TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technology

Gregor Johannson 163304IAPM

TECHNICAL PREREQUISITES FOR ENABLING THIRD-PARTY APPLICATIONS ON THE NEW ESTONIAN ID-CARD

Master's thesis

Supervisor: Juhan-Peep Ernits
PhD

Co-supervisor: Martin Paljak

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gregor Johannson 163304IAPM

TEHNILISED EELTINGIMUSED KOLMANDA OSAPOOLE RAKENDUSTE LUBAMISEKS UUEL EESTI ID-KAARDIL

Magistritöö

Juhendaja: Juhan-Peep Ernits
PhD

Kaasjuhendaja: Martin Paljak

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Gregor Johannson

13.05.2019

Abstract

The Estonian authorities started issuing a new generation of ID-cards, the primary identity document of the citizens in the Republic of Estonia and one of the carriers of electronic identity, from the end of the year 2018. The ID-card is a Java Card smart card following the GlobalPlatform specification for content management. Among other features, this new generation of ID-cards includes multi-application support and a new NFC contactless interface. In order to take advantage of these new features, existing tools enabling content management on the smart cards need to be improved. As the new cards require Delegated Management for card content management, support for this needs to be implemented. In addition, the Estonian authorities currently have not made it their priority to publish information or guides for third parties on how development should be carried out, and what are the restrictions to acceptable applets and how applets will be installed on ID-cards in the future. This means that the development of useful applets for the ID-cards is currently not possible.

In order to fix these deficiencies in the current situation, the current thesis showcases adding support for Delegated Management to one of the most popular open-source tools to manage contents of GlobalPlatform smart cards, GlobalPlatformPro. After support for Delegated Management has been implemented, the added functionality is evaluated by attempting installation of an applet that implements a beneficial use-case to a test ID-card, also presenting a proof-of-concept. All required steps to successfully install an applet are thoroughly documented.

Next, in order to help accelerate publishing information and guides to help developers and allow the development of applets for the new ID-cards, enhancements to documentation are proposed, possible scenarios for enabling applet installation and the privileges of developers are discussed. Finally, potential threats to the ID-card are mapped from existing publications, and countermeasures are suggested to help reduce risks of opening the ID-card to third-party applets.

This thesis is written in English and is 50 pages long, including 7 chapters, 18 figures, 1 table and 3 appendices.

Annotatsioon

Tehnilised eeltingimused kolmanda osapoole rakenduste lubamiseks uuel Eesti ID-kaardil

2018. aasta lõpust on Politsei- ja Piirivalveamet asunud väljastama uue generatsiooni ID-kaarti, mis on peamine isikut identifitseeriv dokument Eesti Vabariigi kodanikel ja üks mitmest võimalikust elektroonilise identiteedi kandjast. ID-kaart on Java Card kiipkaart, mis järgib GlobalPlatform spetsifikatsiooni kaardi sisu haldamiseks. Lisaks kõigele muule sisaldab ID-kaardi uus generatsioon endas tuge mitme rakenduse hoidmiseks ning lähivälja kommunikatsiooni tehnoloogial (NFC) põhinevat kontaktivaba liidest. Nende funktsioonide parimaks kasutamiseks on tarvis täiendada olemasolevaid tööriistu, mida kasutatakse kiipkaartide sisu haldamiseks. Kuna uued ID-kaardid nõuvad *Delegated Management* tuge kaardi sisu haldamiseks, siis vastav funktsionaalsus tuleb tööriistades implementeerida. Lisaks, ID-kaarti elektroonilise identiteedi kandjana arendaval Riigi Infosüsteemi Ametil ei ole lähiajal kavas luua avalikuks kasutuseks arendust abistavat materjali või juhendeid kolmandatele osapooltele, ega kirjeldada millised on piirangud aktsepteeritavatele rakendustele ja kuidas neid rakendusi kavatsetakse tulevikus ID-kaartidele paigaldada. See tingib olukorra, kus kasulike rakenduste arendus ID-kaardile ei ole hetkel võimalik.

Nende puuduste parandamiseks käsitleb käesolev lõputöö *Delegated Management* toe lisamist ühele populaarseimale vabavaralisele GlobalPlatform kiipkaartide sisu haldamise tööriistale, GlobalPlatformPro. Peale *Delegated Management* toe implementeerimist hinnatakse lisandunud funktsionaalsust läbi kasuliku rakenduse paigaldamise test ID-kaardile. Lisaks esitleb edukas paigaldus ka *proof-of-concept*'i. Kõik vajalikud sammud rakenduse paigaldamiseks kirjeldatakse üksikasjalikult.

Järgmisena soovitatakse täiendusi olemasolevale dokumentatsioonile ning arutletakse võimalike rakenduse paigaldamise ja arendaja õiguste stsenaariumite üle, et aidata Eesti ametivõimudel avalikustada informatsiooni ja juhiseid arenduse aitamiseks ja lubamiseks

uutel ID-kaartidel. Viimasena kaardistatakse võimalikke ohte ning pakutakse vastumeetmeid, et aidata vähendada riske kolmanda osapoole rakenduste lubamisega.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 50 leheküljel, 7 peatükki, 18 joonist, 1 tabelit ja 3 lisa.

List of abbreviations and terms

PPA	Estonian Police and Border Guard Board
RIA	Estonian Information System Authority
RQ	Research Question
GP	GlobalPlatform
SCP03	Secure Channel Protocol 3
DAP	Data Authentication Pattern
VM	Virtual Machine
ISD	Issuer Security Domain
JCRE	Java Card Runtime Environment
AID	Applet Identifier

Table of contents

List of figures

List of tables

1 Introduction

1.1 Existing Body of Knowledge

The following sections provide a short and general overview of the new iteration of Estonian ID-cards, Java Card technology, the closely related GlobalPlatform specification and the open-source GlobalPlatformPro tool implementing it.

1.1.1 Estonian ID-card

The Estonian ID-card is the primary identity document in the Republic of Estonia. It is also a mandatory identity document for Estonian citizens and citizens of the European Union permanently residing in Estonia. In the European Union, the ID-card can be used as a travel document. The homepage of the ID-card (https://www.id.ee) also defines it as a "digital identity card or Digi-ID" [1]. This name is adopted since, unlike a traditional passport, the ID-card is a smart card – it can be used for authentication of the holder and to give digital signatures in an electronic environment. For example, the ID-card can be used for authentication when logging in to an e-governance portal, and to sign documents in the portal with a digital signature.

Until the end of 2018, the Estonian ID-cards were manufactured by Gemalto. Now, the Estonian Police and Border Guard Board (PPA) has partnered with a new ID-card manufacturer, Idemia. The first new ID-cards have, as of January 2019, just started arriving to the Estonian citizens. In addition to many other changes, one of the most important updates is that the new card has a contactless NFC interface [2]. Previously, the Estonian ID-card was a contact-only smart card, meaning that communication with any reader could only be carried out through a physical contact interface. This new iteration of ID-cards also enables third-party applications to be deployed and used by the cardholder. Combining the new contactless interface with multi-app support allows the card to have far more functionality than originally envisioned and intended.

This thesis tackles third-party smart card applet deployment onto the new iteration of Estonian ID-cards.

1.1.2 Java Card

"Java Card technology combines a portion of the Java programming language with a runtime environment optimized for smart cards and related, small-memory embedded devices" [3]. The previous and new Estonian ID-cards are Java Card smart cards, meaning they run Java code, albeit with some language limitations, for example missing support for all primitives that regular Java supports.

Similar to regular Java, Java Card source files are written in Java, and then compiled to class files. These are then converted to CAP (for *Converted APplet*) files, which reduce the image size for required download and reduce memory requirements at run-time. Finally, the loaded applet is executed using the Java Card Virtual Machine [4].

Smart cards are not just simple storage or computing devices, but security is equally, if not more important. Bouffard et al. [4] state that "the Java Card platform is a multi-application environment where the sensitive data of an applet must be protected against malicious access from another applet or from the external world". Guyot [5] describes that smart cards do this successfully, since "both hardware and software measures [6] prevent from retrieving [card] content if appropriate credentials are not provided to the smart card".

As stated in Section 1.1.1, this thesis demonstrates the deployment of third-party smart card applets onto Estonian ID-cards. As the ID-card is a Java Card enabled smart card (following the specification Java Card 3.0.4 Classic Edition [7]), the deployed applets are Java Card applets.

1.1.3 GlobalPlatform and GlobalPlatformPro

"GlobalPlatform is an organization that has been established by leading companies from the payments and communications industries, the government sector and the vendor community, and is the first to promote a global infrastructure for smart card implementation across multiple industries" [8]. GlobalPlatform has developed a specification that "… provides common security and card management architecture that protects the most important aspect of a chip card system investment — the infrastructure"

[8]. As the new Estonian ID-card referenced in Section 1.1.1 is compliant with the GlobalPlatform specification v2.2.1 [8], then this version is also used as a reference. It should be noted though, that a newer v2.3.1 [9] is also available as of March 2018.

Following the GlobalPlatform specification for loading and managing the contents, i.e. Java Card applications, of compatible Java Cards can be cumbersome and prone to errors. Many tools have been developed to simplify and standardise this process, the most popular of which, at the time of writing, seems to be the Java-based open-source GlobalPlatformPro[1], maintained by Martin Paljak.

In this thesis, GlobalPlatformPro is used to help manage contents of the GlobalPlatform compliant Java Card equivalent to the new Estonian ID-card.

1.1.4 Contribution

There currently are no third-party smart card applications specifically developed for the new ID-card, and specific documentation on the development and deployment of applets to the particular type of cards have not been made available to the public. Having contacted the Estonian Information System Authority (RIA), this is something they also feel is lacking in the current state of available resources. This thesis shows how third-party Java Card applets can be successfully deployed and used with the new Estonian ID-cards, and thus it presents a proof-of-concept for managing such applets.

In addition, since RIA has not released any how-to instructions or documentation for third-party developers, this thesis attempts to aid RIA in putting together a set of basic requirements and guidelines for RIA to publish. This will aid existing knowledgeable Java Card applet developers and hopefully spark interest in innovative individuals who can come up with new beneficial use-cases for the ID-card in the Estonian tech-savvy society.

To prototype possible use-cases on the ID-cards, GlobalPlatform specification requires card content changes with Delegated Management (DM). Loosely speaking, DM allows Card Issuers to grant content management rights to third parties on a separate logical

[1] https://github.com/martinpaljak/GlobalPlatformPro

section of a card, i.e. "… the possibility of empowering partnered Application Providers the ability to initiate approved and pre-authorized Card Content changes (loading, installing or extraditing)" [8]. When DM is required for Card Content changes, a signed Token must be included in commands sent to a card. The open-source tool GlobalPlatformPro, which is used to load and manage applications on the new ID-card, originally lacked support for DM. This thesis makes a contribution to GlobalPlatformPro by implementing DM support through generating these required tokens dynamically. At the time of writing, the appropriate modifications were made available to the maintainer of GlobalPlatformPro in the form of a pull request1.

Support for DM is a prerequisite for enabling the development of applications that cater to new use-cases, and it also enables the possibility of porting existing solutions to the ID-card that currently require a separate smart card or a custom device. These solutions include, for example, travelcards in public transport, access cards in company offices, authorisation cards at compatible electric car charging stations, NDEF tags for accessing any saved information with an NFC-enabled smartphone, One-Time Password generators, and so on.

1.2 Research Questions

As described in Section 1.1.4, the current thesis aims to fill a gap in the state of the art by adding Delegated Management Token generation functionality to the open-source tool GlobalPlatformPro. Through this functionality, the thesis aims to prove that third-party applets can be loaded onto the new Estonian ID-card. In addition, a set of basic development guidelines and requirements is specified, that can be used by RIA to develop appropriate developer documentation. Also, since there are many publications regarding smart card security, and opening up a smart card to applets introduces even more of them, potential threats and countermeasures to ID-cards have to be mapped.

The problems addressed in the current thesis are relevant for several reasons. First, as one of the most popular GlobalPlatform Java Card content management tools,

[1] https://github.com/martinpaljak/GlobalPlatformPro/pull/155

GlobalPlatformPro, currently does not support DM functionality, developers are left without options to prototype and load their applications on the ID-card test cards. Furthermore, as the Estonian authorities (RIA/PPA) have not released any instructions for applet developers or requirements for acceptable applets, the development of applets that would implement new use-cases is not possible. The authorities first need to confidently open up the card to applets, and this thesis aims to help them towards this goal.

1.2.1 Research Questions

The main research question (RQ) is formulated as follows: **How to develop third-party applets for the new Estonian ID-cards?**

In order to answer the question in a structured and comprehensive manner, the main research question is divided into three sub-questions:

- **RQ-1: How to add Delegated Management support to GlobalPlatformPro for managing content on the new ID-cards?**
- **RQ-2: How can third parties manage the contents of the new ID-cards?**
- **RQ-3: How should third parties be allowed to develop and prototype applets on the new ID-cards?**

1.3 Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 provides a close look at Java Card architecture and development, followed by an overview of the GlobalPlatform standard and specification. Chapter 3 researches the existing functionality of the open-source tool GlobalPlatformPro and describes changes made to the tool to provide support for generating Delegated Management Tokens from calculated command data. Afterwards, in Chapter 4, these enhancements to GlobalPlatformPro are used to describe the process required to successfully load Java Card applets to the test-version of new Estonian ID-cards. In addition, an example applet for a noteworthy use-case is chosen, described, installed and tested, to best showcase the multi-application support and contactless interface of the new Estonian ID-cards. Chapter 5 suggest different methods of approach to how the Estonian authorities should allow third-party Application Providers to develop

and deploy their applets. Also, vulnerabilities and countermeasures regarding the smart card and installed applets are mapped, to give a better understanding of involved risks with allowing installation of third-party applets. Chapter 6 evaluates the work done in this thesis through the Estonian authorities and describes related work in the field. Chapter 7 concludes the thesis and provides an outlook on future work.

2 Background

The following chapter introduces fundamental concepts that expand on the existing body of knowledge and provides the reader with a more comprehensive overview of Java Card smart cards and the GlobalPlatform protocol that defines a common security and card management architecture for these cards.

2.1 Java Card

The following sub-sections introduce Java Card, its architecture and related standards (Section 2.1.1) and development of Java Card applets (Section 2.1.2).

2.1.1 Architecture

As stated in Section 1.1.2, a Java Card is a secure computational device that runs a subset of Java code, compiled into a CAP file. Latest Java Card smart cards implement the standard Java Card 3 [3], implementing a virtual machine (VM) that interprets code that is already packaged with the card at issuance, or downloaded after issuance. Security requirements prohibit downloading code from any source, and these requirements are defined in the protocol standardised by GlobalPlatform. Throughout the current thesis, when talking about Java Card, the "Classic Edition" of Java Card is referred to.

The security of a smart card is provided mainly by its architecture. The components of a smart card are illustrated in Figure 1. Guyot [5] describes: "The core part, hidden behind the external contacts of a smart card, is a complete miniature computer, including a Central Processor Unit (CPU) to process data, an EEPROM module to provide offline storage capacity, a RAM module to handle data processing, a serial port I/O (Input/Output) to communicate with the smart card reader, and most of the time a crypto co-processor to accelerate the execution of cryptographic algorithms". ROM is used to store the operating system and "romized" applications. Iguchi-Cartigny et al. [10] describe the difficulty of physical attacks, since a card is also embedded with sensors, such as light, heat, and voltage sensors, that disable the card in case of physical attacks.

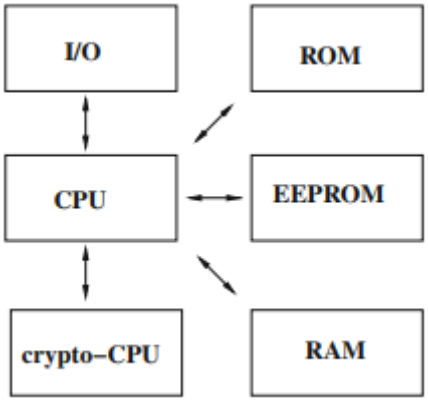

Figure 1. Components of smart cards [5].

A smart card has strong memory constraints, most cards only allowing a few hundred kB for storage (144 kB EEPROM and 512 kB ROM on the new ID-card [11]). Due to these resource constraints, the VM must be split into two parts: the bytecode verifier that is executed off-card, invoked by a converter, and the interpreter, API and Java Card Runtime Environment (JCRE) are executed on the card. The bytecode verifier is the main security measure of a card, performing static code verifications required by the VM. A bytecode converter then transforms the validated Java class files to the CAP file format. The interpreter, API and JCRE are in charge of handling applet behaviour on the card. [10]

Since Java is strongly typed, variable and expression types are determined at compile-time, allowing detection of type mismatches, even in bytecode. Local and stack variables of the VM, however, do not have fixed types even in the scope of one method execution. Since not all type mismatches are detected at runtime, malicious applets can be developed to exploit this issue. In addition, even though Java does not support pointers, the underlying VM uses them extensively, and thus attempts to break security through pointers cannot be outruled. The bytecode verifier is crucial to detect ill-typed applets, but since the process is time-consuming and involves elaborate program analyses, cards do not implement such a component, but rely on the fact that bytecode is verified before downloading on the card and this is assumed to be the responsibility of the organisation signing the code. [10]

In addition, there exists a firewall in Java Card cards. The firewall enforces separation between applets, based on a packaged structure and the notion of contexts. The JCRE uses a unique applet identifier (AID), which enables retrieving the package name in which

it is defined. Two applets are considered belonging to the same context if they are instances of classes coming from the same Java Card package. Every object is assigned an owner context, connected to the applet that created the object. Object methods are executed in the owner context of the object. The context decides whether access to other objects is allowed or not. Contexts are isolated by the firewall, denying access for a method being executed in one context, to methods or attributes of objects belonging to another context. [10]

The standard ISO/IEC 7816-4 defines the communication principles between smart cards and their designated readers. These devices communicate through APDUs, or Application Protocol Data Units. Similar to how a web server would reply with a HTTP response to a HTTP request from a web browser, a smart card will wait until receiving a Command APDU and reply to it with a Response APDU, as depicted in Figure 2. These APDUs may carry up to 255 bytes of data, and if more data is required to be sent, several Command APDUs will be sent in half-duplex processes. [5]

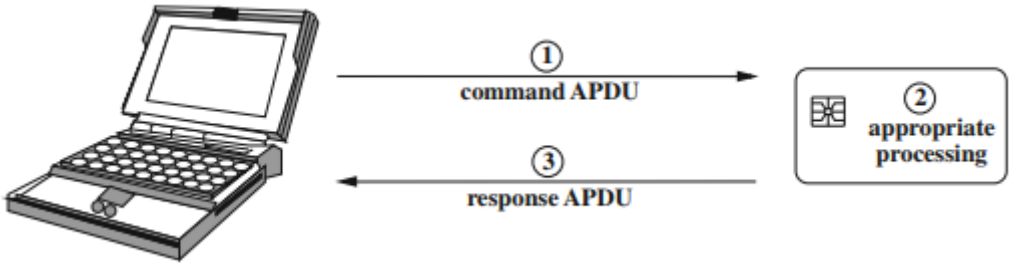

Figure 2. Smart card communication model [5].

Command APDUs have the structure shown in Figure 3. The command header is required and consists of four 1-byte fields: *CLA*, *INS*, *P1* and *P2*. Gomes et al. [12] describe: "The *CLA* field identifies a class of command and response APDU. The *INS* field corresponds to a[n] instruction inside a *CLA*. These instructions can be, for instance, method calls. *P1* and *P2* are parameters that can be used to supply some additional information to the *INS* instruction. The command body is required only for extra data sending or receiving. The data is sent in the *DATA* field and has its length specified in *Lc*. If a response with data is expected, its length has to be informed in the *Le* field. The *Lc* and Le fields have 1 byte of length".

Command APDU						
Header (required)				Body (optional)		
CLA	INS	P1	P2	Lc	Data	Le

Figure 3. Command APDU structure [12].

Response APDUs have a simpler structure, shown in Figure 4. “The response is formed by the optional body, that contains the Data field, with the data returned to the host application and the trailer, which contains the fields SW1 and SW2 that together inform the command [sender of the] APDU processing status” [12].

Response APDU		
Body (optional)	Trailer (required)	
Data	SW1	SW2

Figure 4. Response APDU structure [12].

APDU messages can be transmitted with two different transmission-level Transmission Protocol Data Units (TPDU), T=0 and T=1, which are used to support APDU protocol transmission between a chip reader and a chip itself. APDU protocol is used between the chip application and the chip reader. T=1 is a block-oriented protocol which enables blocks or grouped collections of data to be transferred. These data groups are transferred as a whole between a chip and a reader. The theoretical maximum length of T=1 protocol grouped collections for Command APDUs is 65535 and for Response APDUs is 65536 bytes. The practical maximum length depends on the chip platform that is used for an application. T=0 is a byte-oriented protocol, meaning that the minimum data that can be transferred has a length of one byte. The maximum length of data structure that can be transferred with this protocol for Command APDUs is 255 and for Response APDUs is 256 bytes. [13]

The default transmission protocol used for the new ID-cards is T=1, and whenever Card Content management commands sent to the ID-card are described, the T=1 protocol is used.

2.1.2 Development

The subset of Java that is used to program applets that run on Java Cards is also called "Java Card". Oracle provides a Java Card Development Kit for applet developers. This kit includes an SDK that can be imported to any preferred IDE to ease writing code. The subset is designed for running in constrained environments like smart cards. Development is carried out similarly to regular Java in .java files and compiled to .class files. Afterwards, the compiled file is passed through byte-code conversion, creating a CAP file, and this can then be sent with Command APDUs to a receiving card (Figure 5).

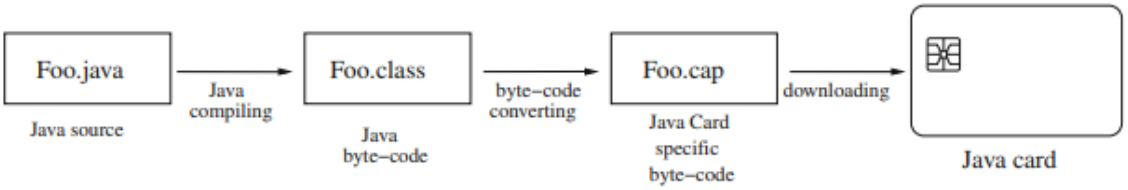

Figure 5. Java Card applet development and deployment [5].

Once an SDK version is imported to a chosen IDE (an *api.jar* file, or *api-classic.jar* if using the Classic Edition, included in the Development Kit), applet development can commence. The version of the SDK should be considered before starting development because the cards that are planned to be used must support the same version of Java Card. An applet is a Java Card class that extends the *javacard.framework.Applet* abstract class. This class "must be extended by any applet that is intended to be loaded onto, installed into and executed on a Java Card technology-compliant smart card" [14]. The class makes the implementation of *install* and *process* methods mandatory. The *install* method will be called by the JCRE first to create an instance of the Applet subclass through its constructor, and subsequently calling the *register* method will register it in the JCRE. The *process* method will be called by the JCRE to process an incoming APDU command. A small example Applet modified from Gomes et al. [12] is given in Figure 6.

```
1.  import javacard.framework.*;
2.
3.  public class Transport extends Applet {
4.      //The current amount of credit
5.      private short balance;
6.      //The INS code of addCredit method
7.      public static final byte ADD_CREDIT = 0x01;
8.
9.      private Transport(byte[] bArray, short bOffset, byte bLength) {
10.         balance = 0;
11.     }
12.
13.     /**
14.      * Invoked by the JCRE, install is the applet entry point. It
15.      * creates an applet instance and registers it in the
16.      * JCRE through the invocation of the applet constructor method.
17.      */
18.     public static void install(byte[] bArray, short bOffset, byte bLength) {
19.         Transport applet = new Transport(bArray, bOffset, bLength);
20.         applet.register();
21.     }
22.
23.     /**
24.      * process receives an APDU object and selects the instruction specified
25.      * in its INS field. Called by the JCRE to process incoming APDU commands.
26.      * An applet is expected to perform the action requested
27.      * and return response data if any to the terminal.
28.      */
29.     public void process(APDU apdu) {
30.         byte[] buffer = apdu.getBuffer();
31.         switch (buffer[ISO7816.OFFSET_INS]) {
32.             case ADD_CREDIT:
33.                 addCredit(apdu);
34.         }
35.         /**
36.          * The method addCredit adds some data to the balance attribute.
37.          */
38.         public final void addCredit (APDU apdu){
39.             byte[] buffer = apdu.getBuffer();
40.             byte bytesRead = (byte) apdu.setIncomingAndReceive();
41.             balance = (short) (balance + buffer[ISO7816.OFFSET_CDATA]);
42.         }
43.     }
```

Figure 6. Java Card applet example modified from [12].

Different tools are available for use once an application is ready to be compiled and converted to a CAP file. For example, ant-javacard[1] is a popular pre-packaged Ant[2] task for building Java Card CAP files. After adding Ant to a project, and adding an Ant buildfile *build.xml* to the project, for example in the root directory, a configuration similar

[1] https://github.com/martinpaljak/ant-javacard

[2] https://ant.apache.org/

to the one displayed in Figure 7 should be written to it. The most important tags here are the <*taskdef* /> and <*javacard* /> tags, one defining the custom *javacard* task, the other defining the properties of the task.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <project basedir="." default="applet" name="MyApplet build">
3.    <property name="JC304" value="ext/jc304_kit"/>
4.
5.    <path id="classpath">
6.      <fileset dir="${JC304}" includes="**/*.jar"/>
7.    </path>
8.
9.   <taskdef name="javacard" classname="pro.javacard.ant.JavaCard"
     classpath="ext/ant-javacard.jar"/>
10.   <target name="applet">
11.     <javacard jckit="${JC304}">
12.       <cap output="MyApplet.cap" sources="src/ee/gj" version="1.0">
13.         <applet class="ee.gj.javacard.MyApplet" aid="0102030405060708"/>
14.       </cap>
15.     </javacard>
16.   </target>
17. </project>
```

Figure 7. ant-javacard *build.xml* configuration example.

Once a valid configuration is written, running the command *ant* from the terminal on the project's root folder should produce a valid CAP file from the project sources to the project root folder.

2.2 GlobalPlatform

The following sub-sections focus on the widely recognised smart card specification GlobalPlatform, its architecture (Section 2.2.1) and it's content management possibilities, including the underlying DM functionality (Section 2.2.2).

2.2.1 Architecture

As stated in Section 1.1.3, GlobalPlatform defines a global specification for smart cards, providing a common security and card management architecture. The GlobalPlatform card architecture is depicted in Figure 8.

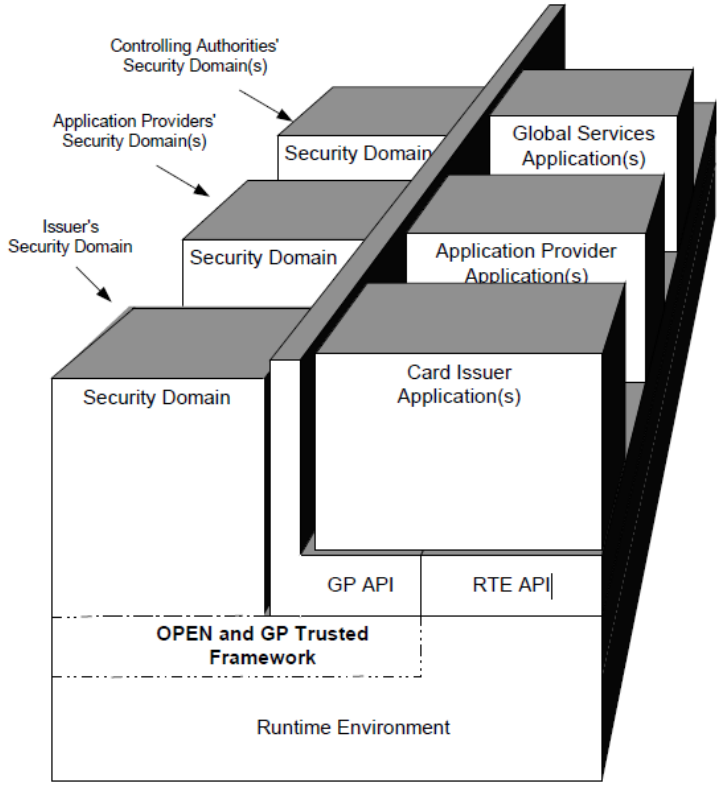

Figure 8. GlobalPlatform card architecture [8].

For the sake of avoiding duplication, not every part of the architecture shall be explained in great detail. For a complete overview, the author refers the reader to the GlobalPlatform specification Chapter 3 [8]. Concepts most relevant to the work in this thesis are described, based on [8], as follows:

- **Security Domain**. "On-card entity providing support for the control, security, and communication requirements of an off-card entity (e.g. the Card Issuer [or] an Application Provider […])" [8]. These domains allow their owners to control included applications without compromising security, i.e. keys or architecture. Technically speaking, security domains are actually on-card applications with a different set of privileges.

- **Card Issuer**. The Issuer Security Domain (ISD) is the primary Security Domain, and in the case of the Estonian ID-card, belongs to the Estonian authorities, i.e. the Card Issuer. The ISD is also the first application installed on a card.

- **Application Provider**. The Application Provider Security Domain (or Supplementary Security Domain) is a secured environment for third-party

developers and maintainers, i.e. Application Providers not strictly related to the Card Issuer, to download, install and maintain their applications.

- **OPEN**. The GlobalPlatform Environment (OPEN) provides "an API to applications, command dispatch, Application selection […] and Card Content management" [8]. It performs application code loading, memory management, manages loaded application installation and is responsible for enforcing security principles.

- **Runtime environment**. "The GlobalPlatform is intended to run on top of any secure, multi-application card runtime environment. This runtime environment is responsible for providing a hardware-neutral API for applications as well as a secure storage and execution space for applications to ensure that each application's code and data can remain separate and secure from other applications on the card. The card's runtime environment is also responsible for providing communication services between the card and off-card entities" [8]. In the case of the Estonian ID-card, the runtime environment is Java Card Runtime Environment.

2.2.2 Delegated Management

"Card Content management on a GlobalPlatform card is the capability for the loading, installation, extradition, registry update and removal of Card Content" [8]. As a Card Issuer may not have access to a card throughout its lifecycle or want to manage all Card Content changes, the design of GlobalPlatform takes into account the necessity of delegating Card Content management to an Application Provider with or without authorization.

Before any changes to a cards' content can be made, OPEN requires that the card life cycle states are not CARD_LOCKED or TERMINATED. In addition, the Security Domains (excluding the ISD) that are targeted for content management must have a Life Cycle State of PERSONALIZED. If the targeted Security Domain has the DM privilege, a specific Token is required in any content management request.

In short, DM is defined as "Pre-authorized Card Content changes performed by an approved Application Provider" [8]. DM, in essence, is a Privilege assigned to Security

Domains, which allows for approved Application Providers to manage their content within these Supplementary Security Domains. Security Domains with DM Privilege require a Token to be sent with Card Content management commands. These Tokens are signatures of one or more DM functions (loading, installing, extraditing and deleting) generated by the Card Issuer, and are used to provide the Card Issuer control over these changes, and to prove that an Application Provider has been authorised to perform these changes. [8]

For example, a Load Token for DM is a signature authorising the transmission of application code to the card and allows the verification of a load request. Figure 9 shows how a Load Token is generated.

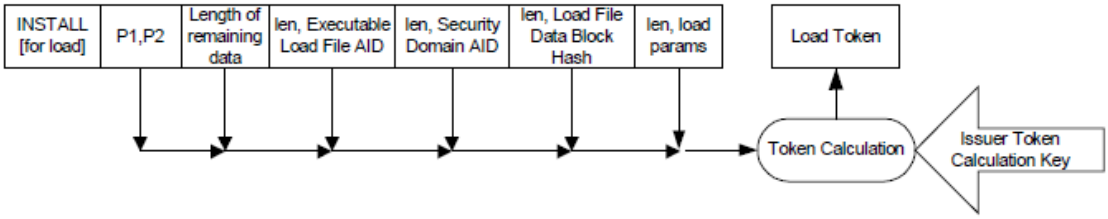

Figure 9. Load Token calculation [8].

If a Security Domain performing the load has the DM privilege, the Security Domain shall require a Token to be present. The ISD shall then verify this Token in order to authorise the operation. For this verification to succeed, the Card Issuer has to have installed a specific set of keys onto the ISD, which are verified against a set of keys used to construct a Token. In order to satisfy this requirement, the generated Load Token must be appended to the end of the command data field, as shown in Table 1. [8]

Table 1. INSTALL [for load] Command Data Field [8].

Name	Length	Value	Presence
Length of Load File AID	1	‘05’ - ‘10’	Mandatory
Load File AID	5-16	‘xxxx...’	Mandatory
Length of Security Domain AID	1	‘00’ or ‘05’ - ‘10’	Mandatory
Security Domain AID	0 or 5-16	‘xxxx...’	Conditional
Length of Load File Data Block Hash	1	‘00’ - ‘7F’	Mandatory
Load File Data Block Hash	0-n	‘xxxx...’ – see section C.2	Conditional
Length of Load Parameters field	1-3	‘00’ - ‘80’, or ‘81 80’ - ‘81 FF’, or ‘82 01 00’ - ‘82 FF FF’	Mandatory
Load Parameters field	0-n	‘xxxx...’ – see section 11.5.2.3.7	Conditional
Length of Load Token	1-3	‘00’ - ‘80’, or ‘81 80’ - ‘81 FF’, or ‘82 01 00’ - ‘82 FF FF’	Mandatory
Load Token	0-n	'xxxx...' – see section C.4.1	Conditional

When comparing Figure 9 and Table 1, it is clear that the set of data components going into the Token generation is identical to the data being sent in command data fields, excluding the reference control parameters P1 and P2, length of the following data fields (the exact length of data in the INSTALL command, prior to a Token being added) and the Token itself. The same applies to all other Card Content management operations, i.e. DM Tokens are signatures of Card Content management command data fields.

2.3 Contactless Interface and Vulnerabilities

This thesis is mostly interested in the contactless interface introduced with the new Estonian ID-cards. GlobalPlatform has provided an extension of the general specification, “Contactless Services” [15], that defines mechanisms, parameters and interfaces to set up and maintain the configuration of applications usable on a contactless interface. These applications are referred to as “Contactless Applications”.

The new Estonian ID-card is a dual-interface card, meaning that it has both the contact and contactless interfaces. Part of the card’s storage is shared through both interfaces, but some, more privacy-sensitive parts are only accessible through the contact interface. When communicating through the contact interface, a card gets its power straight from a

reader. On the contactless interface, however, Messerges et al. [16] explain that electricity "is provided via a radio frequency (RF) signal emitted from the smart-card reader. An antenna on a contactless card inductively couples to this RF signal and a circuit converts the RF power to power that can be used by the smart card's microprocessor".

The contactless interface provides many advantages over the contact interface, for example, ease of use, convenience and compatibility with NFC-enabled smartphones. Avoine et al. [17] also describe the advantages of not requiring a line of sight for reading, no need for a battery, being capable of heavy cryptographic primitive execution and relative cheapness. But with this convenience comes an array of issues, mostly regarding the security of the smart card. As such, it is important to inform the reader of the existence of some of these issues.

For example, Chothia et al. [18] demonstrate the possibility of tracing the movements of a particular RFID-enabled passport, and consequently the holder, without having to break the passport's cryptographic key. An attacker can simply record one session between a passport's RFID tag and a legitimate reader by eavesdropping in on the conversation. By replaying a particular message, a single passport can be distinguished based on the response. In [19], Chothia et al. present the "leakiEst" tool, allowing meaningful estimation of information leakage. This tool is used to test fixes for the traceability attack, and the authors found that "modifying the protocol to continue processing a message even when the MAC check fails, and only reject it at the end of the protocol, leakiEst indicates that it is free from leaks" [19].

Sportiello [20] uses Basic Access Control (BAC), which prevents unauthorized reads of the chip's content and protects contactless communication with a legitimate reader, to carry out a side channel analysis for documents with a contactless chip. Specific timing analysis is done during BAC operations, revealing that the results can be exploited to retrieve the chip's content. As newer smart cards, including the ID-card, use Password Authenticated Connection Establishment (PACE) instead of BAC, this attack cannot be attempted on the new Estonian ID-cards.

Gkaniatsou et al. [21] present a proof-of-concept system, REPROVE, for analysing the APDU protocol of a smart card, regardless of the protocol's implementation. REPROVE was used to extract models from smart cards and successfully reverse-engineer

proprietary implementations of the APDU protocol, finding many insecurities in tested cards, for example, violation of PKCS#11 specification requirements in regards to unique session handles, leaky tokens, treating sensitive data as public, and so on. Bozzato et al. [22] exploit proprietary implementation weaknesses to illustrate attacks on PKCS#11 devices through the APDU protocol, present a threat model and security analysis, complementing the work of Gkaniatsou et al.

As the security implementations on the ID-card are proprietary, depending on the card manufacturer and their agreements with the card issuer, the status quo cannot be deduced from previous publications. It would be a good idea to map available tools and methods to help discover threats that could affect the new ID-cards. Collecting these tools or proving the safety of the software or hardware implementations of security measures on the ID-card is beyond the scope of the current thesis.

3 Delegated Management Support in GlobalPlatformPro

The following chapter deals with the process of adding Delegated Management support to the open-source tool GlobalPlatformPro, to enable content management on cards that require Tokens to be present on targeted Security Domains.

3.1 Introduction

In order to satisfy the requirement of having Tokens sent with content management commands targeted at Security Domains with DM privilege, Tokens need to be generated dynamically from the rest of the command's data fields. This functionality is currently missing from the open-source GlobalPlatformPro tool. The purpose of this chapter is to answer the research question **RQ-1: How to add Delegated Management support to GlobalPlatformPro for managing content on the new ID-cards?**

3.2 Configuration and Setup

In order to test and verify the functionality of any existing or added code in GlobalPlatformPro, Java 8, a USB smart card reader, supporting drivers and a Java Card smart cards similarly configured to the new Estonian ID-cards are required.

The smart card reader used in development and testing is the PC/SC-compliant ACR1251[1] (S/N ACR1251U-A1), produced by Advanced Card Systems Ltd. This reader's predecessors have been successfully used in the literature, for example, Chothia et al. [18] used it in devising a traceability attack method against RFID-enabled [(Radio-Frequency Identifier)] passports, saying that "this is one of the cheapest (~$50) RFID readers on the market and […] using such a reader underlines the fact that our attack does not need specialist hardware" [18]. The same can be said about the current reader – it

[1] https://www.acs.com.hk/en/products/218/acr1251u-usb-nfc-reader-ii/; http://www.acr1251.com/

costs around $50 and works adequately well for managing a smart card's contents, proving that specialist hardware is not required. Even though this work uses a contactless reader, the ID-card also has a more common contact interface, allowing cheaper contact readers to be used as well.

Development and testing are conducted on a Linux distribution. As PC/SC[1] is the standard for integrating personal computers with smart cards and their readers, and Windows operating systems contain the reference implementation out-of-the-box, Windows users should mostly be without any additional software. Installing the ID-software from RIA (https://www.id.ee) is strongly recommended, however. Linux distributions do not come with bundled PC/SC drivers, so an open-source solution such as PCSC-Lite[2] is recommended - RIA's provided Linux bundle includes it as well.

The smart cards used in testing are configured and personalised similarly to the ID-cards issued to the Estonian citizens. These test cards were kindly provided by RIA, in the interest of having the goals of this thesis successfully delivered. The test card with the previously described USB contactless reader is shown in Figure 10.

Figure 10. Test ID-Card and ACR1251U contactless USB reader.

[1] https://www.pcscworkgroup.com/

[2] https://pcsclite.apdu.fr/

3.3 Token Generation Support

Observing the second to last line of Table 1, it is clear that DM Token information has to be appended to the end of every sent command. The latest released version of GlobalPlatformPro at the time of writing this work, release 19.01.22, only appends the zero-length of a Token to all sent commands. This is not enough for the new Estonian ID-cards, as the Supplementary Security Domain pre-installed on the cards has DM privilege. In order for GlobalPlatformPro to support content management functionalities on new ID-cards, DM Token generation support must be added to the tool.

Privileges of existing Security Domains can be read through running the CLI with parameter *l* or *list*, which lists all applets installed on the card, as shown in Figure 11. This command should be used cautiously, however, since the default keys of "40…4F" pre-defined in GlobalPlatformPro are not suitable for either test or production ID-cards, and could lock the card completely if misused too many times. A custom master key should be specified with parameter *key* and key value, followed by parameter *kdf3*, which specifies that secret keys should be derived from the provided key, following the Secure Channel Protocol 3 (SCP03). Also, it should be noted, that Card Content management commands, including *list*, will target the ISD by default, if not stated otherwise with a *sdaid* parameter value. This means that the value assigned to parameter *key* must be assigned to the ISD, if no other Domain's AID value is provided.

```
ISD: A0000001510000 (SECURED)
     Privs:   SecurityDomain, CardLock, CardTerminate, CardReset, CVMManagement,
 TrustedPath, AuthorizedManagement, TokenVerification, GlobalDelete, GlobalLock,
 GlobalRegistry, FinalApplication, ReceiptGeneration

DOM: D233000000444F4D (PERSONALIZED)
     Privs:   SecurityDomain, DelegatedManagement, CardLock, TrustedPath
```

Figure 11. Segment of listed Security Domains from "*java -jar gp.jar -list -key <ISD key> -kdf3*" output on a test ID-card.

Before attempting to implement Token generation, a better understanding was required of what was already implemented according to the GlobalPlatform specification [8], and how. As the code was hard to grasp to a newcomer (further explained in Section 7.1), attempts at changing existing code were made to improve readability and avoid possible faults due to misconception. For example, a 45-line section of code implementing the *load* command's functionality in class *GPTool* was extracted to a separate method, and

the *install* command's partly duplicated code was removed to use this new method, which also added missing Data Authentication Pattern (DAP) verification functionality to the *install* command.

In addition, a class for encapsulating all details related to DAP was created, named *DAPProperties*, displayed in Appendix 1 – GlobalPlatformPro DAPProperties.java. This code was previously spread around in the code implementing the *install* and *load* methods. An instance of this class is created at the beginning of the previously described method, named *calculateDapPropertiesAndLoadCap*, which as the name suggests, calculates properties related to DAP, and continues to attempt loading a CLI-targeted CAP file.

The centrepiece of the DM functionality is the addition of the class *DMTokenGenerator* to GlobalPlatformPro, displayed in Appendix 2 – GlobalPlatformPro DMTokenGenerator.java. An instance of this class is created by class *GlobalPlatform* at the beginning of calling the CLI and used for any following content management command if the parameter *token-key* with a path to an RSA private key is used. If no *token-key* parameter is passed, the zero-length parameter of a token is appended, as it was before.

The last two lines of Table 1 indicate that Token data is always required – if no Token is present, the zero-length of the Token must be written to the command data. This is particularly important to keep in mind in the case of the "Table 11-22: Delete [card content] Command Data Field" described in [8], where the required length of each data component is not separately pointed out. This inconsistency in the presentation of required data elements caused some confusion at the beginning of implementing the Token generation functionality, but was later corrected with the section of code in Figure 12.

```
1.  if (apdu.getINS() == INS_DELETE || apdu.getINS() == (INS_DELETE & 255)) {
2.  // See GP 2.3.1 Table 11-23
3.  logger.debug("Adding tag 0x9E before Delete Token");
4.  newData.write(0x9E);
5.  }
```

Figure 12. Delete Token distinction tag in class *DMTokenGenerator*.

Among other required changes, an additional option for the Load File Data Block Hash calculation had to be added. The Load File Data Block is a "part of the Load File that contains one or more application(s) or libraries and support information for the application(s)" [8] and its hash provides its integrity. Although [8] describes the hash as "a SHA-1 digest of the Load File Data Block", the test card required the hash to be calculated with the more secure SHA-256, and the same can be assumed for the production card. Before the *sha256* option was only respected in hash calculation if DAP was required. Changes were made to respect this option even if DAP is not required, and to include a hash in command data if a Token is being appended to the data, as per specification: "The Load File Data Block Hash is mandatory when a Token or DAP Block is present in a Load File, and is optional otherwise" [8].

At the end of constructing each Card Content management command's data, the constructed *CommandAPDU* is passed through a *transmitDM* method, which modifies the data with the *DMTokenGenerator.applyToken* method. This extracts the existing APDU data, digests it with SHA-1, signs the digest with the RSA algorithm and appends the resulting Token to the APDU data similarly to the process depicted in Figure 9. Finally, the modified APDU data is transmitted to a connected card.

The full list of changes introduced to GlobalPlatformPro with DM Token generation support is outlined in the GitHub pull request "Files changed" comparison at https://github.com/martinpaljak/GlobalPlatformPro/pull/155/files.

3.4 Discussion

The main objective of Chapter 3 is to add Token generation support for generating Load, Install, Make Selectable, Extradition and Delete Tokens. This is most suitable for the use-case the author was handed, i.e. having to generate tokens by using an issuer-provided RSA private key for DM, where the related public key is previously installed on the ISD. However, this does not cover the use-case where the card issuer will not provide a DM RSA private key to third parties, but would rather validate CAP files of third parties, and send the calculated Token data back to the third party if the CAP was deemed safe and valid. For this, an additional CLI option should be added, which can be used to authorise a single specific content management command on a single specific CAP file.

In addition, GlobalPlatform also specifies Tokens for Registry Update and a combined Load, Install and Make Selectable Token, which this thesis does not cover. The latter of these would make the most sense to apply in the use-case where third parties do not have possession of any keys, and RIA would otherwise have to generate a set of three Tokens for every new request – a Load Token, an Install Token and a Make Selectable Token.

3.5 Conclusion

Chapter 3 demonstrates the implementation of DM Token generation support in GlobalPlaftformPro. The currently supported functionality of the GlobalPlatform specification is determined in order to clarify needed improvements. Duplicated or overly complex sections of code are partly refactored and simplified to reduce the risk of introducing errors due to misconception. Afterwards, data constructed for content management is used to create DM Tokens, authorising content management for Load, Install, Make Selectable, Extradition and Delete commands on Security Domains with DM privilege.

4 Content Management on New Estonian ID-cards

The following chapter deals with the process of loading and installing an existing Java Card applet to test ID-cards, using the enhancements to GlobalPlatformPro that are described in the previous section.

4.1 Introduction

Successful Card Content management on GlobalPlatform smart cards requires a specific set of commands, parameters and data to be sent to the card. In the case of targeting Security Domains with DM privilege, the set of required parameters and data is lengthened even more. As these requirements are most probably unknown to third parties, as was the case with the author of the thesis initially, these must be outlined and explained for successfully loading, installing and deleting applets on the new Estonian ID-cards. The purpose of this chapter is to answer the research question **RQ-2: How can third parties manage the contents of the new ID-cards?**

4.2 Example Applet

As the successful installation of a simple "Hello World" application would provide a very basic proof-of-concept, it would not demonstrate the potential of this new opportunity to extend the primary functionality of the ID-cards very well. Creating a new applet with potential for wide use is beyond the scope of this thesis, thus an existing practical application with a sensible use-case should be chosen to showcase this new opportunity.

The chosen applet is from a project called openjavacard-ndef[1]. This project implements NDEF tags for Java Card smart cards, and the applet only implements the exact minimum of APDU commands that the NDEF specification prescribes. It's intended as a reusable

[1] https://github.com/OpenJavaCard/openjavacard-ndef

library covering most use-cases for NDEF, with support for emulating simple NDEF memory tags, as well as dynamic tags. The project includes three different base variants of NDEF tag implementations [23]:

1. TINY – a minimal read-only tag, initialised by providing data during installation. The load file size is less than 1 kB. Recommended for serving static content, such text or a URL.

2. STUB – requires a secondary service (applet) to be used to generate contents. Used for creating dynamic NDEF tags while keeping applet under a proper unique AID. Writing is not supported. The load file size is slightly above 1 kB.

3. FULL – a writable and configurable NDEF tag. Can be configured during installation and at build time with optional advanced features, such as media-independent access control and write-once support. The Load file size varies from 1 to 2 kB depending on selected features.

The project recommends starting with the FULL variant since it contains every available feature except for the external service feature of the STUB variant. Thus, for testing on the test ID-cards, the FULL variant is chosen.

Examining the code of the FULL variant applet, the standard pattern of Java Card applets can be recognised, as described in Section 2.1.2 and exemplified in Figure 6. The class *NdefApplet* extends the *javacard.framework.Applet* abstract class and implements the mandatory *install* and *process* methods.

4.3 Loading, Installation, Uninstallation and Deletion

As described in Section 343.3 and shown in Figure 11, the list of existing Security Domains on a test ID-card can be displayed with the command "*java -jar gp.jar -list -key <ISD key> -kdf3*". This will output, among other information, the AID of the Supplementary Security Domain, D233000000444F4D.

Next, after obtaining an AID for the Supplementary Security Domain, either from the CLI or from RIA, an RSA private key must be acquired to include with commands targeted at the DM privileged Domain. As described in Section 3.3, the private key included with parameter *token-key* is used to sign the data being sent for content management, creating

a DM Token. The public key corresponding to this private key is pre-installed in the ISD, and the private key should be provided by RIA for authorising all DM on a specific Domain. During testing, this scenario seems acceptable, but for production cards, other more suitable and secure scenarios are discussed in Chapter 5.

The required command to load an application to the test ID-card is "*java -jar gp.jar -sdaid <SSD AID> -key <SSD key> -kdf3 -install <CAP file> -token-key <DM key> -sha256*". In the background, this will send both "INSTALL [for load]" and "INSTALL [for install]" instructions, transmitting application code to the card and making it selectable and executable. As described in Section 3.3, the sha256 option is required for switching the Load File Data Block Hash calculation algorithm to SHA-256. The last three lines that were logged with *debug* and *verbose* modes enabled after successful applet installation are shown in Figure 13.

```
[DEBUG] DMTokenGenerator - Using private key for token generation (SHA1withRSA)
[TRACE] GlobalPlatform - Payload:
A>> T=1 (4+0165) 84E60C00 A5 06A0000005272007A0000005272001 07A000000527200101000
2C90080B8B2786ACA4D47F401B64377C3101E3EC45FBFEC8AE02731F266677F7AF1EEB3921BF0F3A
DABC4B33C2360CEAC8D6009E316B3042AAE634AE2594B6822B091E1D18BE7AB1A7D2294B8D7BCA5A
DFC378EB2E2FCF0C265747D595BECBE241A8E981FB9D84927363D2519B4E6F75A64CD9102B81B7B6
BFE67C7BA8C1FA9901F7864B326F8B4CF3F0743
A<< (0000+2) (1s494ms) 9000
```

Figure 13. Last three lines from a successful execution example of "*java -jar gp.jar -dv -sdaid <SSD AID> -key <SSD key> -kdf3 -install <CAP file> -token-key <DM key> -sha256*" on a test ID-card.

If at first the command will not return a successful response message of "9000", and validating all inputs does not reveal any errors, other so-called preventative measures may be required. The readers' mileage may or may not vary, but during our testing, we discovered that the DM RSA public key that was installed on the ISD was different from the one that was said to be installed. Since we were also provided with the key to access the ISD, we were able to replace the DM public key with the one that was said to be installed on the card, and thus we could configure the card to accept the DM Token that was generated in GlobalPlatformPro with the provided DM RSA private key. Since on production cards this cannot be done, it is up to the Estonian authorities to ensure that the keys are valid before issuing cards to citizens, and to update any invalid keys that may have been issued to citizens.

Different keys installed on the card have specific unique versions. To get the version of the 1024-bit RSA key for DM, running the *info* command on GlobalPlatformPro was

required. This will list, among other information, the details of keys installed on the connected card, as shown in Figure 14.

```
Card Capabilities:
Version:   1 (0x01) ID:   1 (0x01) type: AES  length:  32 (AES-256)
Version:   1 (0x01) ID:   2 (0x02) type: AES  length:  32 (AES-256)
Version:   1 (0x01) ID:   3 (0x03) type: AES  length:  32 (AES-256)
Version: 112 (0x70) ID:   1 (0x01) type: RSA  length: 128 (RSA-1024 public)
Version: 113 (0x71) ID:   1 (0x01) type: DES3 length:  16
```

Figure 14. List of installed keys from "java -jar gp.jar -info" output on a test ID-card.

The key version 0x70 can be deduced from the output in the *info* command. Rewriting the DM key is possible with the command "*java -jar gp.jar -key <ISD key> -kdf3 -put-key <DM key> -new-keyver 0x70*".

If after successful installation the applet needs to be removed or replaced at some point, this can fairly easily be done in two ways. For complete removal, using the *uninstall* command with the installed CAP file will send a "DELETE" instruction to the card, deleting the AID extracted from the CAP file. Applet deletion is possible with the command "*java -jar gp.jar -sdaid <SSD AID> -key <SSD key> -kdf3 -uninstall <CAP file> -token-key <DM key> -sha256*". For replacing, using the *force* parameter with the *install* command will send the same "DELETE" instruction if the supplied CAP file (more specifically the same AID defined in the CAP file) exists on the domain already, followed by the regular logic of the *install* command.

4.4 Example Applet Testing

After successful installation of the FULL version applet from openjavacard-ndef, and without appending any parameters through the *params* CLI option at installation, the applet should be initialised with an empty tag. To test this, switching on the NFC reader on a smartphone and placing the smart card on the reader, usually centred on the back of the phone, the phone should display a popup, indicating a successful read and showing the contents of the NDEF tag that was read, as shown in Figure 15.

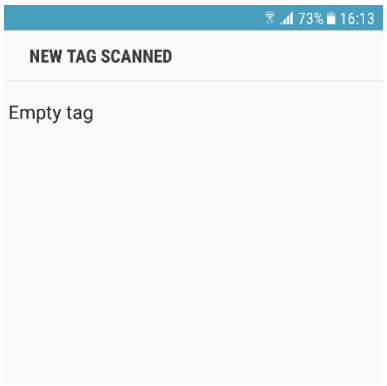

Figure 15. Reading an NDEF tag from the test ID-card.

If, however, no popup is shown, some changes might be required for the buildfile of the applets. As of the 24th of November, 2018, the AIDs for the applets (TINY, STUB and FULL) were changed[1] from the official AID registered to NFC Forum (package D276000085, applet D2760000850101) to unique values. This seems to have hindered the applet's functionality, as phones do not seem to be able to recognise it anymore. After changing the AIDs back to the static value that is registered to NFC Forum, phones should be able to recognise the card as an NDEF tag. As of writing this thesis, the values for the AIDs are unique, but this could be changed in the project in the future.

To rewrite contents on the applet (or any other NDEF tag), several possibilities are available. To test the application in this thesis, a Samsung Galaxy S6 (SM-G920F) running on Android 7.0 is used. Thus, the easiest way to configure the applet would be to download an application from the Google Play Store that's specifically meant for writing data to NDEF tags. One of the many available options is NFC Tools[2].

The process of writing a static text value to the applet (or any other NDEF tag) with NFC Tools is depicted in Appendix 3 – Configuring NDEF Tags. Navigating to the "WRITE" tab of the application will reveal two buttons. Tapping "Add a record" will list all available types of records that can be written to a tag. For example, "Text" is for static text and "URL / URI" is for an address that will be automatically opened in the default

[1] https://github.com/OpenJavaCard/openjavacard-ndef/commit/ef212b7

[2] https://play.google.com/store/apps/details?id=com.wakdev.wdnfc

browser of a phone once the tag is read. After choosing “Text”, assigning a value and tapping “OK”, the value will be appended to the list of values to be written to the tag. Tapping the “Write” button above the list will activate the NFC module for writing, and it will wait for a tag to enter it’s reading range. After placing the card on the back of the phone and writing succeeds, a success message is shown. Finally, after removing the card from the back of the phone, exiting out of the application, and placing the card to the back of the phone again, a popup with the written static text should be shown. If multiple values are appended to the list, all of them will be written to the tag and displayed on the previously described popup as a list as well.

4.5 Discussion

The main objective of Chapter 4 is to showcase the added DM functionality of GlobalPlatformPro to successfully install an example applet to the Supplementary Security Domain on the test Estonian ID-card. A fairly simplistic application that emulates an NDEF tag was chosen and successfully installed and demoed. However, this was not the first choice of application to demonstrate for the thesis. Other more complex applets, like CCU2F[1] or hotp_via_ndef[2] were also tried. Both of these applets were installed on the test ID-card without errors, but making them run as intended proved to be difficult.

The first of these is a universal FIDO U2F authenticator applet, providing a strong two-factor authentication solution similar to security key products sold by Yubico[3]. The CCU2F applet required some changes to the code since the project relied on a proprietary API for cards produced by NXP. Specifically, the proprietary *KeyAgreementX.ALG_EC_SVDP_DH_PLAIN_XY* should be swapped to a generic Java Card 3.0.5 API *KeyAgreement.ALG_EC_SVDP_DH_PLAIN_XY*. However, since the ID-card is version 3.0.4, and access to the proprietary API was not available, swapping to *KeyAgreement.ALG_EC_SVDP_DH_PLAIN* was attempted. This allowed installing the

[1] https://github.com/tsenger/CCU2F

[2] https://github.com/petrs/hotp_via_ndef

[3] https://www.yubico.com/

application, but for it to completely work, a conceptually different algorithm cannot be used, as described by [24].

The second uses the demonstrated NDEF tag project as a base, and builds an HMAC-based One Time Password generator into it, delivering a unique one-time password to the end of a URL saved on the applet. Though this application does not use any proprietary API and should work on cards as old as version 2.2.2, testing the application did not yield expected results. Saving and retrieving a URL from the tag was successful, but the HMAC-based password was never generated to the end of the URL, but the internal counter for calculation of the expected password was.

Thus, the showcased NDEF applet is simple enough and perfectly showcases the contactless abilities of the new ID-card, but other more interesting applets are also available, that better leverage the cryptographic abilities of Java Cards. Installing these on the ID-card would potentially spark even more interest in the card's capabilities of including third-party applets.

4.6 Conclusion

Chapter 4 illustrates installing an application to the Supplementary Security Domain of the test ID-card. An existing applet that implements a generic NDEF tag was chosen and described. Next, the steps required for successfully installing an applet to the card were listed. Using the added DM functionality to GlobalPlatformPro, the chosen applet was installed on the test ID-card's Supplementary Security Domain. Finally, configuring the static value on the tag applet and returning it to an NFC-enabled smartphone on request was demoed.

5 Applet Development and Management Suggestions

The following chapter contains some suggestions to the Estonian authorities in regards to guiding third parties in develop applets for the new ID-cards. Managing access and privileges of third parties in the deployment of applets is also discussed, after hypothesising possible scenarios for applet installation on live ID-cards. In addition, possible vulnerabilities and countermeasures that could be used in the appropriate applet infrastructure are listed.

5.1 Introduction

At the time of writing the current thesis, the Estonian authorities have not released any documentation on how third parties should develop their applets, what are the steps in the full lifecycle of third-party applets on ID-cards, what are the rights of third parties in managing card contents, etc. Knowing the answers to these questions would minimise the need for asking them from RIA and accelerate the growth of useful smart card applets for the new Estonian ID-card. In addition, the amount and potential severity of threats that allowing third-party applets on the Estonian ID-card would introduce is currently not completely clear. The purpose of this chapter is to make suggestions on what and how the Estonian authorities should disclose to third parties, and what potential threats should be eliminated before allowing the deployment of applets on the ID-cards, thus answering the research question **RQ-3: How should third parties be allowed to develop and prototype applets on the new ID-cards?**

5.2 Aiding Application Providers

In order to aid the development of third-party applets for the new Estonian ID-card, it would be useful to have related information publicly available. This information could be, for example, card capabilities, supported cryptographic algorithms and Java Card API compliance. Information on and availability to the proprietary API is also crucial.

As a card's functionality is limited by the manufacturer in terms of Java Card API compliance and supported algorithms, it is their task to provide functionality through the proprietary API, if demanded by the card issuer, in our case the Estonian authorities. Having this information publicly available, and sharing access to proprietary APIs is a prerequisite for allowing third parties to engage with the ID-card meaningfully. If access is not granted, the full potential of the ID-card is just not used.

The technical description of the new ID-card [7] encloses top-level details of the card, the PKI application, and describes entities interacting with the card. The document currently describes three entities:

1. Cardholder – the natural person to whom the authentication key belongs and to whom the usage is reserved. Authentication to the document is only possible with knowing the PIN1 code. In Estonian use-cases, the Cardholder, Signatory and Administrator Cardholder are the same natural person, to whom PIN1, PIN2 and PUK are given.

2. Signatory – the natural person to whom the signature key belongs, and to whom the usage is reserved. Authentication to the document is only possible with knowing the PIN2 code.

3. Administrator – an entity managing contents of the card, without the right to use credentials. Supervision of an administrator allows performing key generation, key import, key export, PIN personalisation and resetting PIN retry counter. In Estonian documents, the Cardholder and Police are administrative entities with different privileges granted.

The three (or four, including the Police) described entities do not include Application Providers, as defined by GlobalPlatform and described in Section 2.1.1. From the PKI application point of view an Application Provider is not connected as such, but since the document should describe entities that "will interact with Estonia eID Documents, with different privileges and security requirements" [7], the author of this thesis strongly believes that a section dedicated to Application Providers, with possible use-cases including them, should be described.

This thesis attempts installing applets on the Estonian ID-cards through GlobalPlatformPro. End-users, i.e. regular citizens seeking to install something useful on their ID-cards most probably will not be directed to use GlobalPlatformPro or other developer-oriented tools. If installing applets was made a free choice for citizens themselves, it most probably will be through the ID-card software developed by the Estonian authorities. This software currently only deals primarily with the PKI application of the ID-card. Thus, the software should be upgraded to include a subset of the functionality in GlobalPlatformPro.

For applets to be made accessible to citizens, regardless of if the previously described case was to become a reality or if they will be installed without consent some other way, they first need to be approved. Approval of an applet should only be followed by rigorous validation and verification of the applet's code. A subset of possible aspects of the application or the card itself that should be verified is described in Section 5.4. But, knowing exactly what will be validated by the authorities could either be beneficial or harmful. If Application Providers knew exactly what is validated and what an applet can and is allowed do, it is much easier to develop applets that comply to all requirements and restrictions from the start, without wasting precious time on fixing issues and revalidating the new fixed versions later on. Knowing every requirement an applet should comply to, however, also opens the possibility of trying to exploit aspects that are not being validated, at least according to the knowledge disclosed to Application Providers. A malicious applet could get lucky and slip past validation, allowing the deployment of the applet on unsuspecting citizens' ID-cards.

5.3 Managing Access and Privileges

The subject of this section, i.e. possible scenarios for managing access and privileges of Application Providers, has been slightly discussed in this thesis, for example in Section 3.4. In order to have a better understanding of the choices available, other scenarios should be discussed as well, each with their advantages and disadvantages.

But before discussing scenarios of what the privileges of Application Providers could be, the ecosystem should also be envisioned from the citizen's viewpoint. After all, if an Application Provider develops great applets, and citizens cannot access them, discussing the rights of developers would not make sense.

As proposed in the previous section, access to applets could be made available through the ID-card software developed by the Estonian authorities. Developed applets could be sent to RIA, be verified, and uploaded to a database where the ID-card software would have access to. This eliminates the need for updating the software every time a new applet is uploaded or an old one is updated. Verified applets could then be installed on a citizen's connected ID-card.

But questions arise regarding applets already installed on a card. What if an applet receives an update – are citizens asked to update immediately after inserting an affected ID-card, or will it be done for security reasons without any input? What if an applet is deemed malicious after being verified and uploaded to the ecosystem – will it be uninstalled immediately after connecting to the ID-software? What if one has not had access to the software in a long time, and his/her ID-card is filled with outdated and/or malicious applets? Who will be blamed and responsible for covering losses if any protected data gets exported from the card, or the card is bricked by malicious software? What if a card is injected with a malicious piece of code, and using the card breaks a reader and/or infiltrates a whole information system behind the reader, rendering the unsuspecting citizen an accomplice in a crime?

The installation of verified applets could be made a choice for citizens through the ID-card software. It could also be, that free choice is not given to the citizens at all, but rather, applets deemed "useful enough" would just be installed, updated or uninstalled from the ID-card at whatever time the Estonian authorities wish. As most probably the market will not become filled with tens of applets, and all useful applets would fit on a card, this scenario would make the most sense either way. Fitment regarding memory is important, because as stated in 2.1.1, a smart card has very limited resources, severely restricting the number of applets that can be installed.

Having proposed two plausible solutions for applet installation from citizen's viewpoint, the Application Providers' options can be discussed. For live ID-cards, keeping in mind that installation will be carried out through the ID-card software, no access or privileges should really be provided to Application Providers for any kind of content management. The only case in which privileges should be assigned is during prototyping with test ID-cards, as is the case with this thesis.

The first, most loose option, would be to provide all keys required to make content changes, including adding new Security Domains, changing keys, installing and deleting applets without restrictions, etc. The same scenario was handed to the author of this thesis. This would be the most hassle-free for developers and officials alike, adding no extra complexity before submitting an applet up for verification. Before handing keys to an Application Provider, they must be validated, reducing the risk of the developers going rogue. This can be done, for example, by signing a non-disclosure agreement digitally with ones existing ID-card. This scenario would help accelerate prototyping on test ID-cards, as Application Providers would not need to get authorisation for every little change made to a card. It would also be best suited for security testing if researchers show interest in finding possible vulnerabilities that would otherwise be harder to access. It could, however, allow developers with wrong intentions to snoop around a card that is configured similarly to real ID-cards, and test potential vulnerabilities before attempting them on real cards.

The second option would be to provide a sandbox for each Application Provider on a test-card, meaning that RIA would have to install a unique Supplementary Security Domain on a test-card before handing it over to the Application Provider. Next, there are two choices available for RIA – either they supply the Application Provider with a DM private key to freely manage contents in the aforementioned Domain, or not. If a DM private key is not provided, the Application Provider would have to send the applet for review and receive a unique Token authorising each of the content management commands separately. This scenario would require adding the functionality to GlobalPlatformPro (including Token data as-is, not generate it through a DM private key, as this thesis has implemented it). It would also require much more time for prototyping, as it very much depends on the speed of the verification process.

The scenario where a citizen can freely select applets from the ID-software and Application Providers are given a Delegated Management key to develop applets without restrictions during prototyping is depicted in Figure 16.

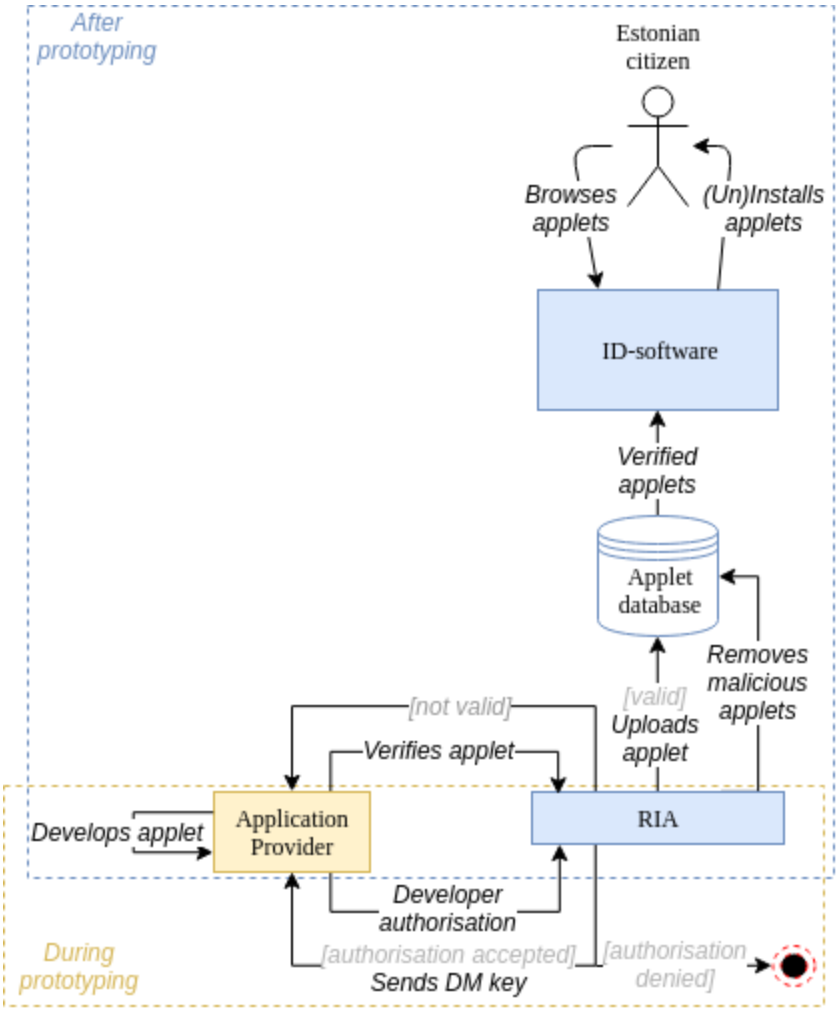

Figure 16. Scenario of applet development and installation on citizen ID-cards.

5.4 Potential Threats

It is unclear whether the new Estonian ID-cards include countermeasures for all possible published attack vectors. And to the knowledge of the author, no publications have been made available that prove this card's security or lack thereof. Proving the security of the ID-card is beyond the scope of this thesis, but instead, the section will attempt to list some of the vulnerabilities of smart cards that could pose a threat to the ID-cards, hopefully motivating the Estonian authorities to include checks for these threats in validation of all Application Provider applets and the card itself. As the main interest of this thesis is the contactless interface on the new Estonian ID-card, and third-party applets that could be deployed on the card, threats related to these two aspects are focused on.

Lackner et al. [25] nicely categorise attacks on Java Card into three large groups: physical attacks, observation attacks and fault attacks. Physical attacks feature removing different physical layers of a Java Card chip by etching and gaining access to different digital or

analogue hardware blocks. Observation attacks are non-invasive attacks exploiting the notion that computation is executed on the card itself – these computations consume a varying amount of power, during a varying amount of time, and targeting various segments of the architecture. This enables drawing conclusions about the card's internals by observing these processes. Fault attacks are provoked by running a chip beyond its technical specifications, disturbing cryptographic operations and behaviour of the Java Card VM.

Section 2.3 describes three publications of threats that could be exploited over the contactless interface. Additionally, Kfir et al. [26] present a relay attack on RFID-enabled smart cards, using a system that first fakes a card to a genuine reader, relays the information to the second part of the system, which then fakes a reader to a genuine card, creating a bidirectional communication channel between a genuine reader and a victim card. The channel provides transparent communication at a range much greater than the nominal system range. This breaks the inherent assumption that contactless communication is as secure as contact communication (the card physically present in a reader's slot is the card that is communicating with the reader and it was presented by the person in front of the reader), since systems based on the ISO-14443 standard are designed to operate over a distance of up to 10 cm – the designed system allows the distance between a reader and a card to be practically unlimited. The authors propose a Faraday-cage product to shield contactless smart cards against attackers or having an on-card input mechanism to physically activate a card. The first one would be advisory to all ID-cards, the second one cannot be done on already issued cards.

Oren et al. [27] deploy a similar solution for relay attacks on RFID-enabled smart cards, using two different antennas and RF front ends, but the authors also see that this range extension scenario can be used with good intentions, for example for handicapped persons sitting in a wheelchair with limited access to readers, or at car parking lots which enable access through reading RFID tags, where the reader is often difficult to reach from the driver's seat.

As described in Section 2.3, Sportiello [20] carries out side-channel analysis through timing analysis during BAC operations, revealing that the results can be exploited to retrieve the chip's content. Markantonakis et al. [28] describe side channel analysis by observing the power consumption of a card over time. As consumption depends on the

type of instruction being executed and the data being manipulated, an attacker could derive key values by simply inspecting the power consumption. What is more, Markantonakis et al. also describe fault analysis from fault injection, for example with abnormal signals, to allow deriving information of keys being used. These two attacks, however, require direct access to the card, and cannot be carried out only through the contactless interface. The authors propose several countermeasures to prevent side channel analysis: constant execution (conduct operations in a constant order irrespective of data), random delays (requiring an attacker to synchronise acquisitions *a posteriori*), randomisation or data whitening (values presented in memory are always masked with a random value) and randomised execution (randomise order of function execution where possible, removing correlation between data being manipulated and an observed side channel).

Iguchi-Cartigny et al. [10] prove the proposal of Hyppönen [29] to exploit static instructions and the CAP-file reference location component, leading to reference spoofing. The authors present a Trojan-like applet to for Java Cards, allowing self-mutable code, and modifying other applications. The attack is based on two hypotheses, which the Estonian ID-card could support, given the right scenario of applet installation: firstly, post-issuance is allowed and necessary credentials are provided, and secondly, there is no bytecode verifier on the card. This applet enables searching and replacing any code fragment in the memory, even if the memory segment is protected by Java Card security mechanisms. This also breaks the segregation properties provided by the firewall. The authors illustrate that the Trojan applet could search any code pattern in system memory and change it, for example replacing a call to *OwnerPIN.check* method by *nop* instructions, rendering PIN codes useless. The authors describe loading time countermeasures (detecting modifications of a CAP file and using on-card type verification to detect ill-formed bytecodes) and runtime countermeasures (detecting illegal memory access).

There could also be more far-fetched approaches. On October 24, 2018, PPA filed a lawsuit against Gemalto AG, since the latter had violated important requirements in the production of the Estonian ID-card. Gemalto AG was the company that for more than 15 years had been producing the ID-card. One of the most important aspects and requirements for security is that the private keys are generated on the chip itself, not outside of the chip. But Gemalto AG had not been meeting this requirement from January

2011 to October 16, 2014. Scientists from the University of Tartu and experts from AS Cybernetica contributed to the discovery of this, explaining that some of the private keys were generated outside of the chip [30]. The possibility of a similar issue seems unlikely, but since other quoted attacks have also used proprietary implementation weaknesses, the ID-card could again be targeted due to not following agreements.

More recently, Lancia et al. [31] present a new flaw in the Java Card bytecode verifier from versions 2.2.2 to 3.0.5, where an external dependency check is missing between the Class component and the Descriptor component, two of twelve components that compose a CAP file. These external checks validate that redundant information specified in different components are compliant with one another. This allows controlling method offset to the header in bytecode, causing an overflow that brings the VM outside of the Class component and to trigger unverified bytecode execution in an applet that is already verified by the bytecode verifier. Finally, the authors gained full read/write OS privileges on the entire card's memory. As the ID-card is compliant with version 3.0.4, this attack could be possible, if proprietary countermeasures are not installed. Searching for proposed countermeasures, however, revealed an inconsistency in the work. The authors state at the beginning of their work that they propose a countermeasure to prevent their attack, but in the end, they state that after disclosure of the issue to Oracle, a new version of the bytecode verifier was released that fixes their presented attack (version 3.0.5u1), checking Class component inconsistency.

Many of the software-level vulnerabilities are traced back to a faulty verification of bytecode, as was the previous one, or the lack of an on-card verifier. Different solutions are suggested, for example, Lackner et al. [25] propose a defensive VM for Java Cards by adding additional hardware, namely bound, type, control flow and data integrity protection units, into a card. This hardware implementation has an execution time overhead of 4% compared to having none. A comparable software implementation would have an overhead of 159%. This, unfortunately, is not possible on the current new ID-cards, since their hardware implementation is set. However, on a new generation of cards, this solution should be considered.

We feel there is one more possibility for reducing the risk of allowing malicious code installed on the ID-card – code whitelisting. Though we could not find any publications on whitelisting (or blacklisting) Java Card code, a blacklist could be made by collecting

bytecode patterns that are connected to published exploits. An applet's code can be validated against it during verification, throwing a flag if a pattern is matched, indicating that a closer look is required. The Estonian authorities could also use tools to be shared with Application Developers to prevent false positives. For example, CheckStyle1 is a popular tool to enforce code conventions in Java, and has been used in several publications [32]–[35]. With a custom configuration of CheckStyle that is freely shared, fishy code can be automatically detected and fixed by Application Providers before submitting it for verification, reducing even more time required for fixes and resubmission.

5.5 Discussion

The main objective of Chapter 5 is to discuss the scenario of the Estonian authorities opening up the new ID-cards to third-party applets, and what should be done to reach that goal. We discussed enhancements to documentation and what should be disclosed to third parties in order to help them in their applet development, but these suggestions are rather subjective, and other developers will most probably have views that differ from ours. We also discussed use-cases for citizens and developers if the cards were to be opened to applets – we did not go very deep in terms of technical details regarding how and what should be done, but rather, envisioned how the relationships work between different parties in the whole cycle. Lastly, we made the reader aware of different types of vulnerabilities to Java Card smart cards, in particular regarding the contactless interface and software. The literature on the topic is vast, and we most definitely did not cover every possible threat, but rather, we tried to cover different categories of attacks.

5.6 Conclusion

Chapter 5 covers the occasion of when third-party applet development on the ID-cards will be opened to Application Providers. First, enhancements to existing documentation are suggested. In addition, resources required for meaningfully developing applets for the ID-cards are discussed. Next, possible scenarios of enabling applet installation on

[1] http://checkstyle.sourceforge.net/

citizens’ ID-cards are presented, followed by privileges that should be assigned to Application Providers during prototyping. Finally, publications on Java Card threats and vulnerability countermeasures are discussed, denoting the importance of security when it comes to allowing third parties’ code on the new ID-cards.

6 Evaluation

An evaluation of the Delegated Management enhancements made to GlobalPlatformPro was made in two steps. First, the enhancements were not simply merged into the project, but a fork was made of the project in GitHub, development was carried out on the fork, a draft pull request was made on the GlobalPlatformPro project and progress was constantly reviewed and verified by the maintainer of the project, Martin Paljak, before finally approving the pull request and merging the fork. Secondly, the enhancements were validated and demonstrated in Chapter 4 by attempting installation of an applet on the test ID-card, which required an appended Delegated Management Token with the content management commands.

To evaluate the results of this thesis with the Estonian authorities, a set of questions was constructed and sent to Andrei Kargin, the eID product manager at RIA, who has been the main contact from RIA during writing this thesis. The questions and answers are listed as follows:

1. Would you use the descriptions on applet development and installation in this thesis to put together instructions for Application Providers if the cards are opened to applets in the future? What would you change in the descriptions?

Answer: I would use this thesis as input in a public procurement for a “central service for adding applications to ID-cards”. The official instructions will be uploaded by RIA with a working development environment (which was initially our plan).

2. Are the scenario suggestions (in Section 5.3) for applet installation regarding citizens, and applet development regarding Application Providers viable in your view? Why?

Answer: I would supplement the depicted process in Figure 16 with an RCM (Remote Card Manager) service [(extra step between “ID-software” and “Applet database”, depicted in Figure 17)]. The “ID-software” in this figure would be the DigiDoc4 software that is currently used, with built-in functionality for managing applications on ID-cards.

A good idea (which already is a project outsourced by RIA) is creating a test environment for developers, where a test configuration of DigiDoc4 can use a test RCM chain with simplified (developer portal) uploading of applets to a Test Applet database.

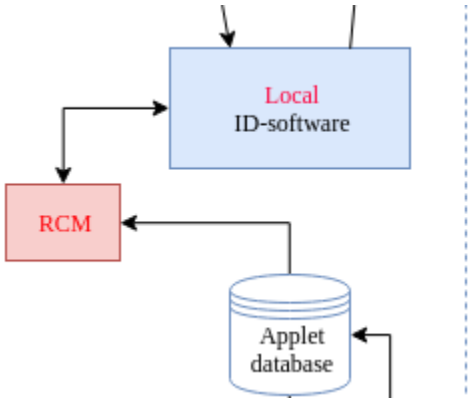

Figure 17. Modified applet installation scenario section.

3. Were you aware of all the vulnerabilities to smart cards outlined in Section 5.4? If not, are they now on your checklist for verification before allowing third-party applets on the ID-cards?

Answer: RIA will most certainly engage in a thorough security analysis regarding RFID (NFC) usage. Every possible risk connected to protocols and the cards will be evaluated. In parallel, the information that can be read over the contactless interface must be researched. If the question is “does RIA know of every possible vulnerability” then the answer is NO. This thesis helps us to better and more thoroughly evaluate possible threats.

4. Are the suggested blacklist (or whitelist) for bytecode instructions and CheckStyle configurations during development viable options to use during and before applet verification? Why?

Answer: I would leave this question unanswered. This highly depends on the development process, code auditing and CAP file compilation process (who and how).

6.1 Related Work

Covering related works to this thesis could include other tools for GlobalPlatform card content management and security analyses of identity documents and smart cards. As Sections 2.3 and 5.4 already cover security analyses by mapping threats and

countermeasures, this section will focus on introducing other open-source tools meant for content management on GlobalPlatform smart cards.

The GlobalPlatformPro project already lists a few similar projects that are available in its README.md file, for example, GPJ[1], which Paljak [36] states is the ancestor of the code in GlobalPlatformPro. The project was forked since "messing with cryptic script files was not nice and I wanted to have a simple, open source, usable and native-to-the-rest-of-development-environment (Java) toolchain" [36]. The GPJ project, however, is fairly outdated, with the latest update being in 2015. The project itself also refers to Paljak's GlobalPlatformPro as the continuation of their project.

There are also several other projects independent of GPJ, for example, GPShell[2]. This project is written in C, and as [36] states, it is often referred to as the *de facto* open-source implementation for GlobalPlatform. A quick search on Google with the phrase '*"globalplatformpro" site:stackoverflow.com*' also shows roughly 100 fewer results than the phrase '*"gpshell" site:stackoverflow.com*'. Usage is a bit more complicated than GlobalPlatformPro, however, since it requires compilation of several components before usage and it does not provide a direct command-line utility. This could also explain the larger amount of search results on Stack Overflow[3] when comparing to GlobalPlatformPro – the ease of use of the latter could also mean requiring less discussions on the popular developer portal. The project is updated regularly.

GlobalPlatform.NET[4] is a project written in C#. The project has a very nice fluent interface, for example, a Delete command can be constructed with the code segment depicted in Figure 18. The main downside to the project currently is that it only supports SCP02, deeming it unsuitable for many cards. The last update was on the 21st of May, 2018.

[1] http://gpj.sf.net

[2] https://sourceforge.net/projects/globalplatform/

[3] https://stackoverflow.com/

[4] https://github.com/jamesharling/GlobalPlatform.NET

```
DeleteCommand.Build
  .DeleteCardContent()
  .WithAID(aid)
  .AndRelatedObjects()
  .UsingToken(token)
```

Figure 18. GlobalPlatform.NET fluent API Delete command [37].

As the author of the current thesis is most comfortable with Java, and we had the invaluable support of Martin Paljak during writing, it made the most sense to use the open-source GlobalPlatformPro project and add Delegated Management support to it for managing contents on the new ID-cards.

7 Conclusion and Future Work

7.1 Conclusion

The main research question we wanted to answer was: **How to develop third-party applets for the new Estonian ID-cards?** We set out working on this thesis with the main goal of showing that third-party applets can indeed be developed and installed on the new Estonian ID-cards (or the test versions that are identically configured). The Estonian authorities had not yet been able to test this assumption when we first approached them, and it was agreed that a proof-of-concept would be very helpful to them. This thesis managed to take one of the most popular tools for content management on GlobalPlatform cards, GlobalPlatformPro, and add the Delegated Management functionality that was missing. This implementation allows authorising content management with the Load, Install, Make Selectable, Extradition and Delete Tokens appropriately. Support for these commands was required for successfully managing content on the test ID-card's Supplementary Security Domains, as they have DM privilege enabled. During adding DM support, we also took the opportunity to partly reformat the code of GlobalPlatformPro, in order to make it easier to grasp for newcomers, improve readability, remove duplication and avoid possible faults due to misconception. This reformatting allowed adding the DAP verification functionality to the *install* command.

As a result, the enhancements to the tool allowed installation and demonstration of an applet emulating an NDEF tag on the test ID-card's Supplementary Security Domain. We described all the required commands to successfully install an applet on the test ID-cards. During attempting installation, we discovered that the DM RSA public key that was installed on the ISD of the test ID-cards was different from the one that was said to be installed. Fortunately, we were able to replace the keys by ourselves, the process of which was also described. On production cards this fix will not be possible for developers, meaning that the Estonian authorities must ensure that the keys are valid before issuing cards to citizens, and to update any invalid keys that may have already been issued to citizens.

Next, as the Estonian authorities had not yet decided on if and how they will open the ID-cards to third-party applets, this thesis made suggestions on what information to disclose to developers, envisioned scenarios of safe applet development and installation for the ID-cards, and mapped vulnerabilities to smart cards that should be considered and to which resistance should be verified before opening the cards to third-party applets.

In conclusion, all of the research questions we set out with in Section 1.2.1 were answered. We hope the contributions of the current thesis are adequate for accelerating the work of the Estonian authorities and providing tool support for evaluating third-party applets on the new ID-cards. Hopefully, this thesis will help to one day open the ID-cards to useful third-party applets that will implement innovative use-cases, involving more people to use their ID-cards daily and replace the dozens of separate smart cards that each currently only serve one very specific task.

7.2 Future Work

The current addition of DM support in GlobalPlatformPro works as expected for the functionality initially required, but several use-cases remain to be covered. For example, adding a token as-is to the commands, and not calculating it dynamically, is an option that would be used if a third-party is not provided full access to a Security Domain, but rather RIA would authorise single content management actions for a specific applet to an Application Provider.

Also, as stated in Section 3.4, GlobalPlatform specifies Tokens for Registry Update and a combined Load, Install and Make Selectable Token, which this thesis does not implement. The latter of these would make most sense to apply in the previously described use-case where third parties do not have possession of any keys for unlimited access on a Security Domain, and RIA would otherwise have to generate a set of three tokens for every new request – a Load Token, an Install Token and a Make Selectable Token.

In addition, it is clear from working with the code of GlobalPlatformPro that it is not very easily readable, and thus not friendly to newcomers. For example, the project's classes currently are listed under a single package, rendering most of the different access modifiers on variables and methods misleading. Also, some methods are described with

comments, some with Javadoc. Additionally, method and variable naming is inconsistent (some as camelCase, some as lowercase with underscore separators) and, occasionally, too laconic or abstract (using abbreviations for GlobalPlatform concepts that are not abbreviated in the specification). What is more, some static parameters are not defined separately as literals, obfuscating their meaning and requiring paying close attention to the GlobalPlatform specification, reading and comparing every code line to the specification (for example, using "*0x01*" as-is for the LOAD instruction P1 parameter instead of referencing something like "*static final byte P1_MORE_BLOCKS = 0x01*"). This increases the learning curve drastically and lengthens the time that developers new to the project can start working on fixes or improvements.

Furthermore, using a code analysis tool, preferably as an IDE extension, such as SonarLint[1], could drastically improve code quality, consistency, readability and style, in addition to fixing critical code smells without too much work. For example, analysing the project with a default configuration of the SonarLint plugin v4.0.2.3009 in IntelliJ IDEA reveals 571 issues in 19 out of 23 project files (in GlobalPlatformPro release 19.01.22). Additionally, requiring all contributors to follow a "best-practices" approach to coding, such as those proposed by Robert C. Martin [38], would additionally improve readability and maintainability. To more easily help enforce these style conventions, a CheckStyle configuration can be adopted, as suggested at the end of Section 5.4.

Section 5.4 discusses existing publications regarding attacks against and vulnerabilities in Java Card smart cards. The existence of the discussed vulnerabilities, or any other publications in the Java Card security community, is unknown for the new Estonian ID-cards. It is important to verify the resistance against known attacks for the security of the ID-card and the Estonian citizens holding the card, before allowing installation of third-party applets on them. And since demanding resistance to every known attack is quite simply unnecessary, providing means to reduce risks during development and verification should mainly be focused on.

[1] https://www.sonarlint.org/

7.3 Summary

This thesis took on the task of proving the possibility of application deployment and development on the new generation of Estonian ID-cards, enclosing multi-application support and an NFC contactless interface. Reaching this goal first required choosing a tool enabling content management on GlobalPlatform compliant Java Card smart cards. The tool we chose was GlobalPlatformPro, which seems to be one of the most popular open-source tools available. This tool, however, lacked support for Delegated Management, an authorisation concept introduced in the GlobalPlatform specification. This was required since the Supplementary Security Domains installed on the cards have to have Delegated Management privilege enabled, meaning that a special authorisation Token must be appended to all content management commands sent to the Domain. Support for Delegated Management was implemented, and this allowed us to take an existing Java Card applet using the NFC functionality and deploy it on a test version of the ID-card, configured similarly to how the real cards are issued. After implementing the functionality, we described the required commands to successfully deploy an applet on a test ID-card. Next, after successfully deploying and demoing an applet on a test ID-card, we took on the task of assisting the Estonian authorities in actually opening this newfound functionality up to third parties interested in developing new useful applications for the ID-cards. For this, we suggested enhancements to documentation, resources for disclosure, scenarios for safe application management for citizens and safe applet development for developers, and finally, we mapped threats to smart cards and suggested countermeasures to help the Estonian authorities reduce the risk of opening the ID-cards to third-party applets.

References

[1] "What is Digi-ID," [Online]. Available: https://www.id.ee/index.php?id=34410. [Accessed 15 April 2019].

[2] A. Pau, "Eesti saab uue ID-kaardi," Postimees, 24 September 2018. [Online]. Available: https://tehnika.postimees.ee/6412572/eesti-saab-uue-id-kaardi. [Accessed 1 Decemeber 2018].

[3] Oracle, "Java Card 3 Platform Runtime Environment Specification, Classic Edition Version 3.0.5," May 2015. [Online]. Available: https://docs.oracle.com/javacard/3.0.5/JCCRE/JCCRE.pdf. [Accessed 25 February 2019].

[4] G. Bouffard and J.-L. Lanet, "Reversing the operating system of a Java based smart card," *Journal of Computer Virology and Hacking Techniques,* vol. 10, no. 4, pp. 239-253, 2014.

[5] V. Guyot, "Smart card, the stealth leaker," *Journal in Computer Virology,* vol. 8, no. 1-2, pp. 29-36, 2012.

[6] W. Rankl and W. Effing, Smart card handbook, John Wiley & Sons, 2004.

[7] Republic of Estonia Information System Authority, "Estonia ID1 Chip/App 2018 Technical Description," 2018. [Online]. Available: https://installer.id.ee/media/id2019/TD-ID1-Chip-App.pdf. [Accessed 1 December 2018].

[8] GlobalPlatform, "GlobalPlatform Card Specification 2.2.1," June 2011. [Online]. Available: https://members.globalplatform.org/kws/specifications/ store_items_from_woocommerce/ GPC_Specification-2.2.1.pdf. [Accessed 13 January 2019].

[9] GlobalPlatform, "GlobalPlatform Card Specification 2.3.1," March 2018. [Online]. Available: https://members.globalplatform.org/kws/specifications/ store_items_from_woocommerce/ GPC_CardSpecification_v2.3.1_PublicRelease_CC.pdf. [Accessed 13 January 2019].

[10] J. Iguchi-Cartigny and J.-L. Lanet, "Developing a Trojan applet in a Smart Card," *Journal in computer virology,* vol. 6, no. 4, pp. 343-351, 2010.

[11] Riigi Infosüsteemi Amet, "RIA-lt saab süsteemide testimiseks tellida uue ID-kaardi testkaardi," [Online]. Available: https://www.ria.ee/et/uudised/ria-lt-saab-susteemide-testimiseks-tellida-uue-id-kaardi-testkaardi.html. [Accessed 7 May 2019].

[12] B. E. G. Gomes, A. M. Moreira and D. Déharbe, "Developing Java card applications with B," *Electronic Notes in Theoretical Computer Science,* vol. 184, pp. 81-96, 2007.

[13] Republic of Estonia Information System Authority, "EstEID v 3.5 Estonian Electronic ID card application specification," 14 March 2017. [Online]. Available:

https://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3.5-20170314.pdf. [Accessed 7 May 2019].

[14] Oracle, “Java Card API, Classic Edition, Version 3.0.5,” [Online]. Available: https://docs.oracle.com/javacard/3.0.5/api/. [Accessed 10 April 2019].

[15] GlobalPlatform, *Contactless Services; Card Specification v2.3 - Amendment C; Version 1.2.1.5 (target 1.3),* 2019.

[16] T. S. Messerges, E. A. Dabbish and R. H. Sloan, “Examining smart-card security under the threat of power analysis attacks,” *IEEE transactions on computers,* vol. 51, no. 5, pp. 541-552, 2002.

[17] G. Avoine, A. Beaujeant, J. Hernandez-Castro, L. Demay and P. Teuwen, “A survey of security and privacy issues in ePassport protocols,” *ACM Computing Surveys (CSUR),* vol. 48, no. 3, p. 47, 2016.

[18] T. Chothia and V. Smirnov, “A traceability attack against e-passports,” in *International Conference on Financial Cryptography and Data Security,* 2010.

[19] T. Chothia, Y. Kawamoto and C. Novakovic, “A tool for estimating information leakage,” in *International Conference on Computer Aided Verification,* 2013.

[20] L. Sportiello, “ePassport: Side Channel in the Basic Access Control,” in *International Workshop on Radio Frequency Identification: Security and Privacy Issues,* Springer, 2015, pp. 173-184.

[21] A. Gkaniatsou, F. McNeill, A. Bundy, G. Steel, R. Focardi and C. Bozzato, “Getting to know your card: reverse-engineering the smart-card application protocol data unit,” *Proceedings of the 31st Annual Computer Security Applications Conference,* pp. 441-450, 2015.

[22] C. Bozzato, R. Focardi and F. S. G. Palmarini, “APDU-level attacks in PKCS#11 devices,” in *International Symposium on Research in Attacks, Intrusions, and Defenses,* 2016.

[23] I. Albrecht, “OpenJavaCard/openjavacard-ndef: NDEF tag implementation for JavaCard,” [Online]. Available: https://github.com/OpenJavaCard/openjavacard-ndef. [Accessed 17 April 2019].

[24] Code Blog Bt, “The principle of elliptic curve ECC ECDH & JavaCard implementation,” [Online]. Available: https://www.codeblogbt.com/archives/459881. [Accessed 18 April 2019].

[25] M. Lackner, R. Berlach, M. Hraschan, R. Weiss and C. Steger, “A defensive java card virtual machine to thwart fault attacks by microarchitectural support,” in *2013 International Conference on Risks and Security of Internet and Systems (CRiSIS),* 2013.

[26] Z. Kfir and A. Wool, “Picking virtual pockets using relay attacks on contactless smartcard,” in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05),* 2005.

[27] Y. a. S. D. a. W. A. Oren, “Range extension attacks on contactless smart cards,” in *European Symposium on Research in Computer Security,* 2013.

[28] K. Markantonakis, M. Tunstall, G. Hancke, I. Askoxylakis and K. Mayes, “Attacking smart card systems: Theory and practice,” *Information Security Technical Report,* vol. 14, no. 2, pp. 46-56, 2009.

[29] K. Hyppönen, “Use of cryptographic codes for bytecode verification in smartcard environment,” *Mémoire de master, Université de Kuopio,* 2003.

[30] Delfi, “ID-kaartide tootja Gemalto: PPA 152 miljoni nõue on ebaproportsionaalselt suur,” 27 September 2018. [Online]. Available: http://www.delfi.ee/news/paevauudised/eesti/id-kaartide-tootja-gemalto-ppa-152-miljoni-noue-on-ebaproportsionaalselt-suur?id=83833885. [Accessed 1 December 2018].

[31] J. Lancia and G. Bouffard, “Java card virtual machine compromising from a bytecode verified applet,” in *International Conference on Smart Card Research and Advanced Applications*, 2015.

[32] M. Smit, B. Gergel, H. J. Hoover and E. Stroulia, “Code convention adherence in evolving software,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011.

[33] S. V. Yulianto and I. Liem, “Automatic grader for programming assignment using source code analyzer,” in *2014 International Conference on Data and Software Engineering (ICODSE)*, 2014.

[34] Z. Lubsen, A. Zaidman and M. Pinzger, “Using association rules to study the co-evolution of production & test code,” in *2009 6th IEEE International Working Conference on Mining Software Repositories*, 2009.

[35] P. Louridas, “Static code analysis,” *IEEE Software*, vol. 23, no. 4, pp. 58-61, 2006.

[36] M. Paljak, “GlobalPlatformPro,” [Online]. Available: https://github.com/martinpaljak/GlobalPlatformPro. [Accessed 3 May 2019].

[37] “GlobalPlatform.NET,” [Online]. Available: https://github.com/jamesharling/GlobalPlatform.NET. [Accessed 4 May 2019].

[38] R. C. Martin, Clean code: a handbook of agile software craftsmanship, Pearson Education, 2009.

Appendix 1 – GlobalPlatformPro DAPProperties.java

```
1.  package pro.javacard.gp;
2.
3.  import joptsimple.OptionSet;
4.  import pro.javacard.AID;
5.
6.  import javax.smartcardio.CardException;
7.
8.  import static pro.javacard.gp.GPCommandLineInterface.OPT_DAP_DOMAIN;
9.  import static pro.javacard.gp.GPCommandLineInterface.OPT_TO;
10.
11. public class DAPProperties {
12.     private AID targetDomain = null;
13.     private AID dapDomain = null;
14.     private boolean required = false;
15.
16.     public DAPProperties(OptionSet args, GlobalPlatform gp) throws CardException,
    GPException {
17.         // Override target and check for DAP
18.         if (args.has(OPT_TO)) {
19.             targetDomain = AID.fromString(args.valueOf(OPT_TO));
20.             if (gp.getRegistry().getDomain(targetDomain) == null) {
21.                 throw new GPException("Specified target domain is invalid: " +
    targetDomain);
22.             }
23.             if (gp.getRegistry().getDomain(targetDomain).getPrivileges()
    .has(GPRegistryEntry.Privilege.DAPVerification))
24.                 required = true;
25.         }
26.
27.         // Check if DAP block is required
28.         for (GPRegistryEntryApp e : gp.getRegistry().allDomains()) {
29.             if (e.getPrivileges()
    .has(GPRegistryEntry.Privilege.MandatedDAPVerification))
30.                 required = true;
31.         }
32.
33.         // Check if DAP is overriden
34.         if (args.has(OPT_DAP_DOMAIN)) {
35.             dapDomain = AID.fromString(args.valueOf(OPT_DAP_DOMAIN));
36.             GPRegistryEntry.Privileges p = gp.getRegistry().getDomain(dapDomain)
    .getPrivileges();
37.             if (!(p.has(GPRegistryEntry.Privilege.DAPVerification) ||
    p.has(GPRegistryEntry.Privilege.MandatedDAPVerification))) {
38.                 throw new GPException("Specified DAP domain does not have
    (Mandated)DAPVerification privilege: " + p.toString());
39.             }
40.         }
41.     }
42.
43.     public AID getTargetDomain() {
44.         return targetDomain;
45.     }
46.
```

```
47.     public AID getDapDomain() {
48.         return dapDomain;
49.     }
50.
51.     public boolean isRequired() {
52.         return required;
53.     }
54. }
```

Appendix 2 – GlobalPlatformPro DMTokenGenerator.java

```java
package pro.javacard.gp;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.smartcardio.CommandAPDU;
import java.io.ByteArrayOutputStream;
import java.security.PrivateKey;
import java.security.Signature;

import static pro.javacard.gp.GlobalPlatform.INS_DELETE;

public class DMTokenGenerator {
    private static final Logger logger = LoggerFactory.getLogger(DMTokenGenerator.class);
    private static final String acceptedSignatureAlgorithm = "SHA1withRSA";

    private PrivateKey key;

    public DMTokenGenerator(PrivateKey key) {
        this.key = key;
    }

    public CommandAPDU applyToken(CommandAPDU apdu) {
        ByteArrayOutputStream newData = new ByteArrayOutputStream();

        try {
            newData.write(apdu.getData());
            if (apdu.getINS() == INS_DELETE || apdu.getINS() == (INS_DELETE & 255)) {
                // See GP 2.3.1 Table 11-23
                logger.debug("Adding tag 0x9E before Delete Token");
                newData.write(0x9E);
            }
            if (key == null) {
                logger.debug("No private key for token generation provided");
                newData.write(0); //Token length
            } else {
                logger.debug("Using private key for token generation (" + acceptedSignatureAlgorithm + ")");
                byte[] token = calculateToken(apdu, key);
                newData.write(token.length);
                newData.write(token);
            }
            return new CommandAPDU(apdu.getCLA(), apdu.getINS(), apdu.getP1(), apdu.getP2(), newData.toByteArray());
        } catch (Exception e) {
            throw new RuntimeException("Could not add DM token to constructed APDU", e);
        }
    }

```

```
48.     private static byte[] calculateToken(CommandAPDU apdu, PrivateKey key) {
49.         return signData(key, getTokenData(apdu));
50.     }
51.
52.     private static byte[] getTokenData(CommandAPDU apdu) {
53.         try {
54.             ByteArrayOutputStream bo = new ByteArrayOutputStream();
55.             bo.write(apdu.getP1());
56.             bo.write(apdu.getP2());
57.             bo.write(apdu.getData().length);
58.             bo.write(apdu.getData());
59.             return bo.toByteArray();
60.         } catch (Exception e) {
61.             throw new RuntimeException("Could not get P1/P2 or data for token
    calculation", e);
62.         }
63.     }
64.
65.     private static byte[] signData(PrivateKey privateKey, byte[] apduData) {
66.         try {
67.             Signature signature = Signature.getInstance(acceptedSignatureAlgorithm
    );
68.             signature.initSign(privateKey);
69.             signature.update(apduData);
70.             return signature.sign();
71.         } catch (Exception e) {
72.             throw new RuntimeException("Could not create signature with instance "
     + acceptedSignatureAlgorithm, e);
73.         }
74.     }
75.
76.     public boolean hasKey() {
77.         return key != null;
78.     }
79.
80. }
```

Appendix 3 – Configuring NDEF Tags with NFC Tools