Reverse engineering the memory layout of the OV-Chipkaart

Ruben Muijrers
Supervisor: Wouter Teepe

Radboud University Nijmegen

Abstract. The OV-Chipkaart is meant to become the used-by-all standard for the Dutch public transport system. It makes use of the, by NXP developed, Mifare Classic. The Mifare Classic however has severe security issues [4]. Based on the work of two students from the University of Amsterdam I continued to reverse engineer the memory layout of the OV-Chipkaart.

1 Introduction

In 2011 the Dutch government and public transport companies want to replace the current system which is used to pay for the public transport. The new system will be using contactless chip cards (disposable and personal cards) and will be applicable in all public transports (right now the tickets for the trains differ from the tickets for the busses and subways). There is a lot of criticism on the new system, it violates the Dutch privacy laws by storing information on the card and logging every move that is made by the end-users. Besides the privacy issues there are also some heavy security issues. The actual brand of the cards is Mifare Classic. In April 2008 an attack was found that could reveal certain parts of the memory of Mifare tags (the cards) [3]. A few months later it was shown that the Mifare Classic tags [4] can be easily read and copied to another Radio Frequency IDentification(RFID) device. Since then it has become a popular research topic.

Despite the heavy criticism and the research results the public transport companies continued the project. This means that within a few years every citizen in the Netherlands is forced to use this system. If it's a public system everybody should be able to look at the security features and check for privacy issues. However the system is closed and there are no documents available explaining what is exactly stored on the tags. This paper focusses on reverse engineering the memory layout of the OV-Chipkaart and trying to find out what is stored on the tag and how.

2 Background

2.1 Previous Work

From the start the OV-chipkaart has been suffering from heavy criticism. In the past year the criticism was backed up by research results from several sources.

In July 2007 it was shown by two students from the University of Amsterdam (UvA) that because of a misconfiguration in the disposable tags it was possible to make trips without being charged [10].

Later that year in December two German researchers showed during the Chaos Communication Congress that had been able to reverse engineer parts of the Mifare Classic chip. This chip is used in all the non-disposable types of travel tags. At they also stated that the encryption module of the Mifare Classic chip was very weak [6].

Also in December 2007 a Dutch student from the Radboud University of Nijmegen demonstrated a way to clone and reset disposable tags using a device that was developed for this purpose. This way it was possible to travel free of charge (again) [11, 7–9, 2].

In April 2008 the Radboud University published a paper which described a way to read out all the data on Mifare Classic tags given the fact that at least one data block per sector is known. The first data block contains manufacturer data and thus gives read access to the first sector of each tag [3].

In October 2008 the Radboud University publishes a method to completely bypass the encryption on the Mifare Classic chips. Within minutes complete write and read access can be obtained for any tag that uses this chip. From this moment on only the fact that the memory layout is unknown prevents altering the OV-Chipkaart tags [4].

In 2009 two students from the UvA reverse engineer parts of the memory layout. They find out how and where the credits and the birth date of the user is stored. They also reverse engineer parts of the travel log which is also stored on the tags [1] (At the moment of writing this article the UvA students haven't published theirs yet).

Despite the impressive research results [5] the Dutch public transport companies still decided to continue the project as it is. The current deadline for putting it to use is at the end of 2009.

2.2 The OV-Chipkaart

The OV-Chipkaart comes in several flavours. First of all there is the disposable tag. It makes use of the Mifare Classic Ultralight chip. This type of tag can be used for two or three trips as well as travelling for a whole day.

Then there is the anonymous tag. This is a rechargeable card which can contain "travel products" such as discount products or day travel tickets. This tag makes use of the Mifare Classic chip.

Last but not least there is the personal OV-Chipkaart which can do the same and more as the anonymous card. Because it is linked to a person age discount is available for users of this card. It is also possible to link the card to a bank account and have it recharge automatically when the credits drop below a certain level.

For this research we are only interested in the anonymous and the personal types.

2.3 Memory Layout

The OV-Chipkaart system makes use of the 4K variant of the Mifare Classic family. This tag has 4 kilobytes of EEPROM memory divided in 256 blocks of 16 bytes. Read and write operation operate per block. The blocks are grouped into sectors the first 32 sectors have 4 blocks each the remaining 8 sectors have 16 blocks each. To access a block one needs to authenticate for the sector containing that block.

The last block of each sector is called the sector trailer, it contains the access keys and the access control bits. These access control bits determine what access is allowed on the sector and with which key (A or B). Usually the keys are not readable and return logical zeroes when accessed. In sector 0 to 31 each block for which $(n+1) \mod 4 = 0$ with n the block number, is a sector trailer. In sector 32 to 39 each block for which $(n+1) \mod 16 = 0$ with n the block number, is a sector trailer.

The UvA students grouped several sectors together and pointed some functionality to them [1]. The most important sectors for this research are 32 up to 39. Sector 32-34 contain which products are currently stored on the card. Sector 35-37 contain the travel log. Sector 38-39 contain the financial log as well as the current balance. More details on what the UvA students found will be shown in Section R̈esults and Analysis̈together with the results obtained during this research.

3 Methodology

3.1 The Hardware & Software

For this project we had Nokia 6361 RFID phone and a Tikitag reader (they are called Touchatag nowadays) which were a relief compared to the big laptop and Proxmark we used for previous researches. My supervisor Wouter Teepe developed an application for the Nokia 6131 which can make a memory dump when a RFID-tag is within range, provided that it has the keys necessary to do it in its database. This tool also allows for annotating the memory dumps and automatically adds a timestamp to them. This made analysis a lot easier.

I acquired the keys with some home-made software from our university that implements the fastest attack on the Mifare Classic tags known at the moment of writing this article (ADD REF). With this software and the Tikitag it is possible to extract all keys from the OV-Chipkaart tags within a couple of minutes (on a Pentium 4 2.2ghz Laptop). After these keys are loaded into the Nokia phone a memory dump takes several seconds to complete.

3.2 Making the Memory dumps

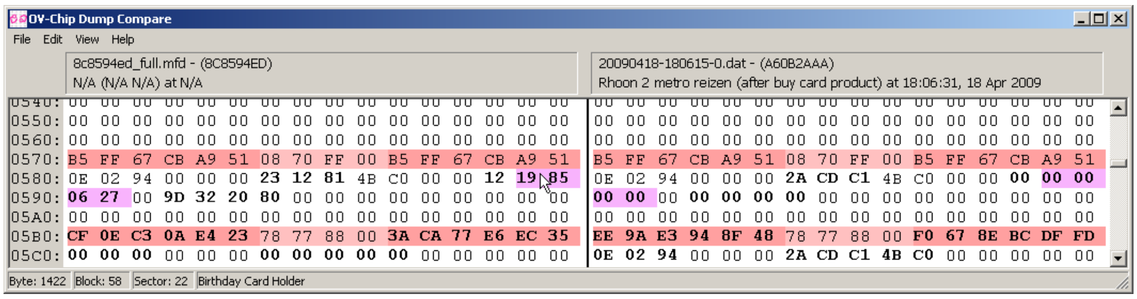
I made a trip to and through Rotterdam. On arrival in Rotterdam I bought a new anonymous OV-Chipkaart, after key extraction I made an initial memory dump. I travelled a bit around Rotterdam making new memory dumps after each action on the tag (check-in, check-out, recharge...). At one station I checked in and out of every gate on that station and at the end of the day I charged the tag with every product available on the machines: 2 trips travel ticket, 2 hour travel ticket, 1 day travel ticket with discount and 1 day travel ticket without discount. In addition I recharged it with 1 euro a couple of times. On the station where I checked in and out at every machine I also checked in and out with an second anonymous OV-Chipkaart (this one has been used before). This left me with 43 consecutive annotated memory dumps from the trip.

– 1 initial memory dumps of each tag I was carrying with me (5 total, not everyone was used during the rest of that day)
– 1 memory dumps after a print out for each tag (5 total)
– 11 memory dumps when traveling around
– 6 memory dumps for two tags each check in an out of every machine on station “Rhoon” (12 total)
– 8 memory dumps when charging the tag and storing travel products on it
– 1 additional memory dump after a print out at the end of the date for both used tags (2 total)

I also made some memory dumps of tags from some fellow students and my supervisor. This brought me to a total of 48 memory dumps.

4 Results and Analysis

4.1 Sector Groups



8c8594ed_full.mfd - (8C8594ED)
N/A (N/A N/A) at N/A

20090418-180615-0.dat - (A60B2AAA)
Rhoon 2 metro reizen (after buy card product) at 18:06:31, 18 Apr 2009

```
0540: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0550: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0560: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0570: B5 FF 67 CB A9 51 08 70 FF 00 B5 FF 67 CB A9 51 | B5 FF 67 CB A9 51 08 70 FF 00 B5 FF 67 CB A9 51
0580: 0E 02 94 00 00 00 23 12 81 4B C0 00 00 12 19 85 | 0E 02 94 00 00 00 2A CD C1 4B C0 00 00 00 00 00
0590: 06 27 00 9D 32 20 80 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
05A0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
05B0: CF 0E C3 0A E4 23 78 77 88 00 3A CA 77 E6 EC 35 | EE 9A E3 94 8F 48 78 77 88 00 F0 67 8E BC DF FD
05C0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | 0E 02 94 00 00 00 2A CD C1 4B C0 00 00 00 00 00
```

Byte: 1422 | Block: 58 | Sector: 22 | Birthday Card Holder

Fig. 1: Screenshot of the HEX comparer used for analysis

For easy analysis of the memory dumps I wrote a specialized HEX-Comparer. It highlights bytes with of which the semantics are known with some custom color. It displays differences between two memory dumps in bold. This gives a quick an easy overview of what is stored on the tag and where. Using this tool I compared each memory dump with it’s follow up memory dump. This way I could spot changes caused by one specific action eg. check-in. I als compared some memory dumps between tags to find out about tag specific information.

The students from the UvA grouped the sectors on the OV-Chipkaart together as is shown in Table 1 (some minor typos were fixed). Relative (to the sector) blockcounts are used in decimal notation.

Sectors	Blocks	Record Type
Sector 0 Block 00-03	Block 0-2	Card info. UID, manufacturer information etc.
	Block 3	Sector trailer
Sector 1-21	Block 0-2	Unknown. These can not be read.
	Block 3	Sector trailer
Sector 22	Block 0-2	Personal. Information about its owner.
	Block 3	Sector trailer
Sector 23-31	Block 0-2	Unknown
	Block 3	Sector trailer
Sector 32-34	Block 0-14	Products. Additional products such as season tickets, 3 blocks each.
	Block 15	Sector trailer
Sector 35-37	Block 0-13	Travel. All the transactions for travel movements (e.g. check-in), 2 blocks each.
	Block 14	Assumed to be unused
	Block 15	Sector trailer
Sector 38	Block 0-3	Unused?
	Block 4-5	Travel. Some recent travel transaction
	Block 6-13	Financial. Financial transactions such as recharging or purchase of a product.
	Block 14	Unused
	Block 15	Sector trailer
Sector 39	Block 0-8	Unknown
	Block 9-10	Balance
	Block 11-14	Unknown
	Block 15	Sector trailer

Table 1: Grouped sectors according to the UvA students

Per sector group the UvA students specified what they found including unknown data and recurring patterns on different tags of which no meaning is known. I used their tables and added my data to them. In the following subsections the tables at the end of each subsection contain a summary of what is known of the corresponding sectors. For each piece of information I gave the source and, if the source is UvA, whether I was able to verify it. The first column specifies the type of the data field, the meaning and formatting of those types as defined by the UvA students is shown in Table 2. I added my findings to the defination table and fixed some typos so it does differ from the table as it appears in the UvA report.

Still a lot of bits have no known meaning yet. For future research on this I also added these bits to the tables. For every sequence I mention wheter I have seen it differ per transaction, per tag or not at all.

Type	Description
UID	Unique identifier, 32 bits integer.
Date	14 bits integer, formatted as days after 1-1-1997.
Time	11 bits integer, formatted as minutes after midnight.
Credit/Fee	12 bits integer in Euro cents. Negative values are stored in two's complement. i.e. if msb is set, the value is interpreted as a bitwise not plus one.
Birthday	The 32 bits are set so that the date is human readable when viewed in hexadecimal. It is formatted as YYYYMMDD.
Type	7 bits integer. We have seen four values so far: 0 = Load, 1 = Check-in, 2 = Check-out and 6 = Change
Counter	Integer, probably 8 bits. It serves as an enumerator.
Duplicate	A single bit, when set, the record is already stored somewhere else on the card.
StationID	The ID of a station, its not clear how many bits the ID is, since it's always preceeded and followed by logical zeroes.
IssuerID	The ID of the issuer of the tag, this is a 4 bit number.

Table 2: The meaning and formatting of the data field types

4.2 Sector 0

I verified that the UID (which is printed in decimal form on the printouts) and the expiration date were stored here. With the help of my supervisor we were able to find the issuer ID. The issuer ID is the 8th digit of the number printed on the back of each ov-chip tag. It seems that the issuer ID is a number from 0 to 6 mapped the the issuer name (in alphabetic order) as can be seen in Table 3.

Issuer	Issuer ID
Arriva	0
Connexxion	1, *verified*
GVB	2, *verified*
HTM	3
NS	4, *verified*
RET	5, *verified*
Veolia	6

Table 3: The OV-Chipkaart Issuers and their corresponding IDs

Type, Bits	Description	Source
UID, 0-31	The card's identifier, used to distinguish different physical cards.	UvA, *verified*
UID-CRC, 32-39	The CRC byte of the UID.	R. Verdult, *verified*
Unknown, 40-79	Always same pattern, 10011000 00000010 00000000 01100100 10001110	-
Unknown, 80-93	Differs per tag	-
Date, 94-107	Expiration date of the card.	UvA, *verified*
Unknown, 108-127	Differs per tag	-
Unknown, 128-223	Always same pattern, 10000100 00000000 00000000 00000000 00000110 00000011 10100000 00000000 00010011 10101110 11100100 00000001	-
Unknown, 224-235	Differs per tag	-
Unknown, 236-271	Always same pattern, 1000 00001110 10000000 10000000 11101000	-
Issuer ID, 272-275	The issuer of the tag	-
Unknown, 276-279	Differs per tag	-
Empty, 280-383	Always logical zeroes	-

4.3 Personal Sector

I didn't find anything new here. The UvA students reported some patterns here that occurred on all their tags. I saw the same patterns on my memory dumps so I can't add anything to their conclusions. Except for the fact that sector 23 (the next sector) sometimes contains a copy of the personal sector.

Type, Bits	Description	Source
Unknown, 0-22	The same pattern (00001110 00000010 100101) for every type of card.	UvA, *verified*
Empty, 23-49	Always filled with logical zeros.	UvA, *verified*
Unknown, 50-82	Unclear, the last 15 bits are always the same pattern (000010 10010111 1).	UvA, *verified*
1 Empty, 83-106	Always filled with logical zeros.	UvA, *verified*
Unknown, 107-111	Always 10010 in personal cards, empty for anonymous cards.	UvA, *verified*
Birthday, 112-143	The birthday of the card holder, empty for anonymous cards.	UvA, *verified*
Unknown, 144-151	Unclear, differs per tag.	UvA
Unknown, 152-175	Always the same pattern (10011101 00110010 00100000).	UvA, *verified only for personal cards*
Card type, 176	Bit is probably used to distinguish discount cards (0) from trajectory cards (1).	UvA
Empty, 177-383	Always empty.	UvA, *verified*

4.4 Products

The UvA students report that the product info is stored in sector 32 to 34. From the memory dumps I collected in Rotterdam it appears that sector 27 to 31 are also involved in product storage. Let sector 27-31 be called the Lower Product Information (LPI) and sector 32 to 34 called the Higher Product Information (HPI).

When a product is stored in the HPI, a hexadecimal 12 is also stored in the second byte of one of the blocks in the LPI. So instead of 3 blocks a product record uses 4 blocks. From the dumps a relation appeared as is shown in Table 4. Which leads to

Block in LPI	Blocks in HPI
0	0-2
1	3-5
2	6-8
13	39-41

Table 4: Relation between blocks in sector 27-31 (LPI) and blocks in sector 32-34 (HPI) in relative decimal values (not including trailer blocks)

When we try to generalise the relation in Table 4. We get $HPI = 3 * LPI$ not including the sector trailers. Note that the block numbers are relative to the first sector of the LPI and the first sector of the HPI. Given the previous relation and the memory available on the cards it is possible to store at most 15 travel products on each card.

This relation was also tested against the personal student discount tags (OV jaarkaart) of me and several of my fellow students. On the student discount tags are two products stored in blocks LPI:0,HPI:0-2 and in blocks LPI:1,HPI:3-5.

The travel products can be grouped into two categories: Activated-On-First-Use (AOFU) and Activated-On-Date (AOD). An example of an AOFU product is a day-travel ticket and an example of an AOD product is a student discount product (this is activates on a predefined date). When an AOFU product is activated some information is stored in bit 8 to bit 30 of the corresponding block in the LPI. Bit 17 to 30 are the activation date of the product.

In the HPI bit 41 to bit 52 (11 bit, but may start at lower bit and be longer) for each entry specify a product ID. This product ID is sometimes equal to what is printed on the print-out but not always. Values I found so far are shown in Table 5. The product description is the name as it appears on the print out. For the latest entry in the table there was no print out available so the Product Number and the Description as it appears on the print out are not available.

Prod. ID	Prod. Nr	Product Description	Block Nr in LPI	First Bit Valid-Date
1995	1991	RET 2 Metroreizen vol tarief 2009	7C	93
1502	1502	RET 2 uur reizen 2009	6C	93
1968	1961	RET 1 dag Reductie 2009	6D	93
1967	1961	RET 1 dag Vol tarief 2009	6E	93
177	177	Studentenkaart korting week	6D	103 and 134
175	175	Studenten ov chipkaart week	6C	93
178	178	Studentenkaart korting weekend	6D	103 and 134
176	176	Studenten ov chipkaart weekend	6C	93
190	190	Tijdelijke Studentenkaart korting week	6C	103 and 134
188	188	Tijdelijke Studenten ov chipkaart week	7C	93
25	N/A	N/A (40% discount for all NS trips)	6C	93

Table 5: Product ID, Product Number and Product Name

The product ID is followed by a few bytes of logical zeroes. The UvA student write that bit 95 to 109 represent a 14 bit date (days since Jan 1st 1997). However if I assume the same format then this conflicts with the “Valid Since” date I found on the cards. In almost all products the “Valid Since” date starts at bit 93 (assuming 14 bits long). With products with product ID 177, 178 and 190 the “Valid Since” starts at bit 103. For these two products this date is repeated starting on bit 134. After these dates more bytes follow but I was unable to find any useful patterns.

From Table 5 it is clear that products are not randomly stored in the memory. It appears as if each product(type) has its own reserved space. What the exact relation is here, is not clear.

Type, Bits	Description	Source
Empty, 0-7	Always zero.	-
Unknown, 8-16	zeroes when not used, 000100100 when a product is stored in corresponding HPI, 001100001 when a product is activated.	-
Date, 17-30	Date on which the product is activated.	-
Empty, 31-127	Filled with zeroes.	-

Table 6: Records in the Lower Product Information

Type, Bits	Description	Source
Unknown, 0-6	Always (0000101) or (0000111).	UvA, *verified*
Unknown, 7-19	On trajectory products the pattern (0 00000010 111) is found, on some discount products this pattern is (0 00000000 111).	UvA, *verified*
Empty, 20-31	Always zeroes.	-
IssuerID, 32-35	Seems to be the ID of the issuer of the product, although this has not been tested on many cards.	-, *conflicts with what UvA found*
Unknown, 34	This bit is only set to 1 in trajectory products and discount products, it is set to 0 in bicycle products.	UvA
ID, 41-52	Product ID, specifies which type of product is stored here	-
Empty, 53-80	Always filled with zeroes	-
Date, 93-106	Date since when a product is valid and ready for use (for most products)	-, *conflicts with what UvA found*
Date, bit 95-109	The date on which the bicycle product is valid.	UvA.
Date, 103-116	Date since when a product is valid and ready for use (for products: 177, 178 and 190)	-
Date, bit 103-116	The first day the product is valid, only found in trajectory products.	UvA
Date, bit 117-130	The last day the product is valid, only found in trajectory products.	UvA
Date, 134-147	Date since when a product is valid and ready for use (for products: 177, 178 and 190)	-
Empty, 200-383	Always filled with zeroes	-

Table 7: Records in the Higher Product Information, All bits not listed here differ per product and have no known meaning.

4.5 Travel

In the travel sector I first tried to verify the results of the UvA students. The travel sectors behave exactly as they describe. Each record uses 2 blocks. On check out block 200 (absolute value here) and 201 are always rewritten. However the UvA students report that there is only 1 bit different in the second record that is written during check out. But from the memory dumps from Rotterdam it appears that more bits are different. The meaning of the differing bits is not clear to me. They also report that the time format is in seconds after midnight. That is incorrect (probably typo) the time format is an 11 bit integer counting the minutes after midnight rounded to the nearest whole minute. So 16:57:29 is $60 * 16 + 57 = 1017$.

The GVB tags store the type of the transaction in a 7 bit integer starting at bit 53. The RET tags store the transaction type starting at bit 77 it has the same format as the GVB tags (meaning 0 = Load, 1 = Check In, 2 = Check Out, 6 = Change).

Bit 117 to 124 (assuming an 8 bit integer) increases each time you pass a gate. However on the brand new tag when entering the first gate these bits were set to 5 and then counted upwards. For non-travel transactions the bits are set to totally different values the meaning of those values is not clear.

The GVB stored the station ID in bit 104 to 127. The UvA students made a tool to parse this, after that the tool failed on the RET logs it was proposed that the RET might use gate IDs instead of station IDs. However from my set of memory dumps it appears that the RET does use station IDs but that they are stored in bit 125 to 141 (assuming a 16 bit length).

The UvA describe describe that each transaction has a fee. The RET charges 4 euro when checking in and gives you back the money when you check (minus the price of the trip). In the RET tags the fee is stored in bit 168 to 181 (assuming a 12 bit length).

Type, Bits	Description	Source
Unknown, 0-6	Always one of two patterns: (0010100) or (0010000)	-
Duplicate, 7	When this bit is set, another copy of the record is available elsewhere in memory. We noticed that when the original record is removed (overwritten) this bit is reset for the copy.	UvA, *verified*
Empty, 8-15	Always filled with zeroes.	-
Unknown, 16-27	Differs per transaction but slightly, same patterns appear on multiple transactions	-
Date, 28-41	The date the transaction took place.	UvA, *verified*
Time, 42-52	The time the transaction took place, it is rounded to whole minutes.	UvA, *verified*
Type, 53-69	The type of transaction such as check-in or check-out. GVB only.	UvA
Type, 77-83	The type of transaction such as check-in or check-out. RET only.	-
Counter, 92-99	Serial number of the transaction, GVB only.	UvA
StationID, 104-127	These bits most definitely indicate the location of the transaction. In Amsterdam any action on the same station resulted in the same value. In Rotterdam there seemed to be difference between different type of actions and maybe the actual gate the transaction took place.	UvA
Counter, 117-124	Serial number of the transaction, RET only.	-
StationID, 125-141	Station ID for RET	-
Fee, 144-155	Every record has an associated fee. In Amsterdam only check-outs have an explicit price whereas Rotterdam also include the initial fee of 4 Euro.	UvA
Fee, 168-181	Fee of the trip for RET.	-

Table 8: All bits not listed here differ per transaction and have no known meaning.

4.6 Financial and Balance

I was not able to spend much time on this part of the memory. The balance system is as the UvA described. The financial log also behaves the same as the UvA students described up to the date and time after that it starts to differ. These parts of the memory are largely unexplored. With every transaction lots of bits change. Due to time constraints I was not able to document the exact behaviour of those bits.

Type, Bits	Description	Source
Duplicate, 7	When this bit is set, another copy of the record is available elsewhere in memory. We noticed that when the original record is removed (overwritten) this bit is reset for the copy.	UvA, *verified*
Date, 28-41	The date the transaction took place.	UvA
Time, 42-52	The time the transaction took place, it is rounded to whole minutes.	UvA
Type, 53-69	The type of transaction such as check-in or check-out. Only for the GVB.	UvA
Counter, 92-99	Serial number of the transaction, GVB only.	UvA
Fee, 142-153	Price of the purchase.	UvA

Table 9: Records in the financial log

Type, Bits	Description	Source
Counter1, Block9, 12-20	Increased after an action.	UvA, *verified*
Credit1, Block9, 82-94	Current/previous balance.	UvA, *verified*
Counter2, Block10, 12-20	Increased after an action.	UvA, *verified*
Credit2, Block10, 82-94	Current/previous balance.	UvA, *verified*
Unknown, Block 12-14	Repeated hexadecimal pattern (0123456789AB). However sometimes some bits are overwritten and the pattern is broken.	UvA, *verified*

Table 10: Records in the balance sector

5 Conclusions

My research continued that of some student from the University of Amsterdam [1]. I further reverse engineered the OV-Chipkaart. I was able to add some more meaning to the bytes stored on the tags. I also pointed out some wrong assumptions of the UvA students.

When looking back at the whole system I have to agree with the UvA students that the system is far from coherent. Even on low level almost every field has a different interpretation. Different products have most likely different data associated with them (or at least it is stored in a different manner). Most fields are not byte aligned, this of course makes it a little bit harder to read and understand the memory dumps but it's at most annoying. Most likely this is caused by the middleware that manages the data storage.

It appears that for (at least) the travel records the first few bytes, up to the time and date, are the same for every company. After that each company may have been able to write to the memory what they wanted as long as it fitted in the blocks. It is possible to make the public transport work with just that since you have to scan the tag to switch transport company (get out a bus, or check out at a station). This way the transport companies only need to know where a user enters their system and doesn't need knowledge of the places the user has been to using a different company.

Another weird thing is that the fact that they use (at most) 14 bits for there representation for dates seems a little small. It would mean that the dates would overflow on November the 9th 2041. This seems far away but so did the millenium bug at the time...

References

1. Joeri Blokhuis, Michiel Timmers, Yuri Schaeffer, and Wouter S. van Dongen. Memory layout of the anonymous and personal OV-Chipkaart, 2008.
2. Gerhard de Koning Gans. Analysis of the MIFARE Classic used in the OV-chipkaart project. Master's thesis, Radboud University Nijmegen, 2008.
3. Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A practical attack on the MIFARE Classic. In *8th Smart Card Research and Advanced Applications Conference (CARDIS 2008)*, volume 5189 of *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2008.
4. Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In *13th European Symposium on Research in Computer Security (ESORICS 2008)*, volume 5283 of *Lecture Notes in Computer Science*, pages 97–114. Springer-Verlag, 2008.
5. Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly pickpocketing a MIFARE Classic card. In *30th IEEE Symposium on Security and Privacy (S&P 2009)*, pages 3–15. IEEE Computer Society, 2009.
6. Karsten Nohl and Henryk Plotz. Mifare, little security, despite obscurity, 2007.
7. Roel Verdult. Proof of concept, cloning the OV-chip card. Technical report, Radboud University Nijmegen, January 2008.
8. Roel Verdult. Security analysis of RFID tags. Master's thesis, Radboud University Nijmegen, July 2008.
9. Ronny Wichers Schreur, Peter van Rossum, Flavio Garcia, Wouter Teepe, Jaap-Henk Hoepman, Bart Jacobs, Gerhard de Koning Gans, Roel Verdult, Ruben Muijrers, Ravindra Kali, and Vinesh Kali. Security flaw in MIFARE Classic. *Press release, Digital Security group, Radboud University Nijmegen, The Netherlands*, March 2008.
10. www.nu.nl. Gratis reizen door fout in ov-chipsysteem, 2007. Available from: http://www.nu.nl/internet/1137109/gratis-reizen-door-fout-in-ov-chipsysteem.html.
11. www.nu.nl. Wegwerpchipkaart gekraakt, 2008. Available from: http://www.nu.nl/internet/1391343/wegwerpchipkaart-gekraakt.html.