UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Robin Mürk

Estonian ID card's personal data file emulator

Bachelor's Thesis (9 ECTS)

Supervisor:
Arnis Paršovs, PhD

Tartu 2025

Estonian ID card's personal data file emulator

Abstract:

One of the Estonian ID card's common uses is to use it as a store's client card. Currently there exists no emulator of ID cards made since 2018, that could emulate the client card's authentication process. This thesis sets out to build such a Java Card based emulator, that emulates the ID card filesystem's personal data files, so the authentication to store terminals could work. The thesis gives an overview of the technical specifications of the current ID card and describes in detail the communication that happens between the ID card and the terminal. Later the ID cards filesystem and personal data files are explained, followed by the implementation of the ID card emulator that could authenticate to store terminals. The developed emulator was tested on 16 store terminals, on both contact and contactless interfaces. The emulator was able to successfully authenticate to 10 terminals, but none of the terminals were able to authenticate over the contactless interface.

Keywords: Java Card, applet, APDU, Estonian ID card, emulator

CERCS: P170 Computer science, numerical analysis, systems, control

Eesti ID kaardi isikuandmete faili emulaator

Lühikokkuvõte:

Üks ID kaardi levinumaid kasutusviise on selle kasutamine poe kliendikaardina. Alates 2018. aastast välja antud ID-kaartidele puudub aga avalikult kättesaadav emulaator, mis võimaldaks kliendikaardi autentimisprotsessi emuleerida. Käesoleva töö eesmärgiks on arendada *Java Card*'i põhine emulaator, mis emuleerib ID-kaardi failisüsteemi isikuandmete faile, et võimaldada autentimist poe terminalides. Töö raames antakse ülevaade ID-kaardi tehnilistest spetsifikatsioonist ning kirjeldatakse täpselt kaardi ja terminali vahelist suhtlust. Hiljem antakse ülevaade ID kaari falisüsteemist ja isikuandmete failist, millele järgneb emulaatori teostus. Emulaatorit testiti 16 poe terminali kontakt kui ka kontaktivaba liidese peal. Autentimine toimis 10 terminalis, kuid kontaktivaba liidese kaudu ei õnnestunud autentimine üheski.

Võtmesõnad: Java Card, rakend, APDU, Eesti ID kaart, emulaator

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1. Introduction 5

2. The Estonian ID card 7

2.1 Current platform and technical specifications 7

2.1.1 Java Card and GlobalPlatform standards 7

2.1.2 IDEMIA platform 7

2.2 Communication 8

2.2.1 Application Protocol Data Unit 8

2.2.2 T=1 and T=0 protocol 9

2.2.3 Answer To Reset and Answer To Select 11

2.3 Filesystem 12

2.3.1 Reading the personal data EF 13

3. Implementation 15

3.1 Transient and persistent memory 15

3.2 Applet layout 16

3.3 Filesystem 19

3.3.1 Handling file selection 20

3.3.2 Handling EF reading 21

3.4 Logging 22

3.5 ATR and ATS 23

3.6 Loading of the emulator applet to the card 24

4. Data gathering and analysis 26

4.1 Methodology for data gathering 26

4.2 Results 28

4.2.1 Ingenico terminals 29

4.2.2 Worldline YOMANI terminals 29

4.2.3 Verifone P400 30

4.2.4 Contactless interface 30

4.2.5 Print In City printer 31

5. Conclusion 32

References 33

Appendices 36

A. Some of the terminals tested in this study 36

B. APDU log, Ingenico unknown model (Alexela gas station, contact interface) 38

C. APDU log, Ingenico Lane/3000, Desk/1600, iPP320, Move/3500 and integrated unknown terminal (Vapiano, Apollo, Olerex, Rahva Raamat, Terminal gas station, Brain Games, Apotheka, contact interface)..... 39

D. APDU log, Worldline YOMIANI terminal (Prisma, contact interface)..... 40

E. APDU log, Worldline YOMIANI terminal (K-rauta, contact interface)..... 41

F. APDU log, Worldline YOMIANI terminal (Bauhof, Klick), contact interface..... 42

G. APDU log, Verifone P400 (Tokumaru, contact interface) 43

H. APDU log, Ingenico iUC150B, IPP320 and Worldline YOMIANI terminals (Olerex, Alexela, terminal, Prisma, contactless interface)..... 44

License..... 45

1. Introduction

The Estonian identity card (ID card) has many use cases. It is primarily used to digitally sign documents and vote in governmental elections. It is one of the only forms of identification that can be used to physically identify a person, as the card has a picture of the card owner on it. Additionally the card functions as a way to digitally identify a person to an online service, such as a bank or an official government site [1].

Many stores have adapted the ID card to be used as a client card. This works by using the store's point-of-sale payment terminals (hereafter referred to as terminal) to read the ID card owner's personal info from the card, to identify the person. The personal data stored on the card is the same data that is also visible on the physical card. This is possible as these personal data files are not cryptographically protected [2]. Theoretically anyone could read the personal data files from anyone's ID card.

Since 2018, the new ID cards, made by IDEMIA (hereafter referred to as IDEMIA ID card), also support a contactless interface. Communication over the contactless interface has to be encrypted using the PACE protocol using the six-digit card access number (CAN) written on the physical card. The CAN is not included in the personal data files, making the use of the contactless interface impractical, as the terminal would have to know the CAN in advance. Sander-Karl Kivivare has done a deep analysis of the PACE protocol of the ID card and demonstrated how the protocol works [3]. As of writing this thesis the Estonian Information System authority has allowed the signing of documents over the contactless interface. This requires the use of the RIA DigiDoc phone application, where a person needs to manually insert the CAN into the app [4]. This is currently the only known usage of the contactless interface of the ID card.

The aim of this thesis is to build an IDEMIA ID card emulator that can successfully authenticate to terminals used by stores in practice. Secondary goals are to test if the terminals support authentication over the contactless interface. The terminal's firmware is responsible for communicating with the ID card and reading the personal file's contents. The hypothesis is that terminals, that have both contact and contactless interface, execute the same connection sequence, regardless of the interface. This emulator could be used by developers who work with these terminals, as a way to test the authentication process.

To achieve this, an IDEMIA ID card emulator that emulates a part of the filesytem where the personal data is stored (hereafter referred to as emulator), is built. For this, an overview of the

IDEMIA made ID card is given. This is followed by a detailed description of the communication that happens between the ID card and the terminal, followed by a breakdown of the IDEMIA ID card's filesystem, regarding the personal data files. Next, a description of the implementation of the emulator is given, where the emulator applet and a logging system are built. Finally, the emulator is tested on terminals used by stores in practice and the results are analysed.

Similar work has been conducted by the supervisor of this thesis, Arnis Paršovs, and his associate, Danielle Morgan: "Using the Estonian Electronic Identity Card for Authentication to a Machine" [5]. In their thesis they also built an Estonian ID card emulator to demonstrate the security risks that come with the ability to do so and to also see what the terminals in practice actually request from a person's ID card. They could not test any contactless functionality, as then in-use ID cards did not have that functionality. This thesis builds upon their work, by also testing terminals with the contactless interface to see if such functionality would be supported by the terminals in use today.

2. The Estonian ID card

This section covers the technical specifications and standards of the IDEMIA ID card used to implement the emulator. This is followed by a detailed description of the the communication that happens between the terminal and the ID card. At the end of the section an overview of the IDEMIA ID card's filesystem, regarding the personal data files, is provided along with a breakdown of how the terminal accesses and reads these files.

2.1 Current platform and technical specifications

2.1.1 Java Card and GlobalPlatform standards

The ID card's platform relies on two main standards: The Java Card specification and the GlobalPlatform standard. The Java Card specification defines the lifecycle of the Java virtual machine (VM) and the application programming interface (API) for the Java Card platform applet development. Java Card also refers to a smart card that has the Java VM on it's chip (hereafter also referred to as Java Card). It defines a subset of the Java language, with notable differences being that only limited data types are allowed: short, byte, char and custom objects. Multithreading and most of the libraries available in Java are disabled [6]. The GlobalPlatform standard defines how the management of Java Cards is conducted. It defines the GlobalPlatform Java Card API that is used to install, manage and delete Java Card applets on the card [7]. The emulator applet developed in Section 3 of this thesis uses the Java Card 2.2.2 specification to build a functioning applet, that could be loaded onto any Java Card compliant card. The card only has to support Java Card version 2.2.2.

2.1.2 IDEMIA platform

The ID card's filesystem, regarding the personal data files, emulated in this study is based on the IDEMIA manufactured ID cards introduced in 2018 (Figure 1). These cards use the ID-One Cosmo v8.1 card platform that support the Java Card 3.0.4 specification and are GlobalPlatform 2.2.1 compliant [8]. They are the first cards to implement a contactless interface that, until recently, did not find any real usage (Section 1). The card uses the Identification Authentication Signature - European Citizen Card (IAS-ECC) specification for its applet. This with the Cosmo v8.1 Java Card specification has been given a Common Criteria (CC) security certification level of EAL5+ [8]. There was an upgraded IDEMIA ID card introduced in 2021 but that upgrade did not affect the functionality and communication interface of the ID card [9]. This thesis focuses on the previous version of the card, introduced in 2018.

Figure 1. IDEMIA produced Estonian ID cards used since 2018 [10].

2.2 Communication

This section covers how the terminal and card communicate with each other, how data is transmitted and what transmission protocols are used.

2.2.1 Application Protocol Data Unit

This subsecton is based on a blog post, that gives an overview of the ISO 7816-4 section 5 [11]. Communication between a terminal and a Java Card uses a request - response cycle, where data is sent through the Application Protocol Data Unit (APDU). Furthermore they are separated to command APDUs (C-APDU) and response APDUs (R-APDU). The C-APDU consists of a mandatory 4-5 byte header and then 0-255 bytes of data. The header bytes define the type of command being sent and if data is expected or data is given. The structure (Figure 2) , types of commands and use of the APDUs is defined in ISO 7816-4. The CLA byte is the instruction class byte, that defines the class of command being sent. In most commands this is kept as `0x00`. The INS byte defines the specific command for example `SELECT FILE` where the byte is `0xA4` or

`READ BINARY` where the byte is `0xB0`. P_1 and P_2 are parameter bytes that define extra options for the command. P_3 is the length expected (L_e) byte, giving info on how many bytes are expected as a response. The fifth byte could also be a L_c byte, that holds the number of C_{data} bytes being sent. The fifth byte depends on which case the C-APDU is. After the C_{data} bytes, there could be an additional L_e byte, to indicate how many bytes are expected as a response.

$$[\texttt{CLA}]\,[\texttt{INS}]\,[\texttt{P1}]\,[\texttt{P2}]\,[L_c]\,[C_{data}]\,\ldots\,[L_e]$$

Figure 2. C-APDU structure [12].

There are 4 cases of C-APDUs possible [11]:

1. A 4 byte C-APDU where no data is given and no data is expected as response;
2. a 5 byte C-APDU where the fifth byte denotes the length of the expected data (L_e byte);
3. a 5 + n * C_{data} byte C-APDU where the fifth byte denotes the length of the C_{data} bytes contained in the C-APDU;
4. a 5 + n * C_{data} + L_e byte C-APDU where data is sent but also Le amount of data is expected as a response.

The R-APDU consists of the data being returned and a mandatory 2 byte status wart (SW) after the optional data. The SW is used for giving information about the C-APDU. Some common SW are [13]:

1. `0x9000` - command was successful (from hereafter referred to as OK SW);
2. `0x6CXX` - the C-APDU must be sent again with the SW2 byte as the L_e byte;
3. `0x61XX` - SW2 amount of bytes that can still be read from the card, a `GET RESPONSE` C-APDU is expected with the L_e byte as SW2;
4. `0x6A82` - file was not found.

2.2.2 T=1 and T=0 protocol

This subsection relies on a blog post written by Dr. David B. Everett [14]. The APDUs are transmitted primarily by two transmission protocols: T=0 and T=1 protocol. There is also a T=CL protocol but this is just the T=1 protocol used over the contactless interface. The T=0 protocol is a byte oriented, half-duplex protocol where only the APDU is sent. It was first defined

in ISO 7816-3. In this protocol the terminal[1] and the card must know who is sending data as data can only move in one way (Figure 3).

The protocol works by first sending the C-APDU header bytes to the card. Then the card responds with a procedure byte that can be of three options. Firstly the procedure byte can be an ACK byte, usually in the form of the C-APDU header INS byte. This indicates to the sender that the command has been received and can now be executed. Depending on if the sender wants to send data or is ready to receive it from the card, the ACK byte must be received first by the sender. Secondly the procedure byte can be a `0x60` byte, indicating that the card needs more time to process the sent header. When the card is ready, it will send an ACK to signal its readiness. Thirdly, the procedure byte can be the first byte of the SW. After this the second SW byte is sent as well. This usually happens if no data is exchanged between the sender and the card [14].

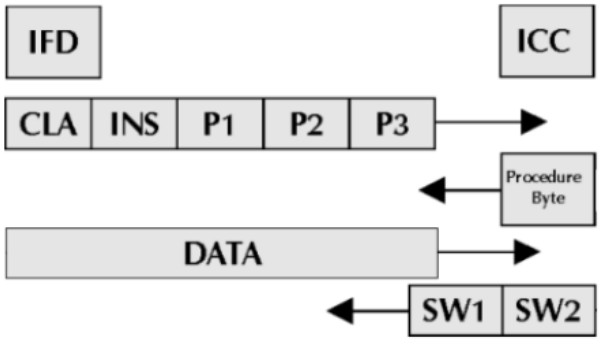

Figure 3. Data being sent to the card with T=0 protocol [12].

The T=1 is a full-duplex, block oriented protocol. It works by encapsulating the APDU in an envelope-like structure. This allows for communication to work both ways in a single round of exchange. In it, a three byte header is added to the C-APDU and a one byte cyclic redundancy check (CRC) byte is added to the end of the C-APDU (Figure 4). The first byte is the node address byte (NAD). It signifies the source and destination address of the block. The second byte is the protocol control byte (PCB). This codes the type of data being sent, like the C-APDU. The length byte (LEN) contains the entire length of the C-APDU. The CRC contains a checksum of the entire block and is meant for error detection [15].

[1]In Figure 3, ICC stands for integrated circuit card, in this case the ID card. IFD stands for interface device, in this case the terminal.

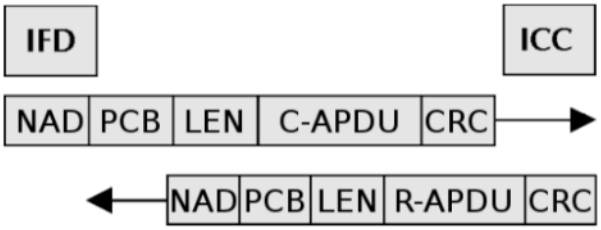

Figure 4. APDU communication over T=1 protocol [12].

2.2.3 Answer To Reset and Answer To Select

Each Java Card that implements a contact interface, needs to send an Answer To Reset (ATR) to the terminal. It contains electronic communication info, what protocols are supported by the card and metadata. It is sent when the card's ATR consists of a mandatory two byte header, TS byte and T0 byte, where extra header bytes are optional depending on the T0 bytes value. After the header follow the optional historic bytes that act as metadata for the card. The last byte is an optional check character (TCK), that checks the integrity of the ATR. The T0 bytes four least significant bits encode the size of the historic bytes, allowing a maximum of 15 historic bytes. The entire ATR length can be a maximum of 33 bytes [14]. The IDEMIA ID card ATR (Figure 5) is 22 bytes long, consisting of 10 byte header, 11 byte long historic bytes and one TCK byte [8].

```
3B DB 96 00 80 B1 FE 45 1F 83 00 12 23 3F 53 65 49 44 0F 90 00 F1
```

Figure 5. IDEMIA ID card's ATR in hexadecimal [8].

Answer To Select (ATS) is sent instead of an ATR if a Java Card's contactless interface is used. It contains a mandatory length byte (TL byte). It encodes the total length of the ATS, TL byte included. It is followed by an optional format byte, interface bytes, historical bytes and one CRC byte. The ID1 has an ATS (Figure 6) that is 16 bytes long, consisting of a four byte header, 11 historic bytes and a CRC byte [16].

```
3B 8B 80 01 00 12 23 3F 53 65 49 44 0F 90 00 A0
```

Figure 6. IDEMIA ID card's ATS in hexadecimal [8].

2.3 Filesystem

The IDEMIA ID card's filesystem (Figure 7) is entirely emulated, where data is held in Java objects and movement through the filesystem is tracked with pointers [17]. There are three file types: master file (MF), dedicated file (DF) and elementary file (EF). Each file has a two byte file identification (FID) code. The applet itself has an applet identification (AID) code for selecting the applet, as multiple applets are supported on the IDEMIA card [8].

The master file is the root of the filesystem with an FID of `0x3F00`. It is the root folder of the emulated filesystem. DF files act as sub directories, where they themselves do not contain any readable data. EF files contain the actual data. This thesis focuses on the personal data DF with a FID of `0x5000` and its EFs. There are 15 personal data files, ranging from PD1 to PD15, that contain information about the cardholder. Their FIDs start from `0x5001` up to `0x500F` respectively. All the info present in PD1 to PD9 are visible on the physical IDEMIA card. If no info is present, a single `0x00` byte will be present [8].

```
IAS-ECC Root
|-- Document Number
|-- EF.Dir
|-- EF.CardAccess
|-- PIN1
|-- PUK
|-- Police Key
|-- DF Personal Data
|       |-- PD1 (Surname)
|       |-- PD2 (First Name)
|       |-- PD3 (Sex)
|       |-- PD4 (Citizenship ISO3166 alpha-3)
|       |-- PD5 (Date and place of birth)
|       |-- PD6 (Personal Identification Code)
|       |-- PD7 (Document Number)
|       |-- PD8 (Expiry Date)
|       |-- PD9 (Date and place of Issuance)
|       |-- PD10 (Type of residence permit)
|       |-- PD11 (Notes Line 1)
|       |-- PD12 (Notes Line 2)
|       |-- PD13 (Notes Line 3)
|       |-- PD14 (Notes Line 4)
|       |-- PD15 (Notes Line 5)
```

Figure 7. IDEMIA ID card's personal data DF file structure [8]. Other parts of the file structure are omitted.

2.3.1 Reading the personal data EF

The following subsection is based on the official developer guide for the IDEMIA ID cards [18]. To read a personal data EF, first the applet has to be selected using the applet's AID. This must be done as the applet is not selected by default as it follows the IAS-ECC applet specification [2]. For this the terminal sends a `SELECT FILE` command (Table 1) to the terminal with the P_1 byte set to `0x04`, signifying it wants to select the applet and P_2 as `0x0C`. This will return the OK SW to the terminal if successful. Selecting the master file is optional[2]. Next, the personal data DF must be selected. For this a `SELECT FILE` command with P_1 byte set to `0x01`. Additionally two bytes as the FID must be sent as the data with the APDU. Finally the personal data EF must be selected using the same C-APDU as for the DF file but changing the FID bytes and P_1 byte to `0x02`.

SELECT FILE						
C-APDU						Comment
CLA	00					Standardised command
INS	A4					Select File command
P_1	**Value**	00	01	02	04	
	File type	MF	DF	EF	AID	
P_2	0C					Type of data returned
C_{data}	FID					The FID represented as two bytes

R-APDU	
SW	9000

Table 1. Sending the `SELECT FILE` command.

[2]Through testing it was found that the master file is already selected by default, contrary to the official developer guide's instructions.

To read the EF file, a `READ BINARY` command (Table 2) must be sent with the Le byte set to `0x00`, as the expected size of the incoming data is unknown to the terminal. If T=1 protocol is used, the R-APDU will contain the requested data and an OK SW. If T=0 is used, a `0x6CXX` SW is returned. The XX byte is the length of the requested data, so the `READ BINARY` command must be sent again with the Le byte set to the XX byte. After this the data is returned with an OK SW.

SELECT FILE		
C-APDU		**Comment**
CLA	00	Standardised command
INS	B0	Read Binary command
P_1	00	
P_2	00	
L_e	FID	Expected data length

R-APDU				
T=0			**T=1**	
L_e	**00**	**XX**	**SW**	9000
SW	6CXX	9000	**DATA**	Data bytes, size dependent on the EF
DATA		XX data bytes		

Table 2. Sending `READ BINARY` command.

3. Implementation

This section covers the development of the emulator[3], that emulates the personal data DF found on IDEMIA ID cards and explains how each required part was implemented. To successfully emulate this behaviour, there are four requirements for the emulator: implementation of the file system regarding the personal data DF, having the correct ATR and ATS and a way to retrieve the data that mimics the IDEMIA ID card. This section will also talk about the different object and variable lifetimes in the Java Card's VM.

The emulator consists of two parts: the emulator applet (hereafter referred to as applet) and the Java Card card, where the applet is installed. Two cards were used for building and testing the applet for redundancy. The first card is the Feitian D11CR (hereafter referred to as the Feitian card). It supports the Java Card 2.2.2 specification and GlobalPlatform 2.1.1 for card management [19]. The second card is G&D SmartCafe Expert 6.0 80K Dual Card (hereafter referred to as the SmartCafe card). It supports the Java Card 3.0.1 specification but the same GlobalPlatform specification [20]. The Feitian card supports the T=0 protocol for contact interface but the SmartCafe card supports both T=0 and T=1 protocols for the contact interface. Both cards support T=CL over the contactless interface.

3.1 Transient and persistent memory

There are two main ways data is stored on Java Cards: either in Random Access Memory (RAM) or in Electrically Erasable Programmable Read-Only Memory (EEPROM). EEPROM acts as the persistent memory where data persists over power cycles. RAM acts as the temporary memory, when the card is powered. Writing to EEPROM wears it down meaning that the use of it is limited [21].

Persistent objects are static variables, class instance variables and all created arrays. Transient objects are variables defined in the scope of a method, method parameters and the APDU buffer that is used to receive and send APDUs. To create an array that is transient, `javacard.framework.JCSystem.makeTransientByteArray()` must be used. The `JCSystem` class also has methods to create boolean, short and object arrays [22].

[3]https://github.com/RobinMurk/IDEMIA-ID-card-emulator

3.2 Applet layout

For the emulator applet development, Java Standard Edition 8 was used for code compilation [23]. For the development, an API meant for Java Card applet development is used [24]. This API gives access to the `javacard.framework` package, that contains most of the necessary Java objects needed.

The emulator applet itself is a class that extends the `javacard.framework.Applet` class and is named "IDApplet" (Listing 1). It contains five main methods that belong to the Applet class [25]:

1. constructor - though not strictly necessary, it is recommended to use one for non-final variable declaration;
2. `install()` - method is only called once when the applet is loaded onto the card. It constructs the applet and uses the `register()` method to register the applet with the Java Card runtime and assigns the AID as its identifier;
3. `select()` method - this method is invoked when the applet is selected by the terminal by sending a `SELECT FILE` command with the applet's AID bytes as the data. If the applet is installed as the default applet, then this method is invoked as soon as the first C-APDU is sent by the terminal;
4. `deselect()` - method is invoked when the applet is deselected. This happens if the terminal selects another applet, even if it reselects the current applet;
5. `process()` - this method is invoked every time an APDU is received by the card. The `select()` method must have been called before this method can be accessed by the terminal. This is the main method that processes the incoming C-APDU and from where the R-APDU is sent. The incoming C-APDU is accessed through the `javacard.framework.APDU` object. If this method finishes without any error SW being thrown, it automatically sends the OK SW (`0x9000`). If a different SW must be sent, the `javacard.framework.ISOException.throwIt()` method is used. The `javacard.framework.ISO7816` class defines basic ISO-7816 compliant static SW values that are used to return SWs.

```
public class IDApplet extends Applet {

  //file pointers and constants
  private final static byte MF = 0x00;
  private final static byte DF = 0x01;
  private final static byte EF = 0x02;
  private static byte FILE_POINTER;
  private static byte FILE_NAME_FIRST;
  private static byte FILE_NAME_SECOND;

  //personal data
  private static final byte[] SURNAME = {
        (byte)0x4A, (byte)0xC3, (byte)0x95, (byte)0x45, (byte)0x4F,
        (byte)0x52, (byte)0x47
        };
  private static final byte[] FIRST_NAME = {
        (byte)0x4A, (byte)0x41, (byte)0x41, (byte)0x4B, (byte)0x2D,
  (byte)0x4B, (byte)0x52, (byte)0x49, (byte)0x53, (byte)0x54,
        (byte)0x4A, (byte)0x41, (byte)0x4E};
  private static final byte[] SEX = {77}; //M
  private static final byte[] CITIZEN = {69, 83, 84};  //EST
  private static final byte[] BIRTH = {48, 56, 32, 48, 49, 32, 49, 57, 56, 48, 32, 69, 83, 84};
  private static final byte[] ID_CODE = {51, 56, 48, 48, 49, 48, 56, 53, 55, 49, 56};
  private static final byte[] DOC_NUMBER = {65, 83, 48, 48, 48, 48, 51, 52, 51};
  private static final byte[] EXPIRY_DATE = {49, 51, 32, 48, 56, 32, 50, 48, 50, 51};
  private static final byte[] ISSUANCE = {49, 51, 32, 48, 56, 32, 50, 48, 49, 56};
  private static final byte[] RESIDENCE_TYPE = {0};
  private static final byte[] NOTES1 = {0};
  private static final byte[] NOTES2 = {0};
  private static final byte[] NOTES3 = {0};
  private static final byte[] NOTES4 = {0};
  private static final byte[] NOTES5 = {0};

  //historical bytes
  private static final byte[] ATRHistBytes =new byte[] {
    (byte)0x00, (byte)0x12, (byte)0x23, (byte)0x3F, (byte)0x53, (byte)0x65,
    (byte)0x49, (byte)0x44, (byte)0x0F, (byte)0x90, (byte)0x00, (byte)0xF1
  };

  private static final byte[] ATRoriginalFeitian =new byte[] {
    (byte)0x09, (byte)0x44, (byte)0x31, (byte)0x31, (byte)0x43,
    (byte)0x52, (byte)0x02, (byte)0x00, (byte)0x25, (byte)0xC3
  };

  private static final byte[] ATRoriginalEMV =new byte[] {
    (byte)0x53, (byte)0x43, (byte)0x45, (byte)0x36, (byte)0x30, (byte)0x2D,
    (byte)0x43, (byte)0x44, (byte)0x30, (byte)0x38, (byte)0x31, (byte)0x2D,
    (byte)0x6E, (byte)0x46
  };

  private static final byte[] aid = new byte[] {
    (byte)0xA0, (byte)0x00, (byte)0x00, (byte)0x00, (byte)0x77, (byte)0x01, (byte)0x08,
    (byte)0x00, (byte)0x07, (byte)0x00, (byte)0x00, (byte)0xFE, (byte)0x00, (byte)0x00,
    (byte)0x01, (byte)0x00
  };
  private final AID testAID;

  //logging info
  private static short LOGGER_ALLOCATED_SIZE;
```

```
  private static byte[] APDULog;
  private static short logDataSize;
  private static short startingPosPointer;
  private static short logDataLeft;

  //constructor
  private IDApplet() {
    LOGGER_ALLOCATED_SIZE = 0x0800;
    APDULog = new byte[LOGGER_ALLOCATED_SIZE];
    logDataSize = 0;
    startingPosPointer = 0;
    logDataLeft = 0;
    testAID = new AID(aid, (short)0, (byte)aid.length);
  }

  /**
   * Method invoked once when installing the applet on the JavaCard
   * @param ba
   * @param offset
   * @param len
   */
  public static void install(byte[] ba, short offset, byte len) {
    (new IDApplet()).register();
  }

  //method is invoked when the applet is selected
  //the T protocol is logged
  public boolean select(){
    logByte(APDU.getProtocol());
    return true;
  }

  public void deselect(){
    logByte((byte)0xDE);
  }

  /**
   * Main method invoked by the terminal that processes the incoming APDU
   */
  public void process(APDU apdu) {
        . . . .
        }
}
```

Listing 1. Emulator applet general layout with personal data EFs as example ID card's values (Figure 1).

In the main `process()` method, the commands are identified and filtered using switch statements (Listing 2). Firstly by the CLA byte and then the INS byte of the APDU. These bytes are received from the APDU buffer that is received by calling the `apdu.getBuffer()` instance method. This returns a byte array that contains the C-APDU bytes, firstly only the C-APDU header bytes and later the data bytes as well. The commands themselves are divided into two: real commands and mock commands. Real commands are the commands that are expected from the terminals used

in stores that were tested in this study. Mock commands are custom card and applet management commands, used mainly for accessing logs (Section 3.4) and changing historic bytes (Section 3.5). The mock commands are identified when the C-APDU header INS byte is a `0xDD` byte. If the CLA byte is not `0x00`, a SW of `0x6E00` is returned, signifying that such CLA is not supported. If the INS byte is not supported a `0x6D00` SW is returned to the terminal.

```
public void process(APDU apdu) {

  byte[] buf = apdu.getBuffer();

  //log header first
  if(buf[ISO7816.OFFSET_INS] != (byte)0xDD){
    logAPDU(buf, apdu, (short) 5, true);
  }

  //for checking main actions requested by terminal
  switch (buf[ISO7816.OFFSET_CLA]) {
    case (byte)0x00:
      switch (buf[ISO7816.OFFSET_INS]) {
        case (byte)0xA4:
          short len;
          len = buf[ISO7816.OFFSET_LC] == (byte)0x00 ? (short)0:apdu.setIncomingAndReceive();
          logAPDU(buf, apdu, len, false);
          selectFile(buf, apdu, len);
          return;
        case (byte)0xB0:
          readFile(buf, apdu);
          return;
        case (byte)0xDD:
          process_mock_commands(buf, apdu);
          return;
        default:
          logAPDU(buf, apdu, getLcValue(buf), false);
          ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
          return;
      }
    default:
      logAPDU(buf, apdu, getLcValue(buf), false);
      ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
      return;
    }
  }
```

Listing 2. The `process()` method.

3.3 Filesystem

This section goes over how the logic for handling the `SELECT FILE` and `READ BINARY` commands for the personal data DF and EF was implemented. These are the only commands that needed to be implemented as no other command is used for retrieving the EF files [18].

3.3.1 Handling file selection

The SELECT FILE command is processed in the method selectFile() (Listing 3) in a simple switch case. The method is called in the process() method. As the C-APDU is sending data, the APDU instance method setIncomingAndRecive() is called before accessing the data (Listing 2). This populates the APDU buffer with the incoming data[4] and returns the incoming data's length. As the filesystem is emulated (Section 2.3), a pointer "FILE_POINTER" is used to store a byte that indicates the file type that has been selected. The byte value corresponds to the SELECT FILE P_1 byte value (Table 1). The incoming FID's two bytes are kept separate as "FILE_NAME_FIRST" and "FILE_NAME_SECOND". If the correct SELECT FILE command for the applet AID is sent, then this will be also caught in the selectFile() method (Listing 2). A javacard.framework.AID object is created with the correct IDEMIA ID card applet AID, that the incoming C-APDU data is compared to. If they do not match, a 0x6A82 SW is returned.

```
private void selectFile(byte[] buf, APDU apdu, short length){
  switch (buf[ISO7816.OFFSET_P1]) {
    case MF:
      FILE_POINTER = MF;
      return;
    case DF:
    case EF:
      if(getLcValue(buf) == (short)2){
        FILE_NAME_FIRST = buf[ISO7816.OFFSET_CDATA];
        FILE_NAME_SECOND = buf[ISO7816.OFFSET_CDATA + 1];
        FILE_POINTER = buf[ISO7816.OFFSET_P1];
        return;
      }else{
        ISOException.throwIt((short)0x6A87);
      }
      break;
    case (byte) 0x04:
      if(!testAID.partialEquals(buf, ISO7816.OFFSET_CDATA, (byte)length)){
        ISOException.throwIt(ISO7816.SW_FILE_NOT_FOUND);
      }
      return;
    default:
      ISOException.throwIt(ISO7816.SW_INCORRECT_P1P2);
      return;
  }
}
```

Listing 3. The selectFile() method for SELECT FILE command processing.

[4]Through testing it was found that the Feitian card populates the APDU buffer with the data bytes without having to call the setIncomingAndReceive() method.

3.3.2 Handling EF reading

For the personal data EFs, 15 byte arrays are hardcoded into the emulator applet with the authors data for testing. These arrays are static and final (Section 3.1). `READ BINARY` commands are handled by the `readFile()` method. Firstly it is checked that the personal data DF is selected, if not then a `0x6A82` SW is returned. Depending on the "FILE_NAME_SECOND" pointer value, the corresponding value is sent back to the terminal. For the sending of data, the `sendPD()` method (Listing 4) is used. The data is copied into the APDU buffer using the `javacard.framework.Util.arrayCopyNonAtomic()` method. After copying the data to the buffer, APDU instance method `setOutgoingAndSend()` is invoked to send the data to the terminal. The data will actually be sent only when a SW is determined, meaning data inside the buffer cannot be changed after this method is called, as this might result in erroneous behaviour. Depending on the transmission protocol, the sending of data is handled differently.

For the T=0 protocol a `0x6CXX` SW is sent as the R-APDU, where the XX bytes encode the length of the data requested. This must be done as the T=0 protocol is byte oriented, meaning the terminal and the card do not know how many bytes are being received (Section 2.3.1). When the `READ BINARY` command with the L_e byte[5] bigger than 0 is received, the selected data is returned with the length of the received L_e byte. For the T=1 and T=CL protocol data can be sent directly, as the T=1 and T=CL protocol header contains the length of the APDU, saving the terminal and the card one round of APDU exchanges.

```
private void sendPD(byte[] DATA, byte[] buf, APDU apdu){
  if (APDU.getProtocol() == APDU.PROTOCOL_T0) {
    if(buf[ISO7816.OFFSET_LC] == (byte)0x00){
      //send back the expected length
      short sendValue = (short)(0x6C00 | ((short)DATA.length & 0xFF));
      ISOException.throwIt(sendValue);
    }else{
      short len = getLcValue(buf);
      Util.arrayCopyNonAtomic(DATA, (short)0, buf, (short)0, len);
      apdu.setOutgoingAndSend((short)0, len);
    }
  }
  else {
    Util.arrayCopyNonAtomic(DATA, (short)0, buf, (short)0, (short)DATA.length);
    apdu.setOutgoingAndSend((short)0, (short)DATA.length);
  }
}
```

Listing 4. `sendPD()` method used to return EF data to the terminal.

[5]This is a case two C-APDU.

3.4 Logging

To log the C-APDUs and transmission protocols, a static byte array named "APDULog" is created with a size of 2048 bytes. As the byte array is static, it is written into EEPROM and will persist (section 3.1). With it, three static pointers are also instantiated: "logDataSize" that points to the next position, where to write in the log array; "startingPosPointer" for keeping track of the starting position for when data is being retrived from the card; "logDataLeft" keeps track of how much data is left to send back. To log the data, two methods are used `logByte()` for logging the `select()` and `deselect()` methods and `logAPDU()` for logging the C-APDU buffer. All data is prefixed with the length of the data being logged. This allows for easy reading and filtering of data.

To see the communication between the terminal and the card, all C-APDUs are logged. In the `select()` method the transmission protocol is logged by calling the `APDU.getProtocol()` class method. This method returns three options: `0x00` byte for T=0 protocol, `0x01` for T=1 and `0xX1` for T=CL protocol[6]. In the `deselect()` method a single `0xDE` byte is logged. For the C-APDU, it is logged in two parts. Firstly the header of the C-APDU is logged in the beginning of the `process()` method. This is for debugging and informational reasons. Mock commands are not logged[7]. For the `SELECT FILE` command data is logged separately, where the length received from the `setIncomingAndReceive()` method is used. If the C-APDU is not supported by the applet, the data is logged depending on the L_c byte value as the length of the data using the `getLcValue()` method.

To receive and manage the logger, three methods are created. To receive the data, the `getLoggerData()` is used. This method behaves differently depending on the transmission protocol used. For T=1 and T=CL the APDU `sendBytesLong()` instance method is used. This allows us to send all the data at once. When T=0 is used, the emulator applet has to communicate to the terminal the length of the data being sent. For this `0x6CXX` SW is used to inform the terminal of the length of the data being sent (Section 3.3.2). Here the `sendBytesLong()` is also used as it allows us to use an external byte array for the data. If the `sendBytes()` method was used, the "APDULog" must first be copied to the APDU buffer as the method is only used

[6]From testing it was found that the Feitian card returns the byte as `0x81` and the SmartCafe card returns it as `0x91`.
[7]There is a small possibility that the terminal sends proprietary C-APDUs with the mock command's `0xDD` INS byte. These C-APDUs are not logged but they are still processed by the emulator applet.

for sending the APDU buffer data. To reset the "APDULog", meaning delete all the data, a `resetLogger()` method is used. All pointers are set to value 0 and all data inside the array is replaced by `0x00` bytes using the `Util.arrayFillNonAtomic()` method. There is also a private `getLoggerIndex()` method that returns the "logDataSize" pointer value. This is only used for debugging purposes.

3.5 ATR and ATS

Terminals could check the ATR and ATS sent by the card to determine if C-APDUs are sent to the card. Currently only the historic bytes of the ATR and ATS can be changed, as the header of the ATR and ATS contain electrical communication info, that cannot be modified. A mock command is used to remotely set the historic bytes, where the P_1 byte denotes that the historic bytes are to be changed, and P_2 denotes what bytes are to be set for the historic bytes. There are three possibilities to set the bytes: the Feitian card's, SmartCafe card's or the IDEMIA ID card's historic bytes[8]. The bytes are hardcoded as static byte arrays. To actually set the historic bytes, the `org.globalplatform.GPSystem.setATRHistBytes()` method is used using the GlobalPlatform API [26]. The same method changes the ATS historic bytes as well. Because only the historic bytes were changed to the IDEMIA card's historic bytes, the ATRs and ATSs do not match the IDEMIA card's ATR and ATS (Table 3). This problem would be mitigated if a card with the same prefix bytes was used but there exists only one card with such prefix bytes and it is not commercially available [27].

	ATR	**ATS**
IDEMIA ID card	3B DB 96 00 80 B1 FE 45 1F 83 00 12 23 3F 53 65 49 44 0F 90 00 F1	10 7A 77 C0 02 00 12 23 3F 53 65 49 44 0F 90 00
Feitian D11CR	3B 6C 00 00 00 12 23 3F 53 65 49 44 0F 90 00 F1	11 78 77 92 02 00 12 23 3F 53 65 49 44 0F 90 00 F1
G&D SmartCafe Expert 6.0 80K Dual Card	3B FC 18 00 00 80 31 FE 45 00 12 23 3F 53 65 49 44 0F 90 00 F1 B5	11 78 B3 B4 02 00 12 23 3F 53 65 49 44 0F 90 00 F1

Table 3. ATR and ATS values after changing the historic bytes to IDEMIA ID card's historic bytes over the T=1 protocol.

[8]For the IDEMIA ID card's historic bytes the checksum `0xF1` byte was also included.

3.6 Loading of the emulator applet to the card

To install the emulator applet onto a blank Java Card card, it must first be compiled into a class file. This in turn must be turned to a CAP file that is actually installed onto the card. To turn the emulator applet into a CAP file, a tool called "Apache Ant" is used. "Apache Ant" is a XML based automation tool that is used for CAP file creation. It uses tasks as instructions to define what to do. These tasks are defined in a "buildV2.xml" (Listing 5) file. To compile and create a CAP file, a custom task named "ant-javacard.jar" must be imported and defined in the "build.xml" file. This task was made by Martin Paljak and is currently the most common way that Java Card applet CAP files are created [28]. The task also requires the Java Card SDK used for development, which is provided also by Martin Paljak's repository [29]. As the Java Card platform of the cards is 2.2.2, the "jc222_kit" is used. As an external `org.globalplatform` package is used, it must be imported separately. The AID of the emulator applet is also defined in the "buildV2.xml". For this a custom AID of A01111117701080007000FE111111 is used. IDEMIA's ID card applet AID[9] is not used. This is done as the Java Card's VM would otherwise reselect the already selected emulator applet when the command for selecting the IDEMIA ID card's applet is received by the card. To mimic the selection of the ID applet by the terminal as close as possible, the AID must be changed. To compile the CAP file, the command `ant -f buildV2.xml` was used.

```
<?xml version="1.0" encoding="UTF-8"?>
<project default="applet" basedir=".">

  <taskdef name="javacard" classname="pro.javacard.ant.JavaCard" classpath="ant-javacard.jar"/>

  <target name="applet">
    <javacard jckit="jc_kits/jc222_kit/">
      <cap output="IDapplet_V2.cap" sources="src">
        <!--<applet class="IDcard.IDApplet" aid="a000000077010800070000fe00000100"/> -->
        <applet class="IDcard.IDApplet" aid="a01111117701080007000fe111111"/>
        <import exps="ext" jar="lib/gp211.jar"/>
      </cap>
    </javacard>
  </target>
</project>
```

Listing 5. Apache Ant "buildV2.xml" file used for CAP file creation.

[9]IDEMIA ID card's AID: A000000077010800070000FE00000100

Once the CAP file is created, it is then loaded onto the blank Java Card using the "GlobalPlatformPro" tool that was developed by Martin Paljak [30]. To install the CAP file, the blank Java Card is inserted to a card reader and the command `java -jar gp.jar --install IDapplet_V2.cap --default` was used[10]. The `-default` flag makes the applet selected by default. This is required, as it allows for all C-APDUs sent to the card to be logged. This means that when the terminal starts sending C-APDUs, the `select()` method is called, first thing logged is the transmission protocol (Section 3.4) and all C-APDUs are logged after that.

[10]The SmartCafe card needs an extra `-emv` flag as the card uses a special key derivation algorithm

4. Data gathering and analysis

This section will cover the methodology of how the data was gathered and what stores were chosen for testing. Later the collected data is aggregated and analysed.

4.1 Methodology for data gathering

To find out if the emulator could authenticate to terminals used in practice, 22 stores were identified that use the ID card as a client card (Table 4). Of the 22 stores identified 16 were chosen for testing (Table 5). These stores also used terminals that had a contactless interface, so authentication over the contactless interface was also tested. Most of the stores tested did not have a supervised authentication process[11]. These locations were chosen as to not arouse suspicion among store clerks that could lead to unintended misunderstandings. This could be mitigated by covering the cards with a fake cover, talked about in the 2017 study [5].

For testing, the same developed emulator applet was used on both of the blank Java Cards. Most of the store clerks were not notified of the experiment. A few terminals tested, that had human supervision, were notified. It was also asked of them to not deviate from normal procedure when authenticating meaning to act as the tested cards were normal ID cards. Each terminal was tested four times: two times with the first card over contact and contactless interface and two times with the second card, also both interfaces. Each successful authentication attempt was also marked. After each attempt to authenticate, the card's C-APDU log was extracted using a custom Python 3 script[12], after which the card's logger was reset. The script also adds context to the extracted data, allowing for better readability.

After the tests were completed, the logs were analysed to see what C-APDUs were sent to the card, what data was read, what transmission protocol was used and what are the differences between the contact and contactless interface logs. It is important to note that at the end of each log, there will always be a line "T=0" logged. This happens because when requesting the logs from the cards, the `select()` method is called (section 3.4). This last line is omitted from the logs included in the appendices of this work.

[11]The non-supervised terminals were mostly self checkout or food ordering terminals.

[12]https://github.com/RobinMurk/IDEMIA-ID-card-emulator/blob/main/scripts/send_mock_commands.py

nr.	Store	Non-supervised	Was tested
1.	K - rauta construction store	Yes	Yes
2.	Bauhof construction store	Yes	Yes
3.	Kalevi candy store	No	No
4.	Olerex gas station	Yes	Yes
5.	Rahva raamat book store	Yes	Yes
6.	Vapiano food store	Yes	Yes
7.	Terminal gas station	Yes	Yes
8.	Brain games store	No	Yes
9.	Prisma store	Yes	Yes
10.	Klick electronics store	No	Yes
11.	Alexela gas station	Yes	Yes
12.	Apotheka pharmacy	No	Yes
13.	Decora construction store	No	No
14.	PetCity pet store	No	No
15.	Apollo theatre	Yes	Yes
16.	Apollo store	Yes	Yes
17.	Tom Tailor clothes store	No	No
18.	Boost drink stand up	No	No
19.	Tokumaru restaurant	No	Yes
20.	Print in City Printer	Yes	Yes
21.	MySushi restaurant	No	No
22.	Olerex store	Yes	Yes

Table 4. List of stores where the Estonian ID card is used as a client card.

4.2 Results

The final results of testing can be seen in Table 5. Out of 16 terminals tested 10 terminals were able to successfully authenticate the emulated ID card over the contact interface. All terminals that communicated with the cards used T=0 transmission protocol. None of the terminals were able to successfully authenticate the emulator over the contactless interface. Eight different terminals were identified: Ingenico Lane/3000 (Appendix A Figure 1), Worldline YOMANI (Appendix A Figure 2), Ingenico iPP320, Ingenico iUC150B[13], Ingenico Desk/1600, Ingenico Move/3500, an Ingenico iUR250[14] (Appendix A Figure 3) and Verifone P400 (Appendix A Figure 4).

No.	Store	Terminal(s)	ATR check	Records read
1.	Alexela gas station	Ingenico iUR250, iUC150B	No	First 9 records
2.	Apollo store	Ingenico iUR250, iUC150B	No	First 9 records
3.	Apollo theatre	Ingenico Lane/3000	No	First 9 records
4.	Bauhof store	Worldline YOMIANI	Yes	-
5.	Brain Games store	Ingenico Desk/1600	No	First 9 records
6.	K-rauta store	Worldline YOMIANI	Yes	-
7.	Klick store	Worldline YOMIANI	Yes	-
8.	Olerex gas terminal	Ingenico iUR250, iUC150B	No	First 9 records
9.	Olerex store terminal	Ingenico iPP320	No	First 9 records
10.	Print In City printer	Proprietary	Yes	-
11.	Prisma store	Worldline YOMIANI	Yes	-
12.	Rahva Raamat store	Ingenico Lane/3000	No	First 9 records
13.	Terminal gas station	Ingenico iPP320	No	First 9 records
14.	Tokumaru food store	Verifone P400	Yes	-
15.	Vapiano food store	Ingenico Lane/3000	No	First 9 records
16.	Apotheka pharmacy	Ingenico Move/3500	No	First 9 records

Table 5. Final result of testing the IDEMIA ID card emulator on terminals in practice.

[13]The iUC150B terminal only has a contactless interface

[14]The iUR250 terminal only has a contact interface

4.2.1 Ingenico terminals

All the terminals that successfully authenticated the emulator card over the contact interface were manufactured by Ingenico. The APDUs logged by these terminals (Appendix B and C) are very similar and show that the first nine personal data EFs were accessed, from PD1 up to PD9. Reading all personal data EFs is not necessary as only the personal identification code or the document number help to reliably identify the person. This raises concerns over what this unnecessary data collected is used for. We have no way of determining what is being done with this data as the terminal acts as a "Black Box" in this scenario.

From the logs we can also see that before the IDEMIA ID card's AID is selected the terminal tries to access payment card's applets. The first AID logged, `315041592E5359532E4444463031`, when decoded with ASCII it becomes `1PAY.SYS.DDF01`. This is a directory on payment cards, that lists the available payment applications on the card. The `SELECT FILE` commands following this, are the terminal's attempts to select specific payment applets [31]. After it fails to do so the IDEMIA ID card AID is selected and the contents are read.

When analysing the logs from Ingenico terminals it was discovered that the terminals send the `SELECT FILE` command for the personal data EFs with the P_1 byte set to `0x01`, which is the incorrect file type for EF files (see Section 2.3.1). Further testing with an actual IDEMIA ID card showed, that this is possible as well, meaning the IDEMIA ID card does not force the use of the correct P_1 value when sending `SELECT FILE` command for the EF file. This is allowed on the IDEMIA card according to the official developer guide [18].

4.2.2 Worldline YOMANI terminals

All Worldline YOMANI terminals tested failed to successfully read the cards. Prisma's store (Appendix D) and K-rauta's store (Appendix E) terminals act very similarly where they also try to access the `1PAY.SYS.DDF01` directory and payment applets. But instead of trying to select the AID of the IDEMIA ID card AID, it stops sending C-APDUs completely. The logger is built so that the C-APDU's header bytes are logged as soon as access to the C-APDUs header bytes is possible in the `process()` method and that there is no header logged means that no C-APDUs are sent. This would indicate that the terminal does check the card's ATR to determine if it is an ID card or not. This claim is supported also by the 2017 study, where they found that Worldline YOMANI terminals do check for the correct ATR of the card [5].

Notable difference was with Bauhof's and Klick's terminals (Appendix F) where the terminal tried accessing the AID of D233000000457374454944207633335, which corresponds to the AID of the previous generation Estonian electronic identity card (EstEID) applet [32]. The 45737445494420763335 bytes decoded in ASCII become "EstEID v35". This indicates the possibility that these terminals still support ID card authentication over the now depreciated ID card's.

4.2.3 Verifone P400

Only one terminal made by Verifone was tested where the terminal failed to read the cards. The APDU logs (Appendix G) show that the terminal was trying to access three applets. The first one is the previous generation EstEID applet AID identified in the previous section. The next AID decoded in ASCII is "ðEstEID ver 1." which corresponds to the EstEID applet AID used in applet versions 3.0 up to 3.5. A 0x30 byte is missing from the end of the logged AID. In the logs the correct L_c byte 0x0E is logged, which denotes a length of 14 for the incoming data, but the in the documentation the AID is 15 bytes in length. This means there is an error in the documentation as the L_c byte should represent the correct length of 15 as 0x0F instead [33]. The last AID D233000001000001000000000000000 belongs to EstEID applet versions 3.0 (not included) and lower. Also here the last L_e byte 0x27 is missing from the end of the logs due to how the logger logs the incoming data bytes (Section 3.4) [33].

4.2.4 Contactless interface

Most of the terminals tested did not even try to communicate over the contactless interface when authenticating with the emulator card. The only terminals that did were the separate Ingenico iUC150B contactless terminals, Ingenico iPP320 terminals and Prisma's Worldline YOMANI terminal (Appendix H). These terminals only asked for one AID: 325041592E5359532E4444463031. This decoded in ASCII is 2PAY.SYS.DDF01 which is also a directory on payment cards, but this directory is only available on payment cards that have a contactless interface. When the 1PAY.SYS.DDF01 directory on payment cards is not mandatory then the 2PAY.SYS.DDF01 directory is [31]. This explains why this is the only C-APDU sent: as it receives the 0x6A82 SW, it stops communicating. It also confirms that the terminals that do wait for a connection on a contactless interface are using a different script to only access contactless payment cards, confirming that terminals do not try to read the ID card's personal data file over the contactless interface.

4.2.5 Print In City printer

The Print In City printer located in the Tartu University's Delta building completely ignored the emulator cards. The logs showed that nothing was logged from both interfaces and both cards. The transmission protocol was not logged meaning that no connection was made to the cards. This suggests that the terminal checks for the correct ATR before initializing a connection to the card. This does contradict the findings in the 2017 study, which found that the printers ignore the ATR during the authentication process [5]. It is possible that over the years the firmware of the terminal has been updated to check the ATR of the card inserted.

5. Conclusion

Stores use ID cards as client cards, using them to authenticate to store terminals. For the purpose of easing future testing and experimentation, this thesis set out to develop an IDEMIA ID card emulator, that emulates the personal data filesystem, so this authentication process could be performed. The developed emulator was able to successfully authenticate to over half of the terminals tested.

The thesis began with an overview of the technical specifications and standards used by the IDEMIA made ID card. After this a comprehensive breakdown of the communications between the ID card and the terminal was given, followed by the same for the card's filesystem, regarding the personal data file. This was followed by a description of the implementation of the emulator applet and its installment, ending with the testing of store terminals.

Out of 16 terminals tested, 10 were able to successfully authenticate the emulator card. The terminals that did successfully authenticate, all being manufactured by Ingenico, read the first nine personal data records. None of the terminals tested were able to successfully authenticate over the contactless interface.

The author suggests multiple improvements and possible future continuations of this work. Firstly, the test could be performed on more terminals, including the supervised ones. This could be done by covering the card with stickers, so the emulator card would look similar to the example ID card (Figure 1) released by the Estonian government. Secondly, if a blank Java Card with the ATR prefix bytes matching that of the IDEMIA ID card was used, a more thorough testing of the emulator applet could be done as it could then confirm if some of the terminals still check for ATR which is suspected in some of the tested terminals. Thirdly, the applet could be improved to have a more dynamic way of updating personal data EFs and ATR historic bytes so that testing of multiple scenarios, for example seeing if the terminals check for document expiry date, could be performed faster.

References

[1] Estonian Information System Authority. ID-card and its uses. https://www.id.ee/en/article/id-card-and-its-uses/ (05/09/2025).

[2] Parsovs A. Estonian electronic identity card and its security challenges. ISSN: 2613-5906. Mar. 2021. http://hdl.handle.net/10062/71481 (05/09/2025).

[3] Kivivare S.-K. Secure Channel Establishment for the NFC Interface of the New Generation Estonian ID Cards. https://comserv.cs.ut.ee/ati_thesis/datasheet.php?id=70557&language=en (05/09/2025).

[4] Estonian Information System Authority. Digital signing with NFC in the RIA DigiDoc mobile application. https://www.id.ee/en/article/nfc-abil-digiallkirjastamine-ria-digidoc-mobiilirakenduses/ (05/09/2025).

[5] Morgan D. and Parsovs A. Using the Estonian Electronic Identity Card for Authentication to a Machine. *Secure IT Systems*. Ed. by Lipmaa H., Mitrokotsa A., and Matulevičius R. Cham: Springer International Publishing, 2017, pp. 175–191. DOI: 10.1007/978-3-319-70290-2_11.

[6] Release Notes - Java Card Specification Version 2.2.1. http://pfa12.free.fr/doc_java/javacard_specifications/RELEASENOTES_jcspecs.html#documentation_roadmap (05/09/2025).

[7] GlobalPlatform. GlobalPlatform Card Specification 2.2.0.7. 2011. https://globalplatform.org/wp-content/uploads/2018/06/GPC_Specification-2.2.1.pdf.

[8] Estonian Information System Authority. TD-ID1 Chip Application Technical Specification. https://www.id.ee/wp-content/uploads/2021/08/td-id1-chip-app-4.pdf (05/09/2025).

[9] Estonian Information System Authority. The new ID-card version will come with some changes for developers. https://www.id.ee/en/article/the-new-id-card-version-will-come-with-some-changes-for-developers/ (05/09/2025).

[10] Tallinn University of Technology. Applying for Estonian ID-card. https://taltech.ee/en/estonian-id-card (05/13/2025).

[11] CardWerk Technologies. ISO7816-4 section 5 smart card standard basic organizations. https://cardwerk.com/smart-card-standard-iso7816-4-section-5-basic-organizations/ (05/09/2025).

[12] Paršovs A. Smart cards (EstEID). https://courses.cs.ut.ee/2024/appcrypto/spring/Main/HomePage?action=download&upname=10.pdf (05/09/2025).

[13] SW1 SW2 list. https://web.archive.org/web/20090623030155/http://cheef.ru/docs/HowTo/SW1SW2.info (05/09/2025).

[14] Everett D. B. Smart Card Technology: Introduction To Smart Cards - Page 3. http://www.sat-digest.com/SatXpress/SmartCard/ISO7816-3.htm (05/09/2025).

[15] Guo N. Smart Card Operation Using Freescale Microcontrollers. https://www.nxp.com/docs/en/application-note/AN4453.pdf (05/09/2025).

[16] ISO/IEC. ISO/IEC 14443-4 Proximity Cards – Transmission Protocol. http://www.emutag.com/iso/14443-4.pdf (05/09/2025).

[17] CardWerk Technologies. ISO 7816-4 Annex C: Record Pointer Management. https://cardwerk.com/iso7816-4-annex-c-record-pointer-management/ (05/09/2025).

[18] Estonian Information System Authority. ID1 Developer Guide. https://www.id.ee/wp-content/uploads/2020/10/id1developerguide.pdf (05/09/2025).

[19] Feitian Technologies. Feitian D11CR Smart Card – Product Datasheet. https://www.grama.es/wp-content/uploads/2016/07/feitian_d11cr_grama_en.pdf (05/09/2025).

[20] Giesecke & Devrient. SmartCafe Expert 6.0 – Product Datasheet. https://www.smartcardfocus.com/files/SCE3272K/Datasheet_SmartCafe%20Expert_Nov2010.pdf (05/09/2025).

[21] Oestreicher M. and Krishna K. Object Lifetimes in Java Card. https://www.usenix.org/legacy/publications/library/proceedings/smartcard99/full_papers/oestreicher/oestreicher.pdf (05/09/2025).

[22] Paršovs A. Smart cards (JavaCard). https://courses.cs.ut.ee/2022/appcrypto/fall/Main/HomePage?action=download&upname=11.pdf (05/09/2025).

[23] Paljak M. JavaCard SDK and JDK version compatibility. https://github.com/martinpaljak/ant-javacard/wiki/JavaCard-SDK-and-JDK-version-compatibility (05/09/2025).

[24] Paljak M. ”jc222_kit” Java Card API. https://github.com/martinpaljak/oracle_javacard_sdks/blob/master/jc222_kit/lib/api.jar (05/09/2025).

[25] Oracle Corporation. Java Card API Applet class documentation. https://docs.oracle.com/en/java/javacard/3.1/jc_api_srvc/api_classic/javacard/framework/Applet.html (05/09/2025).

[26] Paljak M. GlobalPlatform Java Card API. https://github.com/martinpaljak/AppletPlayground/tree/master/ext/globalplatform-2_1_1 (05/09/2025).

[27] List of supported JavaCard algorithms. https://www.fi.muni.cz/~xsvenda/jcalgtest/table.html (05/09/2025).

[28] Paljak M. Ant task for building Java Card applets. Apr. 2025. https://github.com/martinpaljak/ant-javacard (05/09/2025).

[29] Paljak M. Java Card SDK ”jc222”. https://github.com/martinpaljak/oracle_javacard_sdks/tree/master/jc222_kit (05/09/2025).

[30] Paljak M. GlobalPlatformPro. May 2025. https://github.com/martinpaljak/GlobalPlatformPro (05/09/2025).

[31] MST Company. Acquiring: EMV Transaction Flow. Part 2: PSE and AID, Candidate List and Application Selection. https://mstcompany.net/blog/acquiring-emv-transaction-flow-part-2-pse-and-aid-candidate-list-and-application-selection (05/09/2025).

[32] Estonian Information System Authority. EstEID Chip Application v3.5.8 – Technical Specification. https://www.id.ee/wp-content/uploads/2020/02/RIA-EstEID-Chip-App-v3.5.8_fix_form.pdf (05/09/2025).

[33] Estonian Information System Authority. EstEID Chip Application up to v3.5 – Technical Specification. https://www.id.ee/wp-content/uploads/2020/02/TB-SPEC-EstEID-Chip-App-v3.4.pdf (05/09/2025).

Appendices

A. Some of the terminals tested in this study

Figure 1. Ingenico Lane/3000 found in Vapiano.

Figure 2. Worldline YOMANI terminal found in Prisma store.

Figure 3. Ingenico iUC150B and iUR250 terminals found in Olerex gas station.

Figure 4. Verifone P400 found in Tokumaru.

B. APDU log, Ingenico unknown model (Alexela gas station, contact interface)

```
T=0
00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 10 //SELECT FILE: AID
00 A4 04 00 06 A0 00 00 00 25 01 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 20 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 30 60 //SELECT FILE: AID
00 A4 04 00 10 A0 00 00 00 77 01 08 00 07 00 00 FE 00 00 01 00 //SELECT FILE: AID
00 A4 01 0C 02 50 00 //SELECT FILE
00 A4 01 0C 02 50 01 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 05 //READ BINARY
00 A4 01 0C 02 50 02 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 05 //READ BINARY
00 A4 01 0C 02 50 03 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 01 //READ BINARY
00 A4 01 0C 02 50 04 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 03 //READ BINARY
00 A4 01 0C 02 50 05 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0E //READ BINARY
00 A4 01 0C 02 50 06 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0B //READ BINARY
00 A4 01 0C 02 50 07 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 09 //READ BINARY
00 A4 01 0C 02 50 08 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0A //READ BINARY
00 A4 01 0C 02 50 09 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0A //READ BINARY
```

C. APDU log, Ingenico Lane/3000, Desk/1600, iPP320, Move/3500 and integrated unknown terminal (Vapiano, Apollo, Olerex, Rahva Raamat, Terminal gas station, Brain Games, Apotheka, contact interface)

```
T=0
00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 20 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 30 60 //SELECT FILE: AID
00 A4 04 00 05 A0 00 00 00 25 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 03 33 01 01 //SELECT FILE: AID
00 A4 04 00 10 A0 00 00 00 77 01 08 00 07 00 00 FE 00 00 01 00 //SELECT FILE: AID
00 A4 01 0C 02 50 00 //SELECT FILE
00 A4 01 0C 02 50 01 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 05 //READ BINARY
00 A4 01 0C 02 50 02 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 05 //READ BINARY
00 A4 01 0C 02 50 0/ //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 01 //READ BINARY
00 A4 01 0C 02 50 04 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 03 //READ BINARY
00 A4 01 0C 02 50 05 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0E //READ BINARY
00 A4 01 0C 02 50 06 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0B //READ BINARY
00 A4 01 0C 02 50 07 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 09 //READ BINARY
00 A4 01 0C 02 50 08 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0A //READ BINARY
00 A4 01 0C 02 50 09 //SELECT FILE
00 B0 00 00 00 //READ BINARY
00 B0 00 00 0A //READ BINARY
```

D. APDU log, Worldline YOMIANI terminal (Prisma, contact interface)

```
T=0
00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 30 60 //SELECT FILE: AID
00 A4 04 00 06 A0 00 00 00 25 01  //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 65 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 01 52 30 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 03 79 00 00 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 01 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 02 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 03 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 10 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 11 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 12 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 04 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 05 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 03 79 01 00 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF F1 06 //SELECT FILE: AID
```

E. APDU log, Worldline YOMIANI terminal (K-rauta, contact interface)

```
T=0
00 A4 04 00 0E 31 50 41 59 2E 53 59 53 2E 44 44 46 30 31 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 03 20 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 04 30 60 //SELECT FILE: AID
00 A4 04 00 06 A0 00 00 00 25 01 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 00 65 10 10 //SELECT FILE: AID
00 A4 04 00 07 A0 00 00 01 52 30 10 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 01 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 02 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 03 //SELECT FILE: AID
00 A4 04 00 07 FF FF FF FF FF 01 04 //SELECT FILE: AID
```

F. APDU log, Worldline YOMIANI terminal (Bauhof, Klick), contact interface

```
T=0
00 A4 04 0C 0F D2 33 00 00 00 45 73 74 45 49 44 20 76 33 35 //SELECT FILE: AID
00 A4 04 0C 07 A0 00 00 01 51 00 00 //SELECT FILE: AID
```

G. APDU log, Verifone P400 (Tokumaru, contact interface)

```
T=0
00 A4 04 0C 0F D2 33 00 00 00 45 73 74 45 49 44 20 76 33 35 //SELECT FILE: AID
00 A4 04 00 0E F0 45 73 74 45 49 44 20 76 65 72 20 31 2E //SELECT FILE: AID
00 A4 04 00 10 D2 33 00 00 01 00 00 01 00 00 00 00 00 00 00 00 //SELECT FILE: AID
```

H. APDU log, Ingenico iUC150B, IPP320 and Worldline YOMIANI terminals (Olerex, Alexela, terminal, Prisma, contactless interface)

```
T=CL
00 A4 04 00 0E 32 50 41 59 2E 53 59 53 2E 44 44 46 30 31 //SELECT FILE: AID
```

License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Robin Mürk**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Estonian ID card's personal data file emulator,

   supervised by Arnis Paršovs.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Robin Mürk
10/05/2025