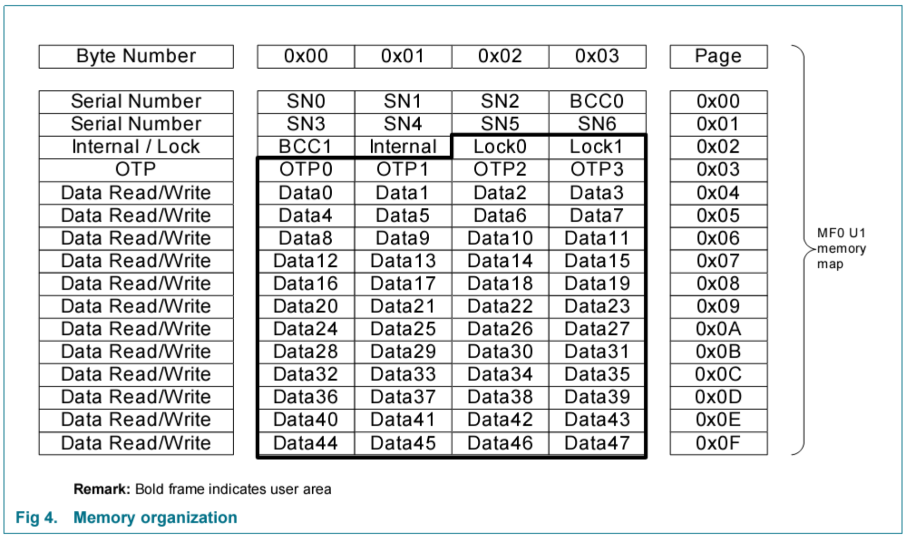
5.3 Mifare

In RFID tags often As proprietary protocols and commands are used. A good example of a proprietary protocol is the MIFARE[10] chip produced by the manufacturer NXP (formerly Philips). The protocol sequence and commands in this product were not publicly known, until Gerhard de Koning Gans recovered them using his practicle attack[12]. Because MIFARE is sold as solution to manufacturers of readers and tags, some information can be found in their documentation, though it is still far from a complete specification. This makes research harder since they can only be reviewed as a blackbox. The NXP products I used during my research are the Mifare Ultralight and the Mifare Classic.

5.3.1 Mifare Ultralight

The cheapest alternative chip embedded in the tag that is produced by NXP is the Ultralight variant[11]. It does not provide any encryption layer for the communication. It has a very small amount of memory (64KB). This is divided into 16 pages of 4 bytes each. The first two pages contain the UID and BCC bytes. *Figure 8* shows an overview of the memory available in a Ultralight tag.

Page 0x02 contains the lock-bits which can lock memory blocks. After locking a memory block, any request to change the memory will be refused. On a new tag only the first two blocks are locked, so that the UID can not be changed. A lock bit can only be set once, clearing a lock bit is not possible. A system could use this to lock an invalid ticket that contains arbitrary memory. Though what should be kept in mind is that there is also a bit that can lock the page that contain the lock-bits, after flagging this bit no changes could be made anymore to the lock-bits. This, in stead, could be useful for an attacker to avoid that his tag gets locked while using arbitrary memory content. Page 0x03 contains a One Time Programmable counter. The original value consists of only zeros. All the bits in this page can be flagged ones. There is no possible way to reset a flagged bit. This makes it a counter that can only take 32 different values ever. A system could use this counter to keep track of the trips made with the ticket. The memory of other pages could be reset, but this counter can only be increased, never reset or decreased.



Byte Number	0x00	0x01	0x02	0x03	Page
Serial Number	SN0	SN1	SN2	BCC0	0x00
Serial Number	SN3	SN4	SN5	SN6	0x01
Internal / Lock	BCC1	Internal	Lock0	Lock1	0x02
OTP	OTP0	OTP1	OTP2	OTP3	0x03
Data Read/Write	Data0	Data1	Data2	Data3	0x04
Data Read/Write	Data4	Data5	Data6	Data7	0x05
Data Read/Write	Data8	Data9	Data10	Data11	0x06
Data Read/Write	Data12	Data13	Data14	Data15	0x07
Data Read/Write	Data16	Data17	Data18	Data19	0x08
Data Read/Write	Data20	Data21	Data22	Data23	0x09
Data Read/Write	Data24	Data25	Data26	Data27	0x0A
Data Read/Write	Data28	Data29	Data30	Data31	0x0B
Data Read/Write	Data32	Data33	Data34	Data35	0x0C
Data Read/Write	Data36	Data37	Data38	Data39	0x0D
Data Read/Write	Data40	Data41	Data42	Data43	0x0E
Data Read/Write	Data44	Data45	Data46	Data47	0x0F

MF0 U1 memory map

Remark: Bold frame indicates user area

Fig 4. Memory organization

Figure 8: Memory of a Mifare Ultralight tag[11]

5.3.2 Mifare Classic

There is much more memory available in a Classic tag than in the Ultralight. The Mifare Classic comes in three different versions, with memory sizes of 1KB, 4KB and the Mini(320 Bytes). The versions only differ in size, the Mini and 1KB version only consist of sectors of 64 bytes, while the 4KB version has 16 extra sectors of 256 bytes. The lower sectors consist of 4 blocks of 16 bytes. Each of these blocks have 4 pages like the Ultralight. But in general the Classic tag only works with blocks and sectors. Every sector has three blocks free for storage except for the first sector which has one block reserved for the UID, BCC and manufacturer data. In *Figure 9* a schematic of the memory can be found. In comparison to the Ultralight, NXP claims that the Mifare Classic tag provides more security features. The communication between the tag and the reader is encrypted. Secret keys and random numbers are used to initialize the encryption. There are different memory sectors available which all can be seperately protected by two keys. The secret keys are shared keys that are known by the reader and the tag. Before any memory operation is performed both sides prove each other that they know the same key. This is done using a 3-way challenge and response authentication protocol.

Every sector is protected by a secret key A and is often makes use of a second secret key B. The access conditions defines the rights per key. It can for example be used to define a read and write key. Both keys and access conditions are stored in the last block of a sector, the sector trailer. A serious design flaw by using this trailer has been found by a fellow student of mine, Gerhard de Koning Gans. He is able to retrieve the plaintext of a sector without even

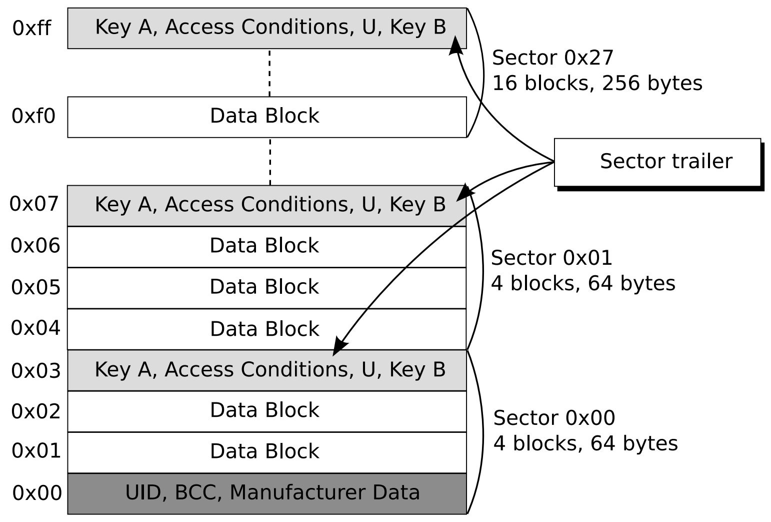


Figure 9: Memory of a Mifare Classic tag[10]

knowing the secret key of that sector[12]. This document describes in the case studies more weaknesses we have found in the protection of the Mifare Classic.

In the last period of writing my master thesis I was member of the Team that reversed engineered the algorithm used in the Mifare Classic. We constructed a very effective practical attack which allows an attacker to retrieve the cryptograpic key within seconds from only one trace of communcated data. This research resulted in an article Dismantling Mifare Classic[13]. This paper describes our findings during the last part of my master thesis research.

The Ghost played a very effective role in this research. We used it to act as a mallicious Mifare Classic tag. An original tag responds during the authentication with a random-looking nonce. With the ghost we were able to control this none and send the same one over and over. With this feature we were able to reveal some serious weaknesses present in the Mifare Classic algorithm and constructed two practical attacks to exploit them.

6 Software

Because the hardware of the Ghost was a brand new design, a new firmware was needed. This firmware is the core of the device and runs on a PIC micro-controller. In this chapter, I give a technical description of the design and usage of the firmware I have developed for the Ghost.

To configure the Ghost and process captured information, I have developed a special application which is called RfidSpy. This application is a GUI oriented application which runs on Windows. The features and design decisions are explained in the second part of this chapter.

To connect the Ghost and RfidSpy I have designed a protocol which describes the packets communicated through a RS232 connection. In the last part of this chapter I will briefly discuss the information transferred between both parties.

6.1 Ghost firmware

The core of the Ghost is the firmware running on the microchip. It handles all realtime events like capturing and sending bits. Developing and testing this firmware took about 4 months. This is because it involves a lot of precise timing issues.

6.1.1 Environment

The micro-controller that was used is a Microchip PIC18F4620. To program the firmware on the chip I used the programming device that is called ICD2. This device can be used by the application MPLAB IDE.

For developing and compiling the source-code I used the application Source-Boost. This is a third party program which supplies their own compiler for PIC micro-controllers. In the end I would have chosen another compiler to develop the firmware. This is because the Sourceboost compiler contains some serious flaws which produce unexpected behaviour of the controller. Luckily there was a fix availavle for every flaw, although it dramaticly slowed down the development.

6.1.2 Design

The compiler only supported C instead of C++, that is why I was not able to make an object oriented design for the firmware. The code is split in isolated modules to improve readability.

- **Core module**

  The brain of the device is implemented in the core module. Everything is controlled and instantiated from this module. The core is focused on receiving and sending of RFID frames. When such a sequence starts, it often requires a quick response and parsing of following requests. When a certain amount of time has passed without receiving any frame, the core will do a quick scan for any incoming RS232 commands.

- **Specifications**

  All the commands that are specified in the ISO standard are stored in this section. Further more, it contains constats that define the supported RFID frame-lengths and polling time.

- **Miller decoding**

  The decoding is very specific and standalone procedure. To keep this implementation as clean as possible it was isolated from the other modules. The receiving has some very specific timing constraints, therefore there was no place for any unused overhead instructions during the communication.

- **Manchester encoding**

  The encoding is quite similar to the decoding section. This module needs to be as optimized as possible. Parts of the encoding section are written in assembly. This was needed to be quick enough to be in the first time-frame which is required for the anti-collision procedure.

- **RS232 communication**

  In this module all the communication between the computer and the Ghost is handled. The pin-layouts and protocol specific matters are covered here. The Ghost cannot buffer any data that is presented at the port. To solve this, it tries to detect when information is available at the port and will wait for a resend of the whole packet.

- **Computer packets handling**

  The commands send by the computer are parsed in this module. When a reaction should be communicated back, it is prepared and composed. Calculation and verifications of the packet checksums are handled before a packet is transmitted or processed.

- **RFID frame handling**

  There are a lot of pre-specified frames which are used for example, in the anti-collision. In this module they are identified and checked for correctness. Most known frame types are recognized, if it can not be mapped to a known frame, it is annotated as unknown and logged by the ghost so the user can analyze later which frames should be manually emulated.

- **Micro-controller specific features**

  There are some very hardware specific settings, like pin connections, interrupt configurations and instructions to perform a hard reset. These features are handled in this module.

The Ghost needs to do two main jobs. At first it needs to process RFID frames that are captured with the antenna. The second one is to handle messages that are send by the computer on the RS232 port. Since the micro-controller does not support multi-threading, another robust solution was required. Capturing of RFID frames is very time-critical and can not be interrupted for some RS232 job. The solution I used was a very quick polling system which tries to detect if anything is available on the RS232, while the capturing is running most of the time and only stops after a relative long capture time-out.

6.1.3 Usage

When the power is connected the firmware will boot automatically. There is a reset button available which will produce the same effect as reconnecting the power. When the Ghost boots it will sends a welcome message and the default configuration through the RS232 connection. It does not matter if a computer is connected since it will not wait for a acknowledgement. This way a computer is not needed to be able to use the device.

The Ghost acts the same as a tag. This means it requires the same distance to the reader and responds within the same time-space. The anti-collision code in the firmware does not support the identifying algorithm for multiple tags, so in general it requires to communicate with the reader alone. The strength of the readers electric field is specified in the ISO, though after doing some tests it turns out that there is much variation in this. This could mean that the positioning distance of the Ghost should be altered a little for certain readers to get optimal results.

6.2 RfidSpy

The host application RfidSpy which runs on the computer is written from scratch. There was no existing application that supported all features I needed for my research. Because the software is published under an open-source GPL license I was able to use some existing GPL libraries.

6.2.1 Environment

Because the aim was to make a easy to use application, the usage of a program language with good visual supports seemed to be the best choice. I had a lot of experience with Borland Delphi, so to speed up the development I used this language for the user interface.

Besides the user interface, the connection to the Ghost and reader was needed. Communication to the Ghost was simply achieved by using a free 3rd party Delphi component QCCom32[2] that could communicate through the RS232 port of the computer. The reader required a driver and a separate library before it could be accessed by the application. I wrote this small library in C language since it is strongly depending on other libraries written in this language.

[2]http://www.bytearts.com/downloads.html

6.2.2 Design

RfidSpy does not contain any complex algorithms. In fact it is only designed to represent the information captured and produced by the Ghost device. This was simply achieved by designing a good userinterface. I will explain more about the GUI in the next chapter, the usage of RfidSpy. Apart from the user-interface an important design is the integration overview with the other components. In *Figure 10* you will find a schematic overview of the interactions between the components.

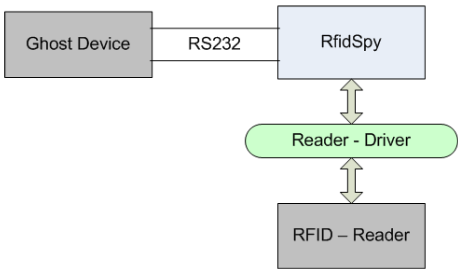


Figure 10: Component interactions

The reader that I used is the OpenPCD reader[3]. This reader supports complete control over the transmissions between reader and tag. The OpenPCD reader supplies an open hard- and software environment, which allows the user to build, compile and investigate all the parts of the reader autonomously.

6.2.3 Usage

To keep the user-interface as simple as possible all the features can be found categorized in one main form which is presented in *Figure 11*. To be able to use the Ghost, a connection should be made. This can be done using the connect button in the upper left corner. It will let you choose from the available COM (RS232) ports on the computer.

When a connection is made a operating mode can be chosen. The modes provide different ways of operation. For each mode a set of options is available.

The reader can be used seperate from the Ghost. When the start button is pressed a connection to the OpenPCD will be requested. After a successful connection, communication with a tag can be requested. The anti-collision is executed and the UID is retrieved from an available tag.

6.3 Protocol between RfidSpy and Ghost

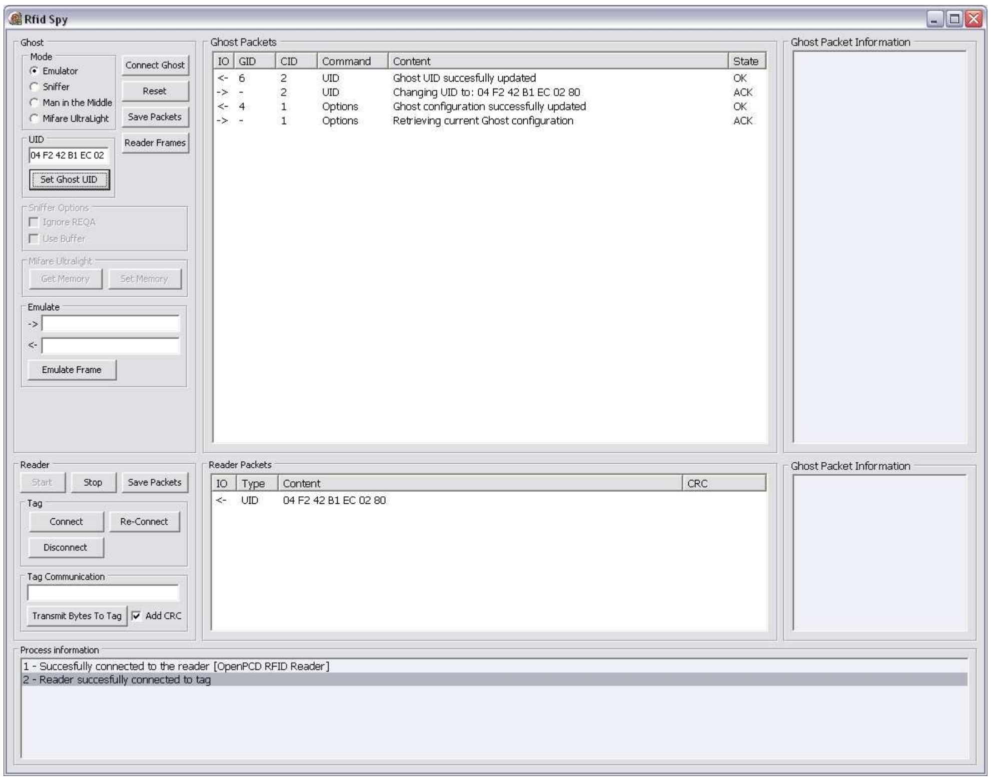
For the communication between RfidSpy and the Ghost a custom protocol is designed. Below I will show the formal representation of this protocol.

```
packet = { Header, PacketContent }

  Header = { StartByte, PacketContentCRC }

    StartByte        = 1 Byte  => 0xBA
    PacketContentCRC = 1 Byte  => 0x??
```

[3]http://www.openpcd.org



Mode: [✓ Emulator] Sniffer Man in the Middle Mifare UltraLight

UID: 04 F2 42 B1 EC 02

Ignore REQA [ ]
Use Buffer [ ]

Add CRC [✓]

Ghost Packets

IO	GID	CID	Command	Content	State
<-	6	2	UID	Ghost UID succesfully updated	OK
->	-	2	UID	Changing UID to: 04 F2 42 B1 EC 02 80	ACK
<-	4	1	Options	Ghost configuration successfully updated	OK
->	-	1	Options	Retrieving current Ghost configuration	ACK

Reader Packets

IO	Type	Content	CRC
<-	UID	04 F2 42 B1 EC 02 80	

Process information

```
1 - Succesfully connected to the reader [OpenPCD RFID Reader]
2 - Reader succesfully connected to tag
```

Figure 11: RfidSpy screenshot

```
PacketContent = { GhostId, ComputerId, Command, CommandInfo }

  Command = { Reset | Transfer | Emulate | Ok | Error | Info |
              Acknowledge | ProgramUid | ProgramATS | GetOptions |
              SetOptions | GetMifareUl | SetMifareUl | GetReaderFrames}
  CommandInfo = { TransferFrame | EmulateFrame | GhostOptions |
                  Message | ProgramUidFrame | MifareULMemory }

    TransferFrame = { Length, Bytes }
    EmulateFrame = { LengthIn, BytesIn, LengthOut, BytesOut }
    GhostOptions = { GhostMode, bIgnoreREQA, bUseBuffer }
    Message = { String }
    ProgramUIDFrame = { uiUIDLength, UidBytes }
    MifareULMemory = { pbtMifareULBytes, pbtCRCBytes, pbtOriginalBytes }
```

Since the Ghost device technically can not support buffering of incoming RS232 messages, it needed to be compensated by the RfidSpy application. Every time a command is send to the ghost it will wait for a confirmation. During this time it will keep re-sending the command until the Ghost replies with an acknowledgement. This undesirable way of communicating is only needed when sending to the Ghost. The RfidSpy does supports buffering of the RS232 port, so the Ghost does not have to wait for any confirmation.

Every packet consists of header, command and commandinfo. There are a few general commands used to configure the ghost. ProgramUid and ProgramATS to set the anti-collision information and (Get/Set)Options to configure options like ignoring similar sequential frames.

Reset, Ok and Acknowledge are used to control the state of the Ghost. The commands Error and Info are implemented to support extensive logging feature. Detected problems can easily reported back to the computer. This is very useful during the development of a embedded device.

7 Attacks

To support different research methods I have chosen to let the Ghost work in four different modes. Each mode has its own advantages for certain scenarios. This chapter describes all four different modes and their support for investigating security features.

7.1 Sniffing

The first configuration the Ghost supports is the sniffing method. This mode can be used to eavesdrop frames sent by a reader. The Ghost will not respond to any message, this way it will not interfere with transactions that are communicated between the reader and a tag. Since the Ghost can not receive the manchester signal from an different tag it is only capable to understand the information send by the reader. At first this looks like a very important limitation. But if the eavesdropped frames captured from the reader are resend by our own reader, the tag will give us the answers we missed earlier. This of course is only possible if there was not some kind of session set up between the original reader and tag. If there was a cryptographic challenge during the communication, a replay attack of the frames could not be performed on the tag. For that reason a man in the middle attack is more appropiated, which will be discussed in chapter 7.3. The field of the reader is not very large, so a position near the reader must be found where both the tag and the Ghost are in the field. Experiments show that the Ghost must be between the reader and the tag to give the most reliable trace.

The communication between a 3rd party reader and tag can be very quick. A request and response could be send within 100 milliseconds. This is to fast to eavesdrop and send through a RS232 connection at the same time. To overcome this problem I have implemented a buffer which can store about 40 frames. After the transaction has taken place, the buffer can be requested from RfidSpy and it will transfer all the frames at once.

To detect a tag the reader keeps sending a welcome (REQA) message through its field at a certain time interval. This interval is depends on the implementation of the vendor. For embedded standalone readers this interval is rather small, like only several milliseconds. In order to prevent to gathering a complete buffer with only REQA frames, there is a filter which will leave out frames that are similar to the last frame. Some information is lost here, though this could simply be overcome by just counting the repeats of the last frame and store this in the packet that is transferred to the computer. This feature was not necessary for my research, so I left it out of my developing scope.

7.2 Emulation

In emulator mode the ghost is able to clone a simple tag. The user can supply an UID which is used in the anti-collision. In addition some incoming and outgoing bytes can be defined, so the Ghost knows how to react on certain frames. This could be useful when a reader only wants to identify the tag and request one simple non-encrypted answer. After the Ghost is configured, it can be disconnected from the computer and be used as a standalone device. This mode is very useful for a replay attack. When the communication between the original reader and tag is known, it can be cloned by the Ghost.

7.3 Man in the middle

This is the most advanced mode of the Ghost. It needs the original tag and reader, the OpenPCD reader and the Ghost connected to a computer. The Ghost communicates with the original reader and transmits the requests to the computer. The computer processes the request through the OpenPCD to the original tag, which answers back to the computer. The computer transmits this answer back to the Ghost, which sends it to the original reader. A schematic overview can be found in *Picture12*.

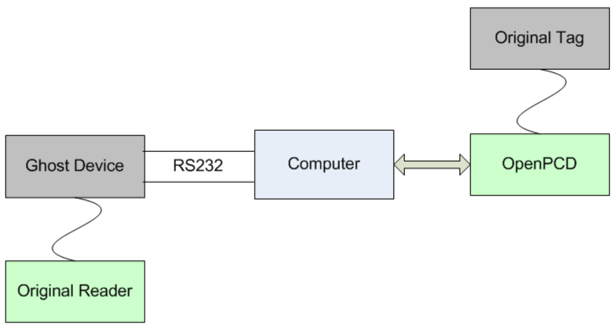


Figure 12: Man in the Middle mode

From the communication both original sides could not detect any of this man in the middle that is set up between them. They only big problem that occurs is the timing issue. There is a RFID communication, computer processing, RS232 transmission and a Ghost processing more than in the original environment. This differs a lot the original timing constraints. It is vendor specific to define any constraints. So it needs to be investigated if certain implementations can detect a attack like this.

7.4 Mifare Ultralight

This is a very specific mode which I needed for a particular case study. It can completely simulate a Mifare Ultralight tag. An Ultralight tag consists of 64 memory bytes, which are partly writable. The Ghost will simulate the Mifare Ultralight tag including its memory read and write methods. It is possible to view the memory of the Ghost at any time with the help of the RfidSpy application. When read or write actions are performed, a trace of commands is being stored within the Ghost buffer. The RfidSpy can read this trace quite similar the way it is done in sniffer mode. Since the Mifare Ultralight does not support any encrypted transaction, it was easy to simulate. The memory that is written though could be encrypted by the used application. This is purely depending on the particular implementation design of the application.

8 Case studies

This chapter describes several case studies performed in the field. For all researches the Ghost was used to research the communication between reader and tag. For each case study a low level security analysis is performed. The vulnerable aspects will be described including their risk of happening. In addition to this, suggestions and counter-measures are presented to prevent attacks on the weaknesses of these systems.

8.1 University parking

The parking system of the Radboud University is the first case study I have performed. Any employee of the University is able to park if they subscribed for this service. This costs the employee some small payment each month. Before the employee enters the parking lot he has to pass a barrier. This barrier will open when a valid employee-card is positioned in front of the integrated RFID reader. The employee-card is actually a Mifare Classic tag and is also used to grand access to the building. The aim was to capture and fake the transaction performed between the gateway and the card.

First I have analyzed the communication between the card and the gateway. I used for this the sniffer mode of the Ghost. In this mode I was able to see all the requests that were send by the gateway when a valid tag was in front. This complete trace is shown below.

```
write len=1, data= 26                    => Welcome (REQA)
read: len=2 val= 04 00: OK               => Respond (ATQA)
write len=2, data= 93 20                 => Select cascade 1 (SEL)
read: len=5 val= 44 45 fa d7 2c: OK      => UID, BCC
write len=7, data= 93 70 44 45 fa d7 2c => Select available tag (SEL)
read: len=1 val= 08: OK                  => Select Acknowledge (SAK)
write len=2, data= f7 49                 => * Unknown Command *
read: len=1 val= 04: OK                  => * Unknown Answer *
write len=2, data= 50 00                 => Halt, deactivate tag
```

It was interesting to see that a trace from a valid tag did not differ from a trace made with an unknown tag. This means the gateway terminal is actually very stupid. It will first perform the complete anti-collision protocol, perform a command and dispatch the tag again. After this it will check its UID against a list of valid entrance codes. When this validation succeeds the gate will open so the employee can enter the parking lot.

The entrance and exit gateway work exactly the same way. Important though is that the system keeps track of the position of the user. When a employee has entered the parking lot, it first needs to check out before entering will work again. This is probably used to avoid simple fraud. No employee will be able to pass their tag to someone else and both park together in the parking lot.

The tags have a Unique Identifier, otherwise the whole system would be useless. Since all manufacturers are involved in this matter, it will be very unlikely a second tag can be found with a similar UID. Manufacturers offer their customers the possibility to buy a collection of tags which all have UIDs within a small predefined range. This makes the hardware perform faster during the verifying methods. In addition to this administration of given out UIDs will be much easier for the manufacturer. While this at first looks harmless, further examination shows that it is actually extremely vulnerable to a domain-replay attack.

If we clone a tag with the ghost and use one UID, it is possible to enter the gateway without trouble. Though the original owner of this tag will not be able to enter the parking lot anymore while we are still in there. But because we know that the UIDs of the tags given by our university are within a special range, we can let the Ghost generate random UIDs that are within this range. Not every employee has a parking subscription, but since a transaction is completed in less than 50ms, we can try 20 different UIDs per second. To optimize this, we can filter out the valid ones and add them to our own entrance list.

We can conclude from these findings that the parking system has almost no security at all. It only depends on the uniqueness of the tags. If a manufacturer suddenly starts producing cheap RFID tags with custom UID numbers on demand, the security of systems like this, simply depending on the UID will fail completely.

8.2 Public transport system

This chapter describes the test I have performed on payment for the public transport system in the Netherlands which is called the OV-chipkaart[14]. Last year two students from the University of Amsterdam contacted me for advice during their research on the disposable OV-chipkaart[15]. Together we discussed cases that could be exploited and tried to get the Ghost operational for their tests. At that time the Ghost was not stable enough to be used so they tried to find functional problems in the system. In this they succeded and published together with Translink Systems a solution to the software bug. Hardware analysis was still not performed, until the Ghost was ready to be used. This is where my research continues theirs.

At the moment it is still in a test-phase and only fully available in the city of Rotterdam. To do my tests I have travelled to this city and bought some tickets from the ticket machine. There were two different tags available at the ticket machine. The first one was a subscription card, which is used to store an amount of travel money. You can recharge the card at any ticket machine. A certain amount of money will be subtracted after you have traveled from a check-in to a check-out point by your destination. After recharging this card could in theory be reused unlimited times. The second type of tickets is very different from the first. The later one is called a disposable ticket and is available as two-trip ticket or a few-days traveling ticket. As can be expected, the first ticket is a Mifare Classic tag, while the second disposable ticket is a Mifare Ultralight tags. The tests that are performed on the dutch OV-chipkaart but the same princeples could be applied for all public transport systems world-wide which are using the same techniques. *Figure 13* shows an overview of different countries and their used RFID tags.

When a traveler wants to enter the station he needs to check-in. This means tickets needs to be verified and updated by the entrance gate. The update contains the location and time of the check-in. With this information stored on the ticket, the traveler will be able to retrieve his travel history.

Figure 14 shows an entrance gate which a traveler must pass. The traveler takes his ticket and holds it in front of the round white and pink sign on top of the gateway. When the ticket is verified, a green light appears and the entrance gate will open so that the traveler can pass.

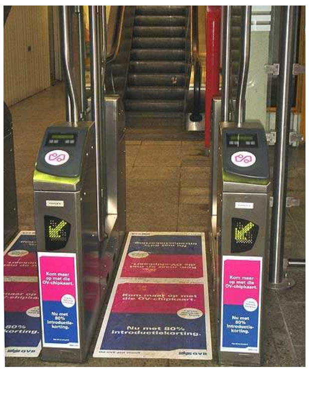
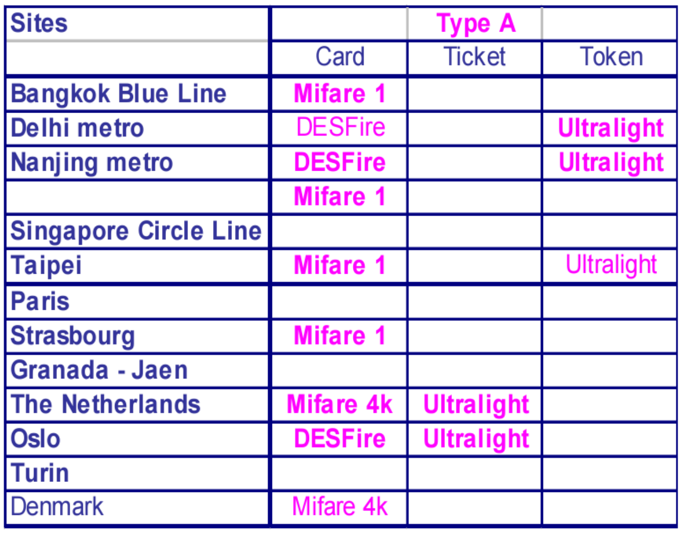


Figure 14: OV-Chip gate

After a check-in, a check-out at the same station is possible unlimited times. This could be useful when the traveler forgot something and needs to return to the other side of the gates. Though it is not possible to check-in two times with the same ticket. Traveling with one tag and pay for your friends is therefore not possible.

During my tests it pointed out that the field of the reader in the gateways are very strong. This is a good thing for performance. Travelers do not need



Sites		Type A	
	Card	Ticket	Token
Bangkok Blue Line	Mifare 1		
Delhi metro	DESFire		Ultralight
Nanjing metro	DESFire		Ultralight
	Mifare 1		
Singapore Circle Line			
Taipei	Mifare 1		Ultralight
Paris			
Strasbourg	Mifare 1		
Granada - Jaen			
The Netherlands	Mifare 4k	Ultralight	
Oslo	DESFire	Ultralight	
Turin			
Denmark	Mifare 4k		

Figure 13: Worldwide usage of RFID tags in public transport

to take out their ticket out of their bags or wallets, they only have to wave the bag in front of the reader to gain access to the station. The side-effect of this is that it is possible to catch the reader signal from a distance, which allows the eavesdropper to recover the transmitted information. The information leakage depends on the implementation of the OV-Chip, which we will investigate later in this chapter.

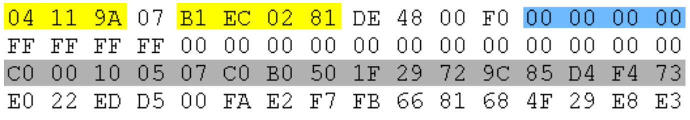
8.2.1 Disposable ticket

This chapter describes the OV-Chipkaart disposable tickets. This is the procedure of a traveler using the disposable ticket:

1 Buy a deactivated ticket at a ticket machine
2 Activate the ticket by first use
3 Check-in
4 Check-out
5 When trips available go to 3 for next trip
6 Throw away the used ticket

First an inspection of the original deactivated ticket bought at the central station. The memory of Mifare Ultralight tags is accessible by any reader. This made it rather easy to dump the total content of the memory. *Figure 15* shows the content of a disposable traveling ticket in original deactivated state.

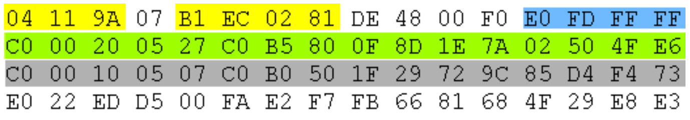
The UID of this tag is 7 bytes long and is present in the yellow marked hexadecimal bytes. The values 07 and DE take no part in the UID, these are



```
04 11 9A 07 B1 EC 02 81 DE 48 00 F0 00 00 00 00
FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00
C0 00 10 05 07 C0 B0 50 1F 29 72 9C 85 D4 F4 73
E0 22 ED D5 00 FA E2 F7 FB 66 81 68 4F 29 E8 E3
```

Figure 15: Content of original disposable ticket

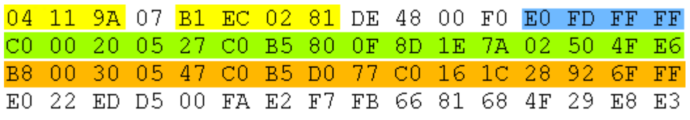
the BCC bytes which are used in the response of the anti-collision cascade levels. The second line starts with 4 times FF, which indicates an empty transaction field. This line is used to store the next transaction. The third line which is marked grey contains transaction info. The ticket is not yet used on a gateway, so this is the first transaction stored by the ticket machine, it contains the date, time and place of selling. Figure 16 describes the content of the ticket after a check in is performed.



```
04 11 9A 07 B1 EC 02 81 DE 48 00 F0 E0 FD FF FF
C0 00 20 05 27 C0 B5 80 0F 8D 1E 7A 02 50 4F E6
C0 00 10 05 07 C0 B0 50 1F 29 72 9C 85 D4 F4 73
E0 22 ED D5 00 FA E2 F7 FB 66 81 68 4F 29 E8 E3
```

Figure 16: Content of disposable ticket after check-in

The first transaction is still present in the memory. The new travel transaction was stored in the second green line. The One Time Programmable, annotated by the blue color is dramatically changed in comparison to the original content. It appears that at an activation of the ticket the counter is initialized. The memory content of the tag after checking-out contained these bytes. After checking out the content of the disposable ticket changed to the state shown in *Figure 17*.



```
04 11 9A 07 B1 EC 02 81 DE 48 00 F0 E0 FD FF FF
C0 00 20 05 27 C0 B5 80 0F 8D 1E 7A 02 50 4F E6
B8 00 30 05 47 C0 B5 D0 77 C0 16 1C 28 92 6F FF
E0 22 ED D5 00 FA E2 F7 FB 66 81 68 4F 29 E8 E3
```

Figure 17: Content of disposable ticket after check-out

The check-out is written on line three. At this point a complete transaction, namely a check-in and check-out is stored in the tag. The One Time Programmable memory is not changed. The second green line containing the check-in will be available until a next check-in is performed. Due the limitation of storage only two transactions can be derived from the memory content of a disposable ticket.

Every transaction is written in line two or three. It depends on which line was used for the last transaction, the transaction before the last transaction will be overwritten. This means that when a transfer took place, from one transport system to another, the original check-in line will get lost. A more detailed explanation of the transactions is presented in *Table 1* which is extracted from the paper published earlier this year about the disposable OV-Chip tag[15].

Bits	Function	Comments
0-4	Unknown	Values: 10101, 10110, 10111, 11000, 11001. Appears to follow a regular pattern.
5-19	Transaction Counter	Regular counter.
20-31	Location	Values: 010 = Amsterdam, 101 = Rotterdam.
32-34	Transaction Type	Values: 000 = Purchase, 001 = Check-in, 010 = Check-out, 110 = Transfer (Overstap).
35-48	Date	Number of days since January 1, 1997.
49-59	Time	Number of minutes since the start of the day.
60-63	Unknown	This value is always 0. Probably unused.

Table 1: Transaction of a disposable OV-chipkaart[15]

An interesting feature of the Ghost is that it can store the commands requested by the gateway in a buffer. After checking in with the ticket we can analyze the new memory content stored on the ticket. Secondly we can view the complete transaction. This transaction trace contains the commands send by the original disposable OV-Chip Tag and the gateway during a valid check-in was the following.

#	Length	Command	CRC	Description
1	4	`30 00`	`02 A8`	Read bytes 0-15
2	4	`30 04`	`26 EE`	Read bytes 16-31
3	4	`30 08`	`4A 24`	Read bytes 32-47
4	4	`30 0C`	`6E 62`	Read bytes 48-63
5	8	`A2 04 00 00 00 00`	`37 92`	Write bytes 16-19
6	8	`A2 05 27 C0 B5 80`	`45 8B`	Write bytes 20-23
7	8	`A2 06 0F 8D 1E 7A`	`89 16`	Write bytes 24-27
8	8	`A2 07 02 50 4F E6`	`F8 72`	Write bytes 28-31
9	8	`A2 04 C0 00 20 05`	`70 DD`	Write bytes 16-19
10	8	`A2 03 60 FD FF FF`	`FC B8`	Write bytes 12-15
11	8	`A2 03 E0 FD FF FF`	`92 95`	Write bytes 12-15

The 30 indicates a read and the A2 a write command.

```
            30 XX = Reads from page XX until XX+3
A2 XX ZZ ZZ ZZ ZZ = Writes Z to memory page XX
```

The anti-collision is excluded from this trace because it adds no valuable information. This sequence was similar to the one described in *Chapter 5.1.* The read command requests 16 bytes at once, while the write command can only write 4 bytes at the time. The trace shows that *commands 1 to 4* read out the complete 64KB of memory from the tag. Then the gateway processes the information, validates it and updates parts of the memory. An interesting point is that the gateway starts with writing only zeros to memory page 0x04 as you can see at *command 5.* Then it updates memory pages 0x05 until 0x07 and completes the update with a write on page 4 again at *command 9.* This indicates support for some kind of verifying mechanism that ensures an update is completed and not interrupted in any way. The *commands 10 and 11* will write to the One Time Programmable counter which is located in memory page 0x03. The question that raises here is, why does the system tries to write to the same memory page 2 times. After some research it occurred that only the first time a tag is used it will receive the command to write ”60...” to the page. After this, all transactions will only write ”E0...” to the page. The binairy representation of 0x60 is 01100000 while 0xE0 stands for 1110000. This means only one bit more is triggered. This is strange since it is writing in the one time programming memory. So far, I have found no real explanation for this strange behaviour of the counter.

The check-out is quite similar. Though it will write in a different memory space. Next there is a trace of a valid check-out.

#	Length	Command	CRC	Description
1	4	`30 00`	`02 A8`	Read bytes 0-15
2	4	`30 04`	`26 EE`	Read bytes 16-31
3	4	`30 08`	`4A 24`	Read bytes 32-47
4	4	`30 0C`	`6E 62`	Read bytes 48-63
5	8	`A2 08 00 00 00 00`	`07 E5`	Write bytes 32-35
6	8	`A2 09 47 C0 B8 F0`	`6E A6`	Write bytes 36-39
7	8	`A2 0A 70 70 EE 37`	`A7 1F`	Write bytes 40-43
8	8	`A2 0B E0 B3 AD CB`	`3F 4E`	Write bytes 44-47
9	8	`A2 08 B8 00 30 05`	`4C 80`	Write bytes 32-25

Interesting is that during a check-out nothing is written to page 0x03. This means that the One Time Programmable memory is only invoked at the start of the trip. A second thing that is good to notice is the encrypted part of this memory written in *Command 7 and 8* are probably very weakly protected. It could be accidental that the signature looks like ”70 70 EE 37 E0 B3 AD CB”. But in my opinion this looks like there is a strong weakness in the hashing algorithm. The bytes in this sequence are quite similar and does not look random at all, this suggests that at least some crypto analysis would be very likely to reveal relations.

During the tests with the OV-Chip I was able to reconstruct a valid ticket that could be reused an unlimited number of times. I demonstrated this to the public in a news item on 14 January 2008[16]. To explain the details of this attack to the public there was a reference document released for newspapers and their journalists. This document is distributed by the official web site of the Radboud University. This chapter describes the applied attack in more details compared to the paper released in January.

The aim of the attack was to simulate free traveling. Please keep in mind that as a student you own a free traveling pass OV-Studentenkaart[17], so I was not breaking any law in that sense.

The 7 bytes UID of the Mifare Ultralight tag should be part of the key during the encryption of the content. This is a useful way to protect the data against duplicates. When you try to copy the content of one tag to another, the system can not decrypt the content anymore since the other tag has a different UID. All the current available RFID chips present a unique identifier. This means that until a manufacturer start to ship programmable UID tags there is no real danger in cloning one tag to another. But we should not forget that this setup is quite comparable to the network MAC-Address scene. The IEEE proposed usage of the Extended Unique Identifier[19] decades ago. But with the new EEPROM chips used in the network interfaces, changing of MAC-Adresses is as easy as changing an IP address. This could mean that we will probably see a similar thing happen to the unique identifiers in the near future. As described earlier the Ghost can reproduce any UID that is needed. To clone a public transport ticked this is one of the main requirements.

The encryption methods to protect the content written on the card could be perfect, but still would be vulnerable against an replay attack. It is good to keep in mind that a replay attack is known in the security world for many years now. Typically what is done during an replay attack is that someone eavesdrops the information that is presented to the listening party and replays the same information in a later session. The eavesdropper does not need any knowledge about the plaintext. This works unless there are taken some counter measures against it. A counter measure is to use of sequence numbers which are registered on both sides, sender and receiver. When a earlier session is replayed, the receiver detects that this sequence number is already used during an earlier session and revokes the request. The problem with keeping track of a sequence number means that the receiving party should be up to date. In terms of a public transport system with a lot of gates it means that the system should be online. Otherwise one gate does not know the sequence numbers of a gate next to it. The OV-Chipkaart currently does not use an online system. It relies purely on the validity of the presented information of a tag.

Since the transaction and transferred information is known an attack could be performed. The content of a freshly bought disposable ticket is retrieved by a ordinary reader, in this attack the Omnikey Contactless Reader[4] was used. The original 64 bytes of memory are copied into the Ghost. All the commands that are defined in the Mifare Ultralight Specification[11] are implemented by the Ghost. A read and write operation is processed in the ghost exactly the same as an original tag would. After completing the operations the ghost starts to investigate the changes and resets the internal memory back to the original state. This means that even after a check-in we can directly present an "original" ticket again. In a normal case you would not be able to check-in with multiple people using only one ticket. In this case though you could enter the sub-way with a very large group, just by passing the ghost to the person behind you in line. When the last person has checked-in to the system the ghost will save the modified memory state. This state is needed to check out again. You can not check-out with an original ticket without having a valid check-in stored into it.The checking out with the group is performed the same way as the check-in,

[4]http://omnikey.aaitg.com

the ghost will provide every malicious traveler a valid check-out ticket.

The tickets that are available for sale are 1,2 trip tickets and 1,2,3 days traveling. At the time of buying the ticket is not ”activated” yet. Activation takes place when a ticket is linked to the first day of usage. It would be possible to buy tickets in advance which get activated by first use. An activated ticket gets disabled after completing the trips payed for, a inactivated ticket though, stays valid undefinitly. This means that the original inactivated ticket which is stored in the memory of the Ghost will never expire. It will get activated every time a check-in is performed, but it will reset immediately to the inactivated state again, when the next check-in is required.

What is good to keep in mind is that we still have our original ticket in our pocket. This ticket is never used for a check-in. When a employee of the transport company wants to check the validity of the ticket, the memory contents of the check-in available in the Ghost can be copied back to the ticket. This means the ticket contains completely valid data. In general it means that a malicious traveler only has to buy a new ticket when the original ticket is compromised.

Interesting to know is that the lock-bits are worthless when it comes to cloning tickets. The Ultralight tag prevents memory changes to be made when certain lock-bits are flagged. Using the Ghost device this prevention is controlled by the Ghost itself. This integrity feature is completely ignored, for an attacker it is far more useful to change memory even after it gets locked. The Ghost though should respond to a gate as the tag would have done. It shows that the memory is locked and can not be changed anymore, while in the background it manipulates the memory in such a way that earlier states can be back loaded into the memory.

The One Time Programmable counter is vulnerable in the same way as the lock-bits. After a certain state is reached, for example that all the counter bits are flagged, no increment could be enforced anymore. Then again, the Ghost controls this memory completely, which enables it to completely reset the counter.

8.2.2 Subscription card

Mifare Classic tags are used for the subscription cards. The subscription card comes in two types, the anonymous and the personal form. The anonymous one is like a prepaid card, it is chargeable with a certain amount of money which can be used for traveling. The tag is called anonymous because it should have no direct link to a person. In practice this is only true if you always top-up your card with cash money. When a traveler ever recharges his anonymous card via a bank transaction the card can be linked to a name.

Secondly, the UID of Mifare Classic tags are static and more important, unique. For instance, the passport contains a chip that generates a random UID every time it is challenged by a reader. This would not allow any 3rd party to track people movements. Though realistic is to say that next to the passport a person always would carry his own subscriber or anonymous OV-Chipkaart. The counter measure in the passport against tracking becomes completely useless if other RFID tags reveal the information the passport is trying to hide.

Mifare Classic tags were chosen because it was field proven technology at the time of selecting the technical infrastructure in 2001. The technique was stated to be field proven because of the use in a lot of big cities as replacement for obsolete paper traveling tickets. The technique never was certified complient to the Common Criteria[20] for Information Technology Security Evaluation. There is much more memory available in the subscription card. They use Mifare Classic 4K tags to store the information. In this memory a history of previous trips is stored. At any recharge point the traveler is able to check the amount that is available and recent traveled trips. In the Dutch system they use the 4KB version instead of the more used 1KB version because all the transport companies involved claimed part of the storage space for their own gathered information. The result is that a check-in transaction generates quite some overhead. There is 15 times as much communication compared to a check-in transaction with the disposable tag. The communication is normally encrypted using the CRYPTO1 algorithm, but with help of our key-retrieval and decryption tools[21] the plaintext transaction could be revealed. This is the decrypted trace of a check-in transaction.

```
  1 | RD  | ok  | AUTH      | 60 FF 8D 74
  2 | TAG |  -  | Nt        | BA 6A 16 8E
  3 | RD  |  -  | Nr + Nt’  | C2 69 12 BC 57 85 82 60
  4 | TAG |  -  | Nt"       | B6 87 90 32
  5 | RD  | ok  | READ      | 30 FB 5E E1
  6 | TAG | ok  | DATABLOCK | 9B 00 03 20 01 23 45 60 12 34 56 78 9A B0 12 34 E1 E2
  7 | RD  | ok  | READ      | 30 FC E1 95
  8 | TAG | ok  | DATABLOCK | 56 78 9A B2 34 56 78 9A 0C 12 34 56 78 9A B8 60 6A 1E
  9 | RD  | ok  | READ      | 30 FD 68 84
 10 | TAG | ok  | DATABLOCK | 9D 00 03 60 01 23 45 60 12 34 56 78 9A B0 12 34 E3 BD
 11 | RD  | ok  | READ      | 30 FE F3 B6
 12 | TAG | ok  | DATABLOCK | 56 78 9A B1 23 45 67 89 A0 12 34 56 78 9A B8 00 40 7B
 13 | RD  | ok  | AUTH      | 60 5F 87 D1
 14 | TAG |  -  | Nt        | D3 5C 9B 3A
 15 | RD  |  -  | Nr + Nt’  | 8B C9 41 D8 93 CD 29 C9
 16 | TAG |  -  | Nt"       | 0F BC 6D 09
 17 | RD  | ok  | READ      | 30 5C EB 30
 18 | TAG | ok  | DATABLOCK | 0E 02 94 00 00 00 22 8A C1 4B C0 00 00 00 00 00 03 5B
 19 | RD  | ok  | READ      | 30 5D 62 21
 20 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
 21 | RD  | ok  | READ      | 30 5E F9 13
 22 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
...
102 | TAG |  -  | ACK       | 0A
103 | RD  | ok  | DATABLOCK | 20 00 80 00 00 00 80 00 D0 04 0C A0 00 00 00 00 33 67
104 | TAG |  -  | ACK       | 0A
105 | RD  | ok  | WRITE     | A0 FC BC 8C
106 | TAG |  -  | ACK       | 0A
107 | RD  | ok  | DATABLOCK | 56 78 9A B0 12 34 56 78 9C 12 34 56 78 9A B8 60 43 4E
108 | TAG |  -  | ACK       | 0A
109 | RD  | ok  | WRITE     | A0 FB 03 F8
110 | TAG |  -  | ACK       | 0A
111 | RD  | ok  | DATABLOCK | 9B 00 03 A0 01 23 45 60 12 34 56 78 9A B0 12 34 43 24
112 | TAG |  -  | ACK       | 0A
113 | RD  | ok  | HALT      | 50 00 57 CD
```

The exact meaning of the transmitted bytes is unknown to us at the moment of writing this thesis. The company that implements the OV-Chipkaart in the netherelands, Trans Link Systems (TLS)[5], only shares this information under a No Disclosure Agreement (NDA). It is clear that at least parts of this trace are not stored encrypted.

For instance, on line 6 the block with the index 0xFB is read containing the following values

```
9B 00 03 20 01 23 45 60 12 34 56 78 9A B0 12 34
         ^
```

Line 111 shows a new value that is written to the same block

```
9B 00 03 A0 01 23 45 60 12 34 56 78 9A B0 12 34
         ^
```

The new value has only one bit changed (20=00100000) => (A0=10100000). This could be some kind of transaction number. The information stored in these blocks look very fimilar. Obviously they wanted to avoid to much zeros in memory, but it is hard to believe that the hexidecimal counter in the last part of the block actually means something useful. A complete check-out trace can be found in Appendix A. It was automaticly decrypted with a tool created by Ruben Muijrers[18].

8.3 Entrance access

The access control of our university building is protected by RFID tags that contain chips with the Mifare Classic technology. These tags are presented by the manufacturer as a secure solution for several applications. During my research of these RFID chips it came clear that the protections are not as strong as the manufacturer claims it to be.

The access control of our building uses the same tags as the university parking, the difference though is that, to gain access to the building the gate does not verifies the UID. It requests a identification number stored in memory sector 0x30 on the tag. This identification is checked against an online database, when the ID is authorized the systems grants access.

This implicates that if the sector key is known a employees access tag can be challenged with a regular reader to retrieve its identification number. This number can be stored in any Mifare Classic tag. A demonstration[22] shows that a blank manufacturer tag can be used for this. The key that is retrieved is programmed into the manufacturer tag together with the retrieved employees identification. This is a very high security risk since you can mass-produce fake access tags very quickly with low costs. While checking the UID will not solve any weaknesses in the Mifare Classic algorithm, it makes it harder to gain access to this system. For example, the UID could be used to generate a key per tag. This is called diversified keys. Here is an example of a process that uses diversified keys.

[5]http://www.translink.nl

Reader	Tag
Get UID	
	Send UID
Encrypt(UID,sector#) with master key	
Authenticate for sector# using generated key	
	Verify key

In this example the master key is only known by the firmware chip in the reader. To be sure the firmware is not recovered or tampered it should be implemented in a smartcard. Using a smartcard for this in a RFID reader is called a Secure Accsess Module (SAM).

Every tag has its own set of keys which are derived from this master key. The key needs to be transported to the Mifare Classic chip from NXP in the reader. This chip does all the encrypted communication by itself. It has its own connection to the antenna of the reader and uses this to communicate directly to a Mifare Classic tag which is shown in *Figure 18*. There is a major problem with this. The communication between the firmware and the chip from NXP is not protected in any way. An attacker would be able to tap the pin where key transmitted and eavesdrop the key without need of any special knowledge about the system. This way one key for one specific card is recovered, while the master key will still be secret. Last november NXP presented a new chip to bring a solution to this problem. This chip is compatible with the formerly used Mifare Classic chip but also implements algorithms for diversifying keys. This way it would be possible to load the master key into this smartcard and let it calculate internally the diversified key for the presented tag. This smartcard should be fully compatible with current SAM modules available in the market.

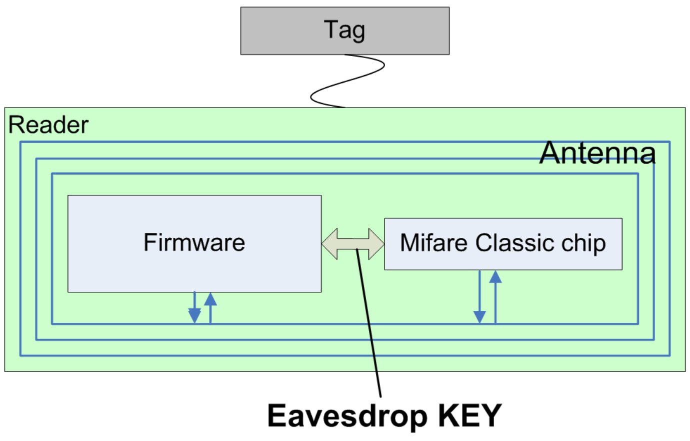


Figure 18: Eavesdrop diversified key

To fake the UID a device like the Ghost is needed. This device needs to implement the Mifare Classic algorithm to successfully communicate with the reader. Gerhard de Koning Gans implemented these features on the Proxmark device[12]. In a public demonstration we have shown the easiness of eavesdropping a tag, retrieving the key, decrypting the communicated information and impersonating someone to gain unauthorized access.

9 Conclusions

Security is a real issue when wireless techniques like RFID are globally used. At several crowded places, tags can be cloned without the need of touching any victim. Impersonalisation can be reached within seconds. Digital money stored in a tag can be multiplied without any loss of integrity like signatures.

The general conclusion that can be drawn from this reseach is that most RFID tags that are in use do not provide any real security. The Mifare Ultralight tag does not provide any communication security at all, while the more trusted bigger brother the Mifare Classic is proven to be almost just as weak. The information that is stored in the tags can be retrieved very easyly. Furthermore it can be used to create a copy which is indistictable for the reader. This is a major security issue because it allows a mallicious user to eavesdrop and copy access keys from the distance.

There are new high-end alternatives available[25]. These tags provide encryption using widely used and proven algorithms like DES, 3DES and AES. But even when the tags themselfs provide enough security features, the applications using this techniques should be careful in designing the protocol. For example, recent studies[26][27] at our department show weaknesses found in the new Dutch passport using the newest technologies.

The public should be aware of these security threats. The trust that is put into RFID tags is often much more thant it deserves. And last but not least, encryption algorithms should not base their security on secrecy of the system. In general, it is far better to use well-established and well-reviewed cryptographic primitives and protocols than proprietary ones. As was already formulated by Auguste Kerckhoffs in 1883, and what is now known as Kerckhoffs Principle, the security of a cryptographic system should not depend on the secrecy of the system itself, but only on the secrecy of the key[23]. So many times it is proven that details of the system will eventually become public; the previous obscurity then only leads to a less well-vetted system that is prone to mistakes. Examples of other systems which turned out to be insecure because of applying security by obscurity are Bluetooth[28], DVD CCS protection[29], Mobile GSM system[30], Wireless Internet, Wired Equivalent Privacy (WEP)[31] and many more.

References

[1] ISO/IEC 14443. *Identification cards - Contactless integrated circuit(s) cards - Proximity cards*, 2001.

[2] ISO/IEC 11785. *Radio-frequency identification of animals. Technical concept*, 1997.

[3] ISO/IEC 15693. *Identification cards - Contactless integrated circuit(s) cards - Vicinity Integrated Circuit(s) Card*, 2001.

[4] Escorics *13th European Symposium on Research in Computer Security* Malaga, Spain, 2008.

[5] Timo Kasper, Dario Carluccio, Christof Paar. *An Embedded System for Practical Security Analysis of Contactless Smartcards*, 2003.

[6] I. Kirschenbaum and A. Wool. *How to build a low-cost, extended-range RFIDskimmer* Cryptology ePrint Archive, Report 2006/054, 2006.

[7] Craig Ross and Ricardo Goto. *Proximity Security System RFIDskimmer*, 2006.

[8] Thomas S. Heydt-Benjamin , Daniel V. Bailey , Kevin Fu , Ari Juels , and Tom OHare. *Vulnerabilities in First-Generation RFID-enabled Credit Cards*, Eleventh International Conference on Financial Cryptography and Data Security Scarborough, Tobago, 2007.

[9] Gerhard P. Hancke. *Practical Attacks on Proximity Identification Systems (Short Paper)*,SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy, pages 328-333 http://www.cl.cam.ac.uk/ gh275/SPPractical.pdf, 2006.

[10] Philips Semiconductors. *Mifare Standard 4 kByte Card IC - MF1 IC S70 - Functional Specification - Rev. 3.1*, 2002.

[11] Philips Semiconductors. *Mifare Ultralight - MF0 IC U1 - Contactless Single-trip Ticket IC - Functional Specification - Rev. 3.0*, 2003.

[12] G. de Koning Gans, J.-H. Hoepman, and F. D. Garcia. *A practical attack on the MIFARE classic*, 2008.

[13] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. *Dismantling MIFARE Classic*, 2008.

[14] Trans Link Systems (TLS). *OV-chipkaart Project*, http://www.ov-chipkaart.nl, 2008.

[15] Pieter Siekerman and Maurits van der Schee. *Security Evaluation of the disposable OV-chipkaart*, v1.6, 2007.

[16] Roel Verdult. *Proof of concept, cloning the OV-Chip card*, http://www.cs.ru.nl/ flaviog/OV-Chip.pdf, 2008.

[17] Informatie Beheer Groep. *OV Studentenkaart*, http://www.ib-groep.nl, 2008.

[18] Ruben Muijrers. *Mifare Trace Decrypter (MiTraDe)*, Not public available, 2008.

[19] Institute of Electrical and Electronics Engineers, Inc. *Guidelines for use of a 48-bit Extended Unique Identifier*, http://standards.ieee.org/regauth/oui/tutorials/EUI48.html, 2008.

[20] ISO/IEC 15408. *Information technology – Security techniques – Evaluation criteria for IT security*, Second Edition, 2005.

[21] Ruben Muijrers, Peter van Rossum, Ronny Wichers Schreur. *Mifare Toolkit*, 2008.

[22] Radboud University, Digital Security. *Visual demonstration of the Mifare Hack*, http://nl.youtube.com/watch?v=NW3RGbQTLhE, 2008.

[23] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX, 1883. pp. 5–38, Jan. 1883, and pp. 161–191, Feb. 1883.

[24] Karsten Nohl and Henryk Plötz. Mifare, little security, despite obscurity. Presentation on the 24th Congress of the Chaos Computer Club in Berlin, December 2007.

[25] Philips Semiconductors. *SmartMX platform features, Secure Smart Card Controller Platform*, Short Form Specification, Rev 1.0, 2004.

[26] J.-H. Hoepman, E. Hubbers, B. Jacobs, M. Oostdijk, and R. Wichers Schreur. Crossing borders: Security and privacy issues of the european e-passport. In Hiroshi Yoshiura, Kouichi Sakurai, Kai Rannenberg, Yuko Murayama, and Shinichi Kawamura, editors, *Advances in Information and Computer Security. International Workshop on Security (IWSEC 2006)*, volume 4266 of *Lecture Notes in Computer Science*, pages 152–167. Springer Verlag, 2006.

[27] Henning Richter, Wojciech Mostowski, and Erik Poll. *Fingerprinting Passports*, NLUUG 2008 Spring Conference on Security, Ede, the Netherlands, 2008.

[28] Yaniv Shaked and Avishai Wool. *Cracking the Bluetooth PIN*, Proceedings of the 3rd international conference on Mobile systems, applications, and services, June 06-08, Seattle, Washington, 2005.

[29] Frank A. Stevenson. *Cryptanalysis of Contents Scrambling*, 1999.

[30] Elad Barkan, Eli Biham, and Nathan Keller. *Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication*, Advances in Cryptology CRYPTO 2003, volume 2729 of Lecture Notes in Computer Science, 2003.

[31] Erik Tews, Ralf-Philipp Weinmann, Andrei Pyshkin. *Breaking 104 bit WEP in less than 60 seconds*, 2007.

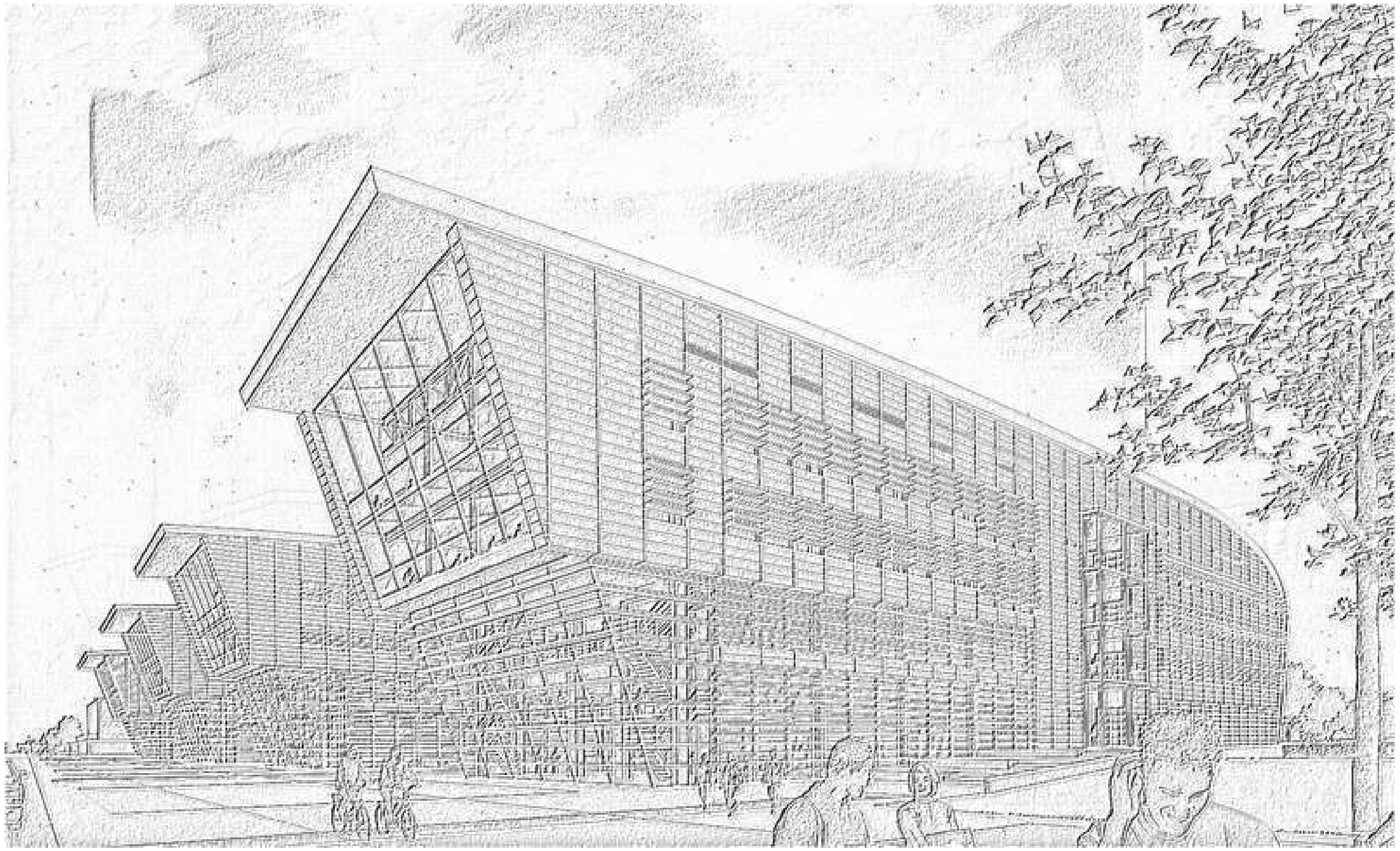
10 Appendix A: Full decrypted check-out trace OV-chipkaart

```
-------------------------------------------------------------------------------
Plain Mifare trace
Decrypted with MiTraDe by Ruben Muijrers
MiTraDe version 1.7a
Decrypted on Fri Apr 04 20:25:01 2008
----+-----+-----+-----------+-------------------------------------------------
Idx | Src | CRC |   Type    | Packet
----+-----+-----+-----------+-------------------------------------------------
  0 | RD  |  -  | REQUEST   | 26
  1 | TAG |  -  | AWAKE     | 02 00
  2 | RD  |  -  | ANTI COLL | 93 20
  3 | TAG |  -  | UID       | 26 05 7E D1 8C
  4 | RD  | ok  | ANTI COLL | 93 70 26 05 7E D1 8C 86 74
  5 | TAG | ok  | TAG TYPE  | 18 37 CD
  6 | RD  | ok  | AUTH      | 60 03 6E 49
  7 | TAG |  -  | Nt        | 8F 08 F9 A3
  8 | RD  |  -  | Nr + Nt'  | 64 CE F5 8D 28 B4 06 FA
  9 | TAG |  -  | Nt"       | 17 44 B0 DF
 10 | RD  | ok  | READ      | 30 01 8B B9
 11 | TAG | ok  | DATABLOCK | 84 00 00 00 06 03 A0 00 13 AE E4 01 5C 18 0E 80 FC 86
 12 | RD  | ok  | READ      | 30 02 10 8B
 13 | TAG | ok  | DATABLOCK | 80 E8 40 00 00 00 00 00 00 00 00 00 00 00 00 00 21 96
 14 | RD  | ok  | AUTH      | 60 FF 8D 74
 15 | TAG |  -  | Nt        | 98 85 61 D8
 16 | RD  |  -  | Nr + Nt'  | FC 65 EE 77 83 0F EF 55
 17 | TAG |  -  | Nt"       | 86 38 73 08
 18 | RD  | ok  | READ      | 30 FB 5E E1
 19 | TAG | ok  | DATABLOCK | 9B 00 07 20 01 23 45 60 12 34 56 78 9A B0 12 34 58 11
 20 | RD  | ok  | READ      | 30 FC E1 95
 21 | TAG | ok  | DATABLOCK | 56 78 9A B9 A0 12 34 56 7C 02 34 56 78 9A B6 20 3A 1F
 22 | RD  | ok  | READ      | 30 FD 68 84
 23 | TAG | ok  | DATABLOCK | 9C 00 07 60 01 23 45 60 12 34 56 78 9A B0 12 34 4A C0
 24 | RD  | ok  | READ      | 30 FE F3 B6
 25 | TAG | ok  | DATABLOCK | 56 78 9A B8 9A 01 23 45 6C 02 34 56 71 9A B6 40 77 5D
 26 | RD  | ok  | AUTH      | 60 5F 87 D1
 27 | TAG |  -  | Nt        | 98 59 01 2E
 28 | RD  |  -  | Nr + Nt'  | CA 07 C8 21 34 AA AE A0
 29 | TAG |  -  | Nt"       | 95 D1 4E C9
 30 | RD  | ok  | READ      | 30 5C EB 30
 31 | TAG | ok  | DATABLOCK | 0E 02 94 00 00 00 22 8A C1 4B C0 00 00 00 00 00 03 5B
 32 | RD  | ok  | READ      | 30 5D 62 21
 33 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
```

```
34 | RD  | ok  | READ      | 30 5E F9 13
35 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
36 | RD  | ok  | AUTH      | 60 FF 8D 74
37 | TAG |  -  | Nt        | 0E DF D4 D5
38 | RD  |  -  | Nr + Nt'  | 6C DD 2D C2 C1 0B 17 90
39 | TAG |  -  | Nt"       | 90 22 72 7F
40 | RD  | ok  | READ      | 30 F9 4C C2
41 | TAG | ok  | DATABLOCK | 20 00 F8 00 00 00 80 01 C0 04 12 E0 00 00 00 00 E2 14
42 | RD  | ok  | READ      | 30 F1 04 4E
43 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
44 | RD  | ok  | READ      | 30 F2 9F 7C
45 | TAG | ok  | DATABLOCK | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 49
46 | RD  | ok  | READ      | 30 F7 32 2B
47 | TAG | ok  | DATABLOCK | 9C 01 59 08 01 60 00 05 70 01 58 00 55 00 15 00 40 2B
48 | RD  | ok  | READ      | 30 F8 C5 D3
49 | TAG | ok  | DATABLOCK | 05 34 01 62 40 16 10 00 00 00 00 00 00 00 00 00 80 8A
50 | RD  | ok  | AUTH      | 60 EF 0C 64
51 | TAG |  -  | Nt        | 11 B0 97 17
52 | RD  |  -  | Nr + Nt'  | 75 54 E8 BF A3 9F 42 7F
53 | TAG |  -  | Nt"       | 80 6C 54 59
54 | RD  | ok  | READ      | 30 EA 56 E0
55 | TAG | ok  | DATABLOCK | 08 10 55 04 03 E0 00 00 28 00 00 A8 1F 80 20 02 0F D9
56 | RD  | ok  | READ      | 30 EB DF F1
57 | TAG | ok  | DATABLOCK | 0A 00 FA 00 00 00 00 00 00 00 00 00 00 00 00 00 AC A9
58 | RD  | ok  | READ      | 30 EC 60 85
59 | TAG | ok  | DATABLOCK | 08 10 55 04 03 DB 58 00 20 00 00 10 10 90 20 02 AD FC
60 | RD  | ok  | READ      | 30 ED E9 94
61 | TAG | ok  | DATABLOCK | 3A 00 FA 00 00 00 00 00 00 00 00 00 00 00 00 00 25 43
62 | RD  | ok  | AUTH      | 60 CF 0E 45
63 | TAG |  -  | Nt        | 16 56 E1 23
64 | RD  |  -  | Nr + Nt'  | DA 31 8C 83 6B 2B 87 3D
65 | TAG |  -  | Nt"       | BA 24 26 58
66 | RD  | ok  | READ      | 30 C2 1C 4D
67 | TAG | ok  | DATABLOCK | 28 00 55 44 03 E0 40 10 00 50 00 01 F0 3F 00 10 21 79
68 | RD  | ok  | READ      | 30 C3 95 5C
69 | TAG | ok  | DATABLOCK | 0E 50 19 00 00 00 00 00 00 00 00 00 00 00 00 00 A8 5B
70 | RD  | ok  | READ      | 30 CA 54 C1
71 | TAG | ok  | DATABLOCK | 29 00 55 44 03 E0 40 10 00 50 00 01 F0 3F 00 10 31 F7
72 | RD  | ok  | READ      | 30 CB DD D0
73 | TAG | ok  | DATABLOCK | 0E 50 00 00 19 00 00 00 00 00 00 00 00 00 00 00 97 86
74 | RD  | ok  | AUTH      | 61 CF D6 5C
75 | TAG |  -  | Nt        | AB 96 61 CD
76 | RD  |  -  | Nr + Nt'  | 2A 72 46 46 F2 CD 5F 39
77 | TAG |  -  | Nt"       | E9 19 61 74
78 | RD  | ok  | WRITE     | A0 C0 53 77
79 | TAG |  -  | ACK       | 0A
```

```
 80 | RD  | ok  | DATABLOCK | 28 00 55 44 03 E0 E8 20 00 50 00 02 00 3F 30 10 A3 CA
 81 | TAG |  -  | ACK       | 0A
 82 | RD  | ok  | WRITE     | A0 C1 DA 66
 83 | TAG |  -  | ACK       | 0A
 84 | RD  | ok  | DATABLOCK | 1A 90 09 70 00 00 00 00 00 00 00 00 00 00 00 00 73 2C
 85 | TAG |  -  | ACK       | 0A
 86 | RD  | ok  | AUTH      | 61 DF 57 4C
 87 | TAG |  -  | Nt        | 86 11 5B 6F
 88 | RD  |  -  | Nr + Nt'  | 4F E0 C1 1F D3 79 53 53
 89 | TAG |  -  | Nt"       | 77 D7 FF 26
 90 | RD  | ok  | WRITE     | A0 DA 88 C8
 91 | TAG |  -  | ACK       | 0A
 92 | RD  | ok  | DATABLOCK | 29 00 55 44 03 E0 E8 20 00 50 00 02 00 3F 30 10 B3 44
 93 | TAG |  -  | ACK       | 0A
 94 | RD  | ok  | WRITE     | A0 DB 01 D9
 95 | TAG |  -  | ACK       | 0A
 96 | RD  | ok  | DATABLOCK | 1A 90 01 50 09 70 00 00 00 00 00 00 00 00 00 00 C3 A7
 97 | TAG |  -  | ACK       | 0A
 98 | RD  | ok  | AUTH      | 61 FF 55 6D
 99 | TAG |  -  | Nt        | 1D AF 41 FC
100 | RD  |  -  | Nr + Nt'  | B0 7D 42 7C 6C 95 31 4E
101 | TAG |  -  | Nt"       | 0A C6 79 41
102 | RD  | ok  | WRITE     | A0 F5 7D 11
103 | TAG |  -  | ACK       | 0A
104 | RD  | ok  | DATABLOCK | 9C 01 49 08 01 60 00 05 70 01 58 00 55 00 15 00 97 FD
105 | TAG |  -  | ACK       | 0A
106 | RD  | ok  | WRITE     | A0 F6 E6 23
107 | TAG |  -  | ACK       | 0A
108 | RD  | ok  | DATABLOCK | 05 34 01 62 40 16 10 00 00 00 00 00 00 00 00 00 80 8A
109 | TAG |  -  | ACK       | 0A
110 | RD  | ok  | WRITE     | A0 FA 8A E9
111 | TAG |  -  | ACK       | 0A
112 | RD  | ok  | DATABLOCK | 20 01 00 00 00 00 80 01 D0 04 1A A8 00 00 00 00 50 E4
113 | TAG |  -  | ACK       | 0A
114 | RD  | ok  | WRITE     | A0 FC BC 8C
115 | TAG |  -  | ACK       | 0A
116 | RD  | ok  | DATABLOCK | 56 78 9A B7 89 A0 12 34 5C 02 34 56 78 9A B6 20 74 59
117 | TAG |  -  | ACK       | 0A
118 | RD  | ok  | WRITE     | A0 FB 03 F8
119 | TAG |  -  | ACK       | 0A
120 | RD  | ok  | DATABLOCK | 9B 00 07 A0 01 23 45 60 12 34 56 78 9A B0 12 34 FA D7
121 | TAG |  -  | ACK       | 0A
122 | RD  | ok  | HALT      | 50 00 57 CD
----+-----+-----+-----------+-------------------------------------------------
```

11 Appendix B: Blueprint, design and components of the Ghost

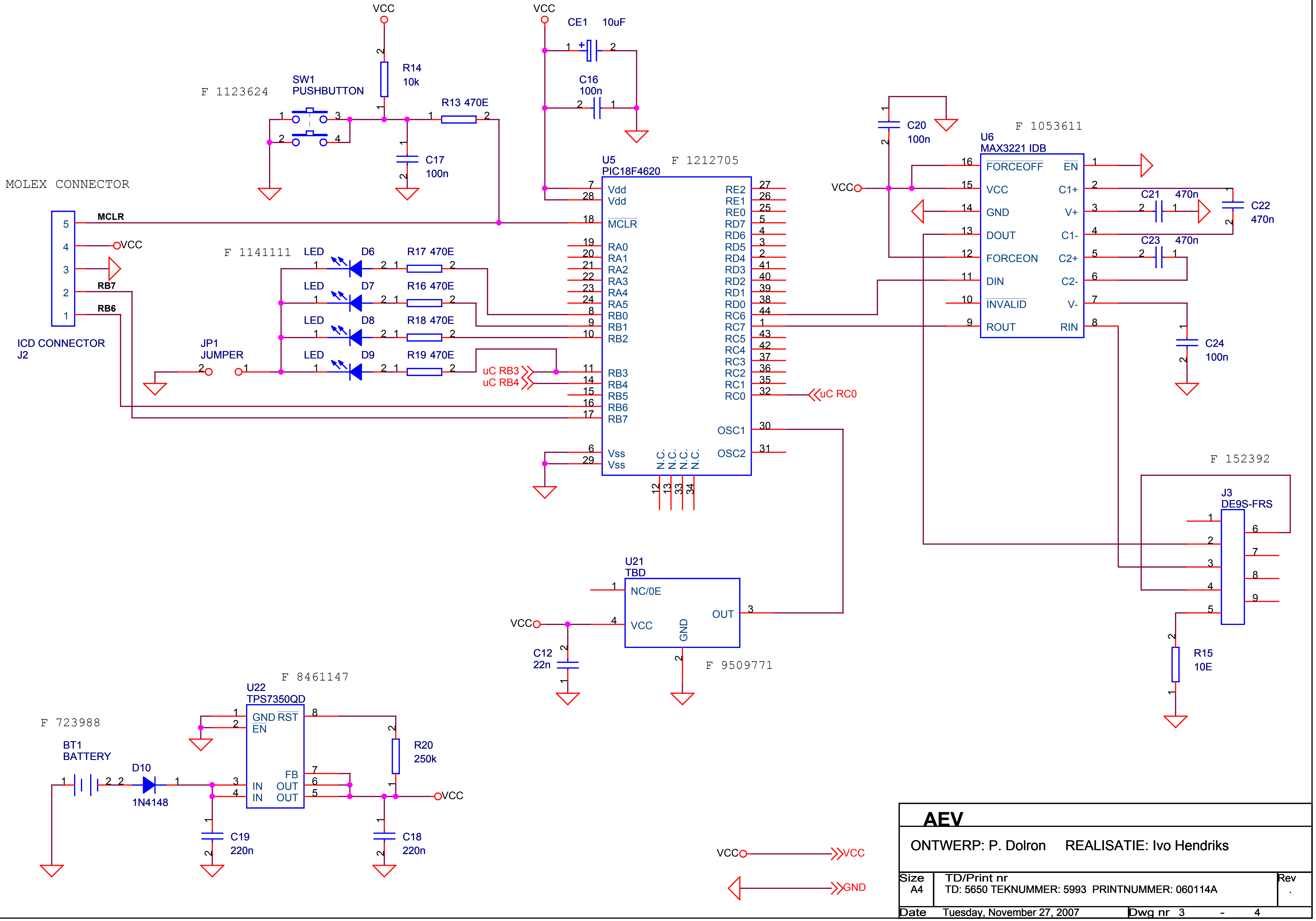


Eaves-dropping

FNWI
TechnoCentrum

EAVESDROPPING

Project nummer:	62000521
tek nummer:	5993
ontwerper:	Peter Dolron
realisatie:	Ivo Hendriks
datum:	29 november 2007
update:	---
modificatie:	---

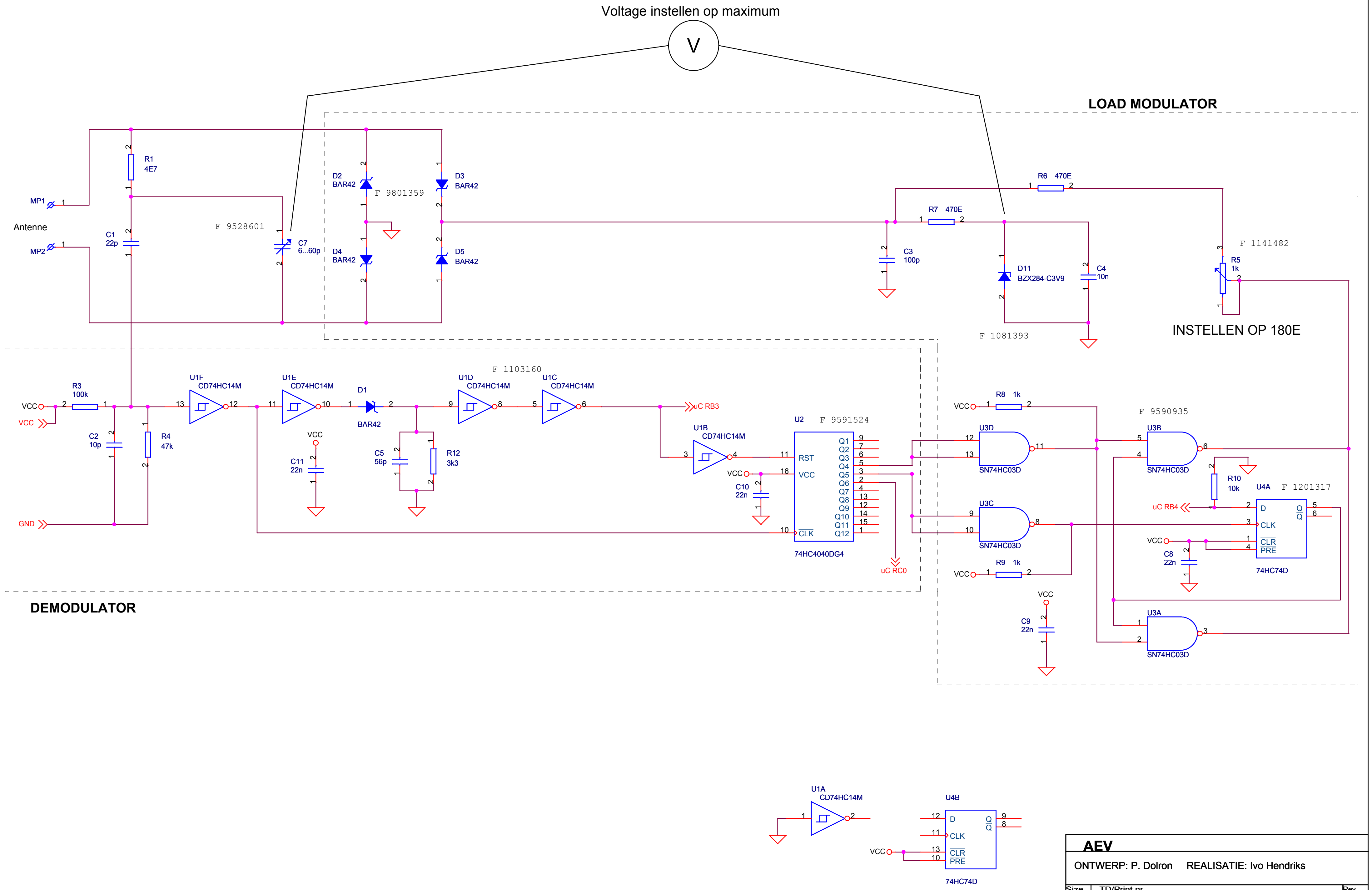


AEV

ONTWERP: P. Dolron REALISATIE: Ivo Hendriks

Size	TD/Print nr	Rev
A4	TD: 5650 TEKNUMMER: 5993 PRINTNUMMER: 060114A	.

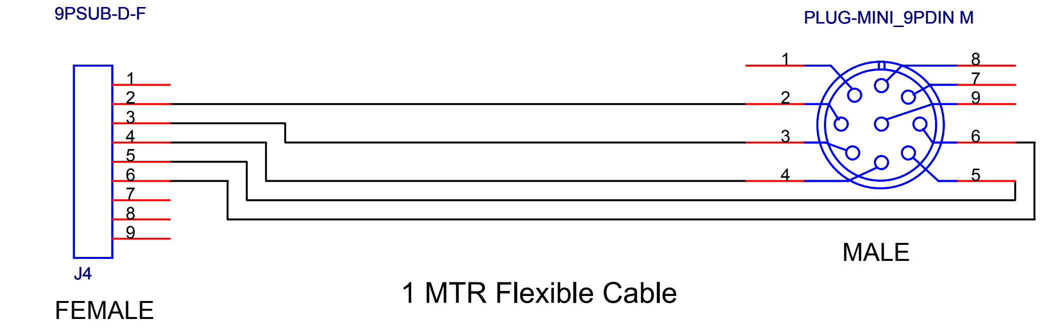
Date Tuesday, November 27, 2007 Dwg nr 3 - 4

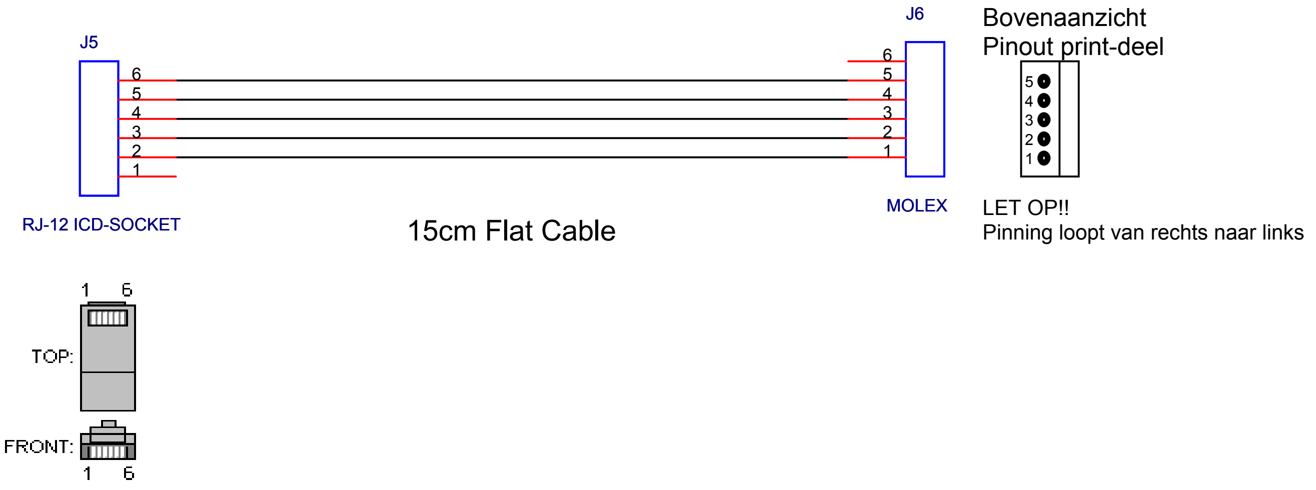


AEV

ONTWERP: P. Dolron REALISATIE: Ivo Hendriks

Size A3	TD/Print nr TD: 5650 TEKNUMMER: 5993 PRINTNUMMER: 060114A	Rev .
Date	Thursday, November 29, 2007	Dwg nr 4 - 4





AEV Cable		
ONTWERP: P. Dolron REALISATIE: Ivo Hendriks		
Size A4	TD/Print nr TD: 5650 TEKNUMMER: 5993 PRINTNUMMER: 060114A	Rev .
Date	Tuesday, December 18, 2007	Dwg nr 4 - 4

TD: 5650 TEKNUMMER: 5993 PRINTNUMMER: 060114A   Revision: .

Eavesdropping

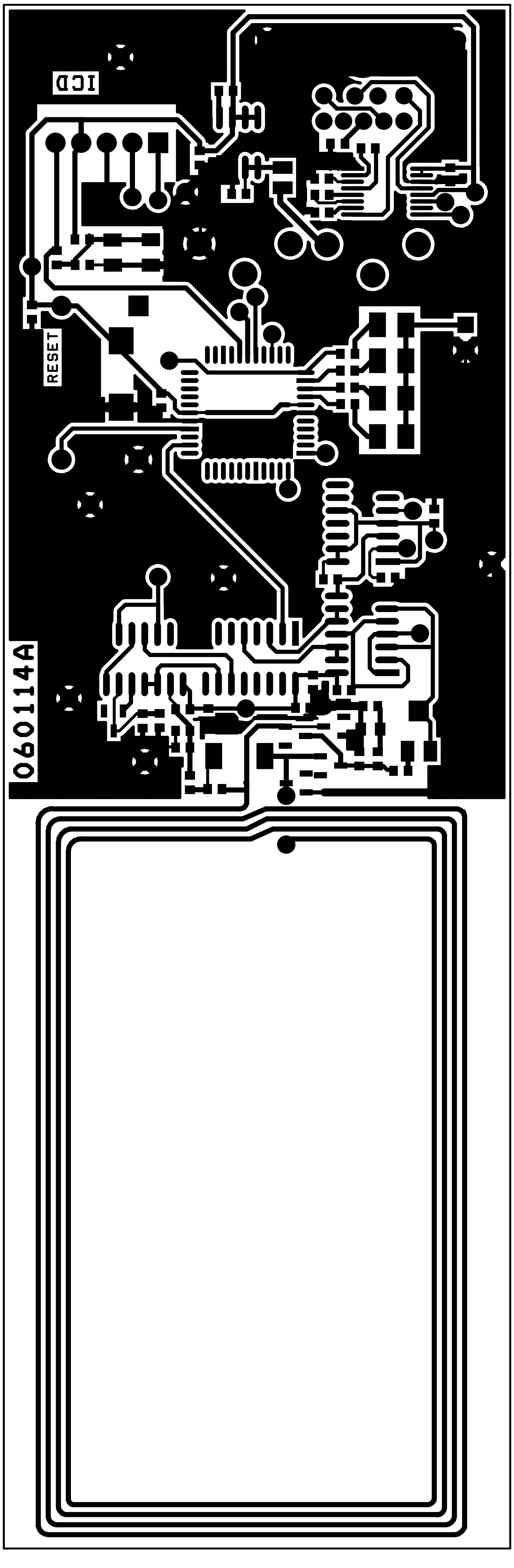
ONTWERP: P. Dolron   REALISATIE: Ivo Hendriks

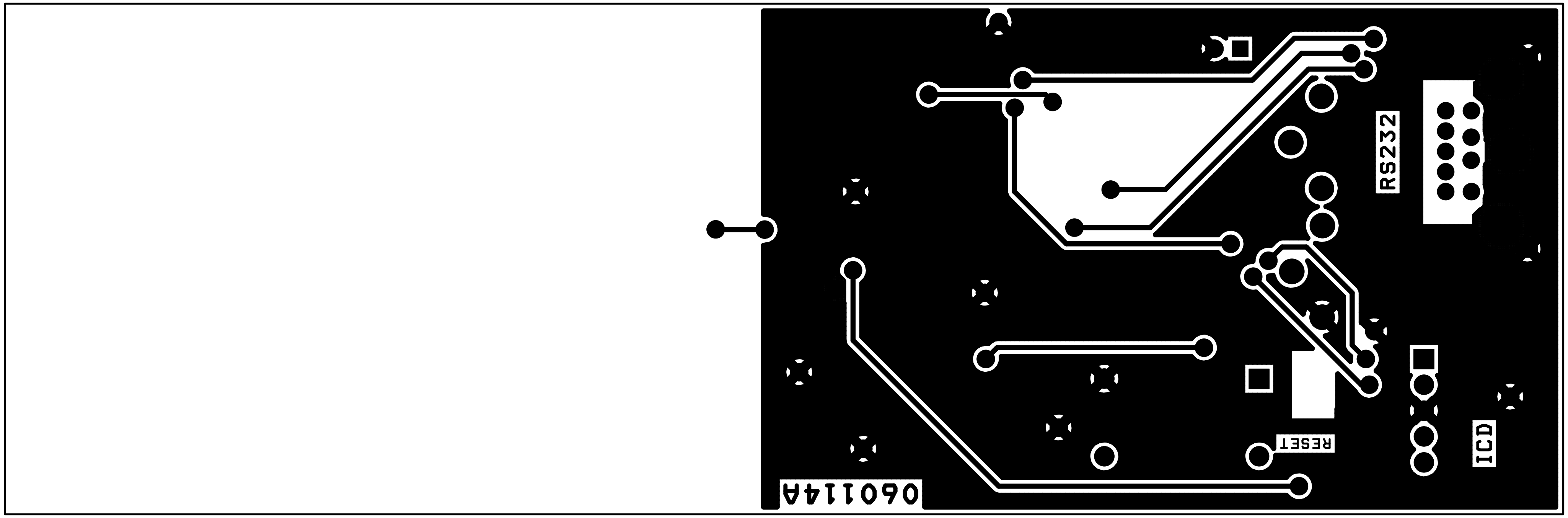
Bill Of Mat‹ Page1

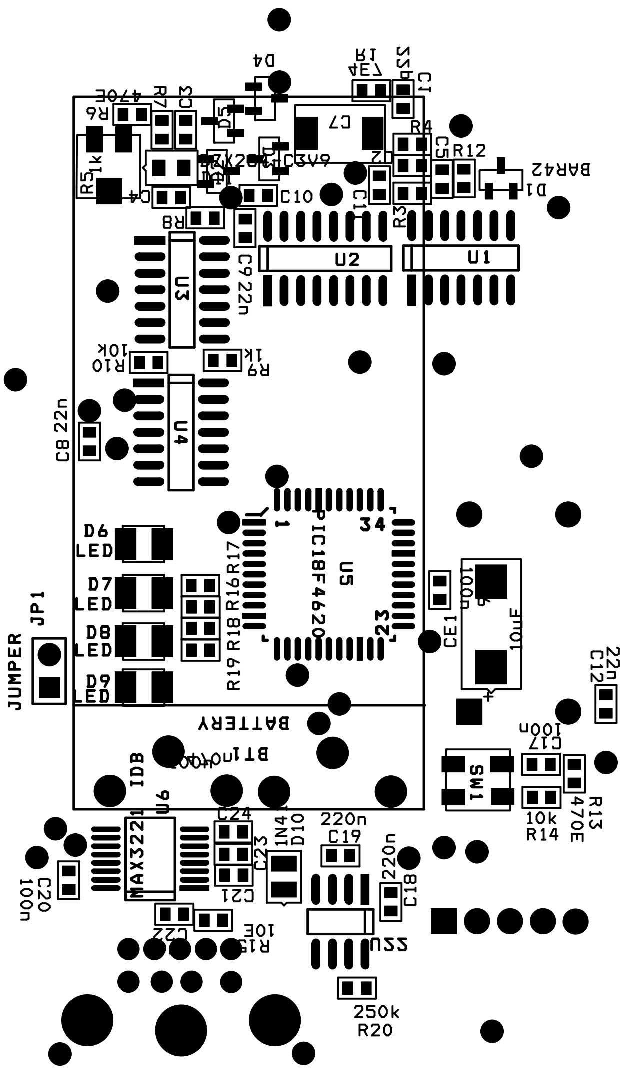
Quantity	Reference	Part		Bestelnummer Farnell:
1	BT1	BATTERY		723988 (clipjes)
1	CE1	10uF		
1	C1	22p		
1	C2	10p		
1	C3	100p		
1	C4	10n		
1	C5	56p		
1	C7	6...60p		9528601
5	C8	22n		
	C9	22n		
	C10	22n		
	C11	22n		
	C12	22n		
4	C16	100n		
	C17	100n		
	C20	100n		
	C24	100n		
2	C18	220n		
	C19	220n		
3	C21	470n		
	C22	470n		
	C23	470n		
5	D1	BAR42		9801359
	D2	BAR42		
	D3	BAR42		
	D4	BAR42		
	D5	BAR42		
4	D6	LED		1141111
	D7	LED		
	D8	LED		
	D9	LED		
1	D10	1N4148		
1	D11	BZX284-C3V9		1081393
1	JP1	JUMPER		
1	J2	ICD CONNECTOR		
1	J3	DE9S-FRS		152392
2	MP1	MEASURINGPIN	***DOOR SOLDEREN***	
	MP2	MEASURINGPIN	***DOOR SOLDEREN***	
1	R1	4E7		
1	R3	100k		
1	R4	47k		
1	RT5	1k		1141482
2	R8	1k		
	R9	1k		
7	R6	470E		
	R7	470E		
	R13	470E		
	R16	470E		
	R17	470E		
	R18	470E		
	R19	470E		
2	R10	10k		
	R14	10k		
1	R12	3k3		
1	R15	10E		
1	R20	250k		
1	SW1	PUSHBUTTON		1123624
1	U1	CD74HC14M		1103160
1	U2	74HC4040DG4		9591524
1	U3	SN74HC03D		9590935
1	U4	74HC74D		1201317
1	U5	PIC18F4620		1212705
1	U6	MAX3221 IDB		1053611
1	U21	TBD		9509771
1	U22	TPS7350QD		8461147

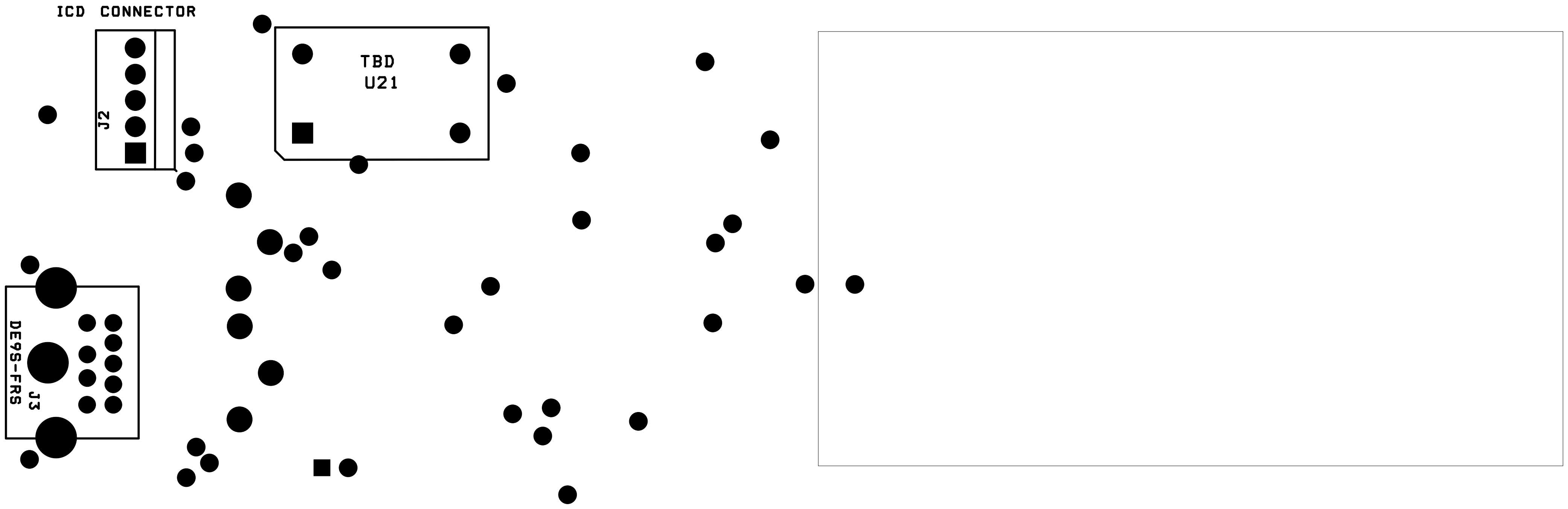
Montage materiaal:

Plug mini din 9p	152391
Sub-D 9way	1075335


ICD
RESET
060114A


RS232
ICD
RESET
060114A


D4
R1
C3A
C1
22p
R6
470E
R7
C3
D5
C7
R5
1k
Z7K20M
C3V9
R4
C5
R12
BAR42
D1
C4
D11
C10
C11
R3
R8
U2
U1
U3
C9 22n
R10
10K
R9
1K
C8 22n
U4
D6
LED
D7
LED
D8
LED
D9
LED
R17
R16
R18
R19
PIC18F4620
U5
1
34
23
100n
CE1
10uF
+
C12
22n
JP1
JUMPER
BATTERY
BT1
IDB
U6
MAX3221
C24
C23
C21
1N4
D10
220n
C19
220n
C18
C20
100n
C22
R15
10E
22u
250k
R20
SW1
C17
100n
10k
R14
R13
470E


ICD CONNECTOR
J2
TBD
U21
J3
DE9S-FRS