TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Pavel Kargin 204784IVCM

Development of a Third Party Applet for the Estonian National ID

Master's thesis

Supervisor: Juhan-Peep Ernits
PhD

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Pavel Kargin 204784IVCM

Kolmanda osapoole apleti arendus eesti ID-kaardile

Magistritöö

Juhendaja: Juhan-Peep Ernits
PhD

Tallinn 2022

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: Pavel Kargin

12.12.2022

Abstract

The current master's thesis describes JavaCard technologies with a focus on the development of an applet for the Estonian electronic identity card. The first half of the work presents a brief history of the Estonian national identity card with an overview of the standards currently used in electronic identity cards and other chip-based technologies using JavaCard technology. After that, a password manager solution is presented, which will be used as an example to create a practical part of the thesis. The practical part focuses on creating a working applet that can be placed on the Estonian ID card, and instructions on how to download and install it using some of the existing solutions and tools still in development.

Electronic documents such as the Estonian ID card are created following standards that are developed by ICAO and the Global Platform. These standards set the architecture and security requirements for JavaCard-based chips. In the case of Estonian documents, these standards have been fully observed since the 2018 version of the identity card

A password manager is a popular solution for keeping personal data, such as passwords, securely in one encrypted vault. Although passwords are becoming less popular, they are still widely used. Some of the solutions that are gaining popularity are, for example, token-based access or one-time passwords. For the current thesis, a password store was chosen to show one example of how to add additional features to an ID card that can be used in conjunction with other document features while also considering the availability of storage resources on the card. The practical work is devoted to the development of a simple example of a password manager in the form of an applet, a practical example of loading and installing on an Estonian ID card, and then using the functionality through the terminal.

The current thesis is written in English and is 56 pages long, including 5 chapters, 26 figures, and 6 tables.

Annotatsioon

Kolmanda osapoole apleti arendus eesti ID-kaardile

Käesolevas magistritöös kirjeldatakse JavaCardi tehnoloogiaid, keskendudes Eesti elektroonilise isikutunnistuse apleti arendamisele. Töö esimeses pooles esitatakse lühidalt Eesti riikliku isikutunnistuse ajalugu koos ülevaatega praegu kasutatavatest standarditest elektroonilistes isikutunnistustes ja muudes JavaCard tehnoloogiat kasutavates kiibipõhistes tehnoloogiates. Seejärel esitletakse paroolihalduri lahendust, mida kasutatakse näitena lõputöö praktilise osa koostamisel. Praktilise osa fookuses on Eesti ID kaardile paigutatava töötava apleti loomine ning juhised selle allalaadimiseks ja installimiseks, kasutades mõnda olemasolevat ja arendatavat tööriista.

Elektroonilised dokumendid nagu Eesti ID kaart kasutavad ICAO ja Global Platformi poolt välja töötatud standardeid. Need standardid määravad javakaardipõhiste kiipide arhitektuuri- ja turvanõuded. Eesti dokumentide puhul on neid standardeid täielikult järgitud alates isikutunnistuse 2018. aasta versioonist.

Paroolihaldur on populaarne lahendus isiklike andmete (nt paroolide) turvaliseks hoidmiseks ühes krüptitud hoidlas. Kuigi paroolid muutuvad üha populaarsemaks, kasutatakse neid endiselt laialdaselt igas tööstusharus. Mõned lahendused, mis populaarsust koguvad, on näiteks token-põhine juurdepääs või ühekordsed paroolid. Selle lõputöö jaoks valiti paroolide hoidla, et näidata, kuidas lisada ID kaardile lisavõimalusi, mida saab kasutada koos teiste dokumendifunktsioonidega. Praktiline töö on pühendatud apleti kujul oleva paroolihalduri lihtsa näite väljatöötamisele, kuidas seda Eesti elektroonilisele ID-le paigutada ja seejärel terminali kaudu funktsionaalsust kasutada.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 56 leheküljel, 5 peatükki, 26 joonist, 6 tabelit.

List of abbreviations and terms

e-ID	Electronic identification
APDU	Application protocol data unit
PM	Password Manager
PIN	Personal Identification Number
PKI	Public Key Infrastructure
APDU	Application protocol data unit
ARMIS	Application Remote Management Information System
AID	Applet identifier
ICAO	The International Civil Aviation Organization
IDE	Integrated development environment
eIDAS	Regulation on electronic identification and trust services
RIA	Estonian Information System Authority
eMRTD	Electronic machine readable travel document

Table of contents

List of Figures

List of tables

1 Introduction

The motivation for the current work is to provide a working concept of using the Estonian national ID card as an additional tool in everyday life in addition to its main functionality. As a document, an Estonian electronic identity card can be used online to sign documents or log into online services. But there are no working solutions that can provide additional functionality. The main goal of the current work is to prove by example the general viability of such add-ons and to give 3rd parties an example JavaCard applet that can be loaded to the ID card.

The underlying hardware of electronic documents like ID cards, which are based on JavaCard technology, can, in principle, support some additional security-enhancing functionality without disrupting the primary purpose. We will demonstrate by the example of a password manager applet how to utilize the available resources provided by the hardware.

The questions that the current thesis sets out to find answers to are:

1. How can a third party create an applet for an Estonian ID card?
2. What testing workflow can be used to ensure the quality of the developed applets and the security of the existing parts of the document?
3. What criteria must an electronic document meet after installing the applet?
4. What can be developed for JavaCard using the Estonian ID card as an example, and what choices do third parties have?

The scope of the work is to have a working conceptual prototype of a password manager applet on a test JavaCard with an Estonian ID profile (the same as in the real document, but with test keys).

The key assumption is that an applet can be installed on an Estonian ID card, which can be accessed after installation, while the basic security criteria of the ID card will still be met. The limitation of the current thesis will be that only one specific applet will be created and tested only on the Estonian electronic identity card. Thus, memory constraints and the availability of the protected Estonian ID domain will restrict the developed solution.

The first topic for research is the Estonian identity card. Several studies have been carried out in Estonia on various aspects of the document. The main gap was the lack of information from international sources about the possibilities or security issues of the Estonian national identity card. Research like “Tehnilised eeltingimused Kolmanda osapoole rakenduste lubamiseks uuel Eesti ID-Kaardil,” [8] (Technical Perquisitions to Allow Third Party applications to new ID card), “Proaktiivsete Teenuste Võimalikkus ID-Kaardi Näitel,” [9] (Proactive Service Possibility on Example of ID card) give a good overview of the capabilities of the Estonian ID that came out in 2018. Also, an additional useful overview of the success of the Estonian ID can help to get a better overview of the Estonian ID history and future possibilities “Eesti ID-kaardi ja selle elektroonilise kasutuse levik ja edu loo tagamaad,” [10] (Estonian ID Card and it’s Electronic Usage Spread and Success Story).

The next topic of research is the overview of JavaCard technology and the development of an applet for it. The technology is well known, and many different papers and experiments have been carried out with it. There exist examples of different solutions on the market for electronic ID cards and password managers. The gap addressed in the current thesis was the lack of research on potential applet developments for national ID cards. Research like “Smart card programming and security” [4], “Design and analysis of an improved smart card-based Remote User Password Authentication Scheme,” [5] and “A robust smart card and remote user password-based authentication protocol using extended chaotic maps under smart cities environment” [6] show how to create password managers for smart cards in general. They can help with the understanding of basic principles and ways to approach development in my research. At the same time papers such as “Methods for pre-processing smart card data to improve data quality,” [7] can help with the general quality of the applet to improve the results of later tests on the test cards. The main problems with these studies are that they are performed on the basic test card without any profile on it. In the current research, it is planned to explore how to place an applet on a card with an existing profile and limited memory without affecting existing parts of the document.

The last part of the research is devoted to the example application - password manager. There are different approaches for creating secure storage, which give different results, for example, more secure, but with a higher power or memory consumption, lighter solutions, but with complex implementations, and cloud solutions that may not correspond to the ideology of storing passwords in the local environment. There has been good research done about password vaults and about inexperienced developers or developers who do not do enough research on the topic and are trying to implement secure storage. This leaves the minimal problem of finding the best implementations and mostly comparing the proposed solutions. Examples of solutions are “Implementation of enhanced secure hash algorithm towards a secured web portal “[1] "Authentication and password storing improvement using SXR algorithm with a hash function" [2] "Securing password using dynamic password policy generator algorithm" [3].

2 Electronic Identity Card

Electronic Identification Card (eIC), which is a physical proof of identity that can be used for online and offline personal identification or authentication. eIC is a smart card in the ID-1 format of a regular bank card with identification information printed on the surface and an embedded RFID microchip, similar to that used in biometric passports [16].

An electronic identity is a set of data that links a person to their physical identity in an electronic environment. This means that the electronic part of the ID card also contains personal information, as well as certificates that are used to verify identity in online transactions. The basis for the functioning of the electronic identity is the public key infrastructure or PKI.

The PKI model is based on two keys - a private key and a public key. As the names of the keys already say, the private key must be protected and only the person to whom it is issued can use it. Again, the public key is available to everyone and there is a definite relationship between the two keys [12]. PKI is used to secure online information transactions for different situations like online banking or e-mailing confidential documents.

An example is the Estonian ID card, which stores personal data and certificates, which can be used to determine a person's identity. Part of the information is hidden behind PIN1 and PIN2. They can be used to read off the card, for example, the authentication and signing certificates, and this is also necessary to confirm operations in an internet environment, such as an online bank or governmental e-service. The same approach to protecting data using a PIN can be used to protect password manager passwords in the applet on a smart card.

2.1 Continuations of Previous Research

In 2019 Gregor Johannson, who researched potential additional usage for the Estonian ID card published a thesis that focused on showing that third-party applets can be placed on the new version of the Estonian ID card from 2018. His work aimed to give guidelines for the developers and give some documentation that could have been used by the Estonian Information System Authority [8].

The main focus of his work was on a theoretical potential approach to placing applets in a new supplementary domain on a card to spark interest in developing applets for the Estonian ID card. The questions he seeked answers to are: " How to add Delegated Management support to GlobalPlatformPro for managing content on the new ID-cards?", "How can third parties manage the contents of the new ID cards?" and “How ID card parties should be allowed to develop and prototype applets on the new ID cards?” [8].

The current thesis is a logical continuation of Gregor's research. It focuses on showing a more practical approach to installing an applet and exploring the scenario of using it. This means that it will partly answer the question of how third parties can install applets and what applet prototyping might look like [8].

2.2 Standards

Any technology that is used by a large number of people, such as in big companies and internationally between different countries, should have some agreements and rules for its use. It accelerates development and helps prevent problems between different versions. This also applies to the development and design of electronic identity cards.

2.2.1 ICAO Standards

The standards of the International Civil Aviation Organization or ICAO partly help to understand what a valid Estonian national identity card should look like, as well as some technical details. In the context of Estonia, they provide some guidance on how the communication with a smart card should look like in the electronic machine readable travel document or in short eMRTD part of the card, what the design of a smart card should be, and an easier way to validate cards against standards to ensure that the document behaves correctly according to ICAO Doc 9303. All documents that follow these guidelines have a special symbol in the form of a small rectangle that looks like a camera symbol. If a document has this symbol, then it is a biometric document. These documents also have biometric data such as facial images and fingerprints and can be read electronically. The symbol is shown in Figure 1.

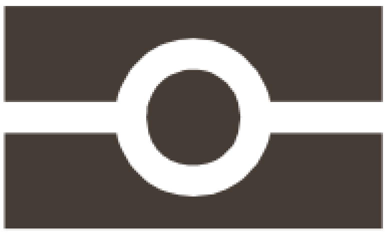


Figure 1. ICAO eMRTD symbol [18].

ICAO's work on machine-readable travel documents began in 1968 with the establishment of the Air Transport Bureau Group Council Committee on Passport Cards. This group was tasked with developing recommendations for a standardized passport book or card that would be machine-readable to expedite the processing of passengers at passport control. The group made several recommendations, including the adoption of optical character recognition as the preferred machine reading technology due to its maturity and cost-effectiveness. and reliability. In 1980, the specifications and guidance material developed by the Panel were published as the first edition of Doc 9303 entitled Machine-Readable Passport, which became the basis for the initial issuance of machine-readable passports in Australia, Canada, and the United States. [14].

In general, following these guidelines helped different countries to ease traveling between countries. This approach covers several sides.

2.2.1.1 Security of the document

Researching and implementing security features requires appropriately skilled experts and is thus expensive. To ensure that documents cannot be easily counterfeited, or data spoofed, testing and proof-of-concept generation are necessary. In the case of documents, high security is extremely important. Different organizations may find different ways to protect their documents, but if there is no agreement between the parties on this matter, then the use of documents can be limited to only one country.

With unified document standards, different parties can also more quickly agree on a new implementation of the physical checks and encryption standards that must be used to ensure that the person traveling with these documents is really whom they say they are. In the context of electronic ID cards, there are additional features attached to the card, such as electronic authentication and electronic signature. The process of creating security artefacts, such as encryption keys, is simplified if there is confidence in the algorithms and lengths of the keys used.

2.2.1.2 Efficiency of the document checks

The classic way to check documents at the border is to check the face on the physical document, check the name in the database for criminal records, and check if the document looks legitimate. The main responsibility and expertise in these cases lie with the border guard. Which is hard work and does not rule out mistakes.

In the 21st century, the use of biometric verification such as fingerprints and facial recognition, reading the electronic part of a document, and scanning the document itself is becoming more common. Thus, the use and use of more electronic ways of document verification are spreading. The main problem with this check is that each country has a different structure within the document. For other countries to be able to perform verification, either a single software or a single way of creating documents is needed.

ICAO standards describe the implementation of electronic parts of documents, which solves the problem of different document implementations. This speeds up the verification of each document that complies with these rules and simplifies the movement of data at destination airports.

2.2.1.3 Faster integration of the documents

With a standardized way to implement the security functions and structures on the document, countries can create their document following the standards much faster. Countries do not need to spend time developing their standards and structures, they can use already established methods and, if necessary, seek assistance from countries that have already implemented documents in the same way. At the same time, the base of using the same secure documents is also growing, which improves security in each country and reduces the chances of traveling with fake identities.

2.2.2 GlobalPlatfom Specifications

The GlobalPlatform specifications are developed by the non-profit organization that set the standards for the variety of smart card systems like cards, terminals, and management systems [11][51]. Around 2010, the world of smart cards experienced a change that led to a new generation of cards. In this context, GlobalPlatform aims to create specifications that standardize certain aspects of the technology to increase the availability and interoperability of multi-application smart card technology. The specification was created by Visa International as the Visa Open Platform (VOP) [51].

For smart cards to reach their full potential, consumers must be able to use them for a variety of purposes and functions. For example, cards can be used with mobile phones to make purchases over the Internet as well as for secure access to a PC. Smart cards also need to be economical and feature-rich. The GlobalPlatform architecture is designed to provide card issuers with a system management architecture to manage their smart cards. Although GlobalPlatform is based on the paradigm of having only one card issuer, it offers flexibility to the card issuer to manage the ever-changing business area of partners who want to run applications on the card issuer's cards.

2.2.2.1 Card architecture

Card Manager is the central component of the GlobalPlatform card. It consists of three services: Issuer Security Domain, GlobalPlatform environment, and Cardholder Verification Method services provider. Each service is launched through it, and it also provides an interface that can be used internally through the GlobalPlatform API or externally through APDU commands. In addition, it contains the security domain of the card manufacturer. It consists of three parts the GlobalPlatform environment, issuer security domain, and cardholder verification method services.

The **runtime environment** is designed to run on the runtime environment of any secure multi-application card. The runtime environment is responsible for ensuring that the hardware-neutral API for applications, secure data storage, and the start-up space for applications are secure. This ensures that each application's code and data are separate and secure from other applications on the card. In addition, this part is responsible for being able to communicate with the card and external devices of the card. Regarding reporting parameters such as communication protocol, logical channels, and command chains in ATQ/ATR, smart cards must comply with the relevant standards: ISO 7816-3, ISO 7816-4, ISO 14443-3, and ISO 14443-4 [11].

Security domain act as internal representatives of external authorities. Among them are the keys on the card. This part can also be called an application with different sets of permissions. The issuer security domain represents the card administrator on the smart card. The card administrator is usually the card issuer. Additional security domains or supplementary security domains are additional, optional, built-in application representatives of providers or the Card Issuer. Controlling authority security domains are special kinds of supplementary security domains that represent regulatory authorities [11].

2.2.2.2 Security architecture

Well-designed security architectures are critical to protecting the structure and function of the cards in the GlobalPlatform system. The standard provides guidelines for designing a secure architecture without delving into the topic of cryptography. Robustly built components and well-designed communication between the various parts can provide the necessary security when a new version of a smart card is published. With the smart card applet's secure architecture, developers have fewer security concerns to address.

2.2.2.3 Implementation

The GlobalPlatform standard describes well the recommendations for implementing smart cards and applets on them. The main part that is important for the current study is the status of the applets on the smart card. During the life cycle of an applet, it can be in states such as LOADED, SELECTABLE, and LOCKED. For an applet to be used on a smart card and achieve the final practical result, it must have the SELECTABLE status.

Since the current study is focused on creating an applet for an already personalized document known as the Estonian national ID card, there is no need to implement any security solutions during the practical part. This will reduce the scale and help avoid issues such as key management and card communication implementation. Although to implement the smart card applet, APDU commands logic that is described in the GlobalPlatform specification must be followed.

2.3 Estonian ID card

In the context of Estonian, an ID card is an identity document intended for electronic use, which is issued by the Police and Border Guard Board. The Estonian ID card uses a so-called secure data carrier, which consists of a chip, certificates, and keys (private and public) stored in it, which ensure the security of transactions. It is compliant with electronic transactions in the internal market regulation (eIDAS Regulation), which means that it is recognized as a travel document in the same way as a passport within the European Union.

2.3.1 Gemalto version

Initially, it was decided to implement an ID card on top of the slightly modified Micardo platform versions. All used platform modifications and used features of the Micardo operating system, card commands, and structure were specified in the local standard EVS 827:2004. [15].

In Estonia, this version of the ID card was issued since the year 2002, the manufacturer of which was Gemalto. The owner of the document could use it as a regular document, conduct an electronic identification of themselves and put a digital signature. One of the important services that a citizen with this card could use was also electronic voting.

In this version of the ID card, there was no domain available for ordinary users to install additional applets, although technologies allowed them to do so. The main smart card domain contained only those applets that were installed there during manufacturing.

2.3.2 IDEMIA version

A new generation of electronic identification cards has appeared in Estonia since the end of 2018. This document is based on the JavaCard technology and conforms to the GlobalPatform specification [8]. IDEMIA became the manufacturer of the new version of the ID card. Although there are several reasons for changing the manufacturer, one of them could be legal problems in 2017 between Gemalto AG and the Police and Border Guard Board [52].

The visual side of the new ID card is also different from the previous one. Some new security elements were added to the new card, the layout of the information was changed, and the image was changed to color. In addition, hidden security elements that can only be seen with a UV light have been changed.

The transition to JavaCard technologies also made it possible to create a Supplementary Domain for the Estonian ID card, where potentially every interested party can place their developed applet. This opportunity gave impetus to the beginning of the research and development of the proof of concept for this paper.

In the summer of 2019 with new European Union legislation, there were new requirements for travel documents such as ID cards, which also affected the Estonian national ID card [17]. According to the new rules, the Estonian ID card had to have a face and fingerprints on the card itself. Previously, this data was stored off the card and was only available electronically through government web pages or apps like DigiDoc. In August 2021, a new version of the IDEMIA ID card for Estonia was released, which contains biometric data following the new rules. The picture that is physically visible on the card can also be obtained through the card reader. With some additional steps, it is also possible to read fingerprints.

2.3.3 ID card crisis

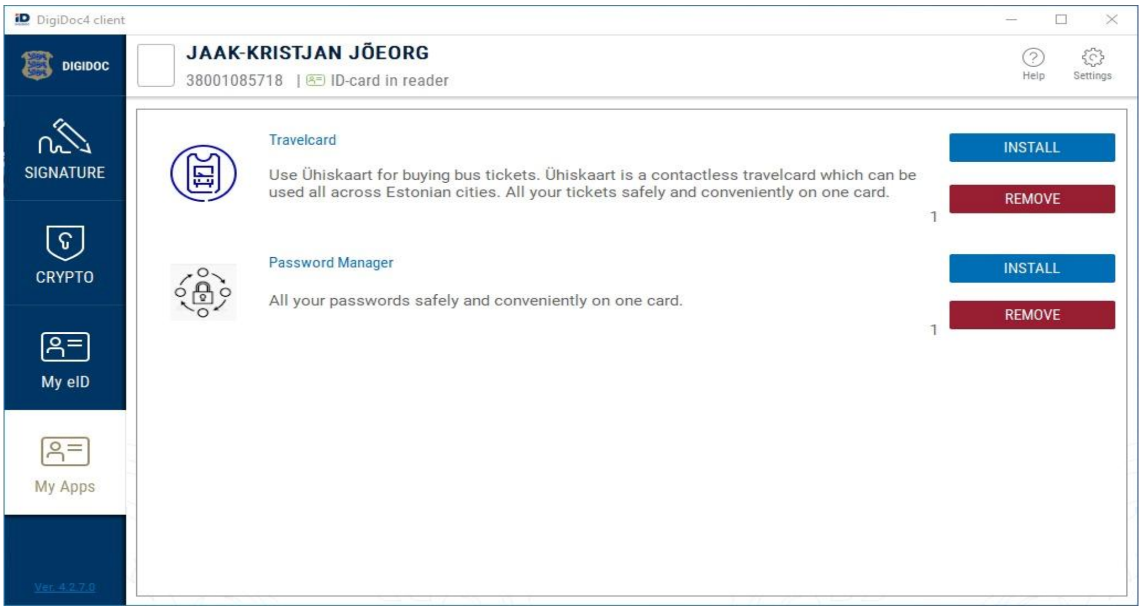
Estonian ID card credibility was hit in 2017. This was an example of an insecure implementation of a cryptographic algorithm. Studies have shown that the Estonian ID card is theoretically insecure due to defective chip manufacturing. This crisis affected about 800,000 documents, which is a huge number considering the 1.3 million inhabitants of Estonia. The ID card needed to be updated to protect the certificates. The Estonian Information System Authority, in cooperation with Estonian technology companies such as SK Solutions, has developed software that allows ID card holders to update their documents without having to replace them. This was a huge achievement and also proved the government's ability to deal with the e-governance crisis. Although the vulnerability was not exploited by attackers, it posed a real threat [24].

2.3.4 ARMIS

Application Remote Management Information System or ARMIS is a project controlled by the Estonian Information Systems Authority that aims to make it easier for everyone to develop and obtain applets. ARMIS can be regarded also as an applet manager (manager applet). This project aims to create a manager applet that will help sign and install applets on a smart card. All operations are planned to be performed through the DigiDoc [46] interface. There are some requirements for the Estonian Information System Authority, users, and third-party vendors that must be followed for everything to work [49]. Development of the manager applet is done in by RIA.

2.3.4.1 Government responsibilities

All the operations to manage the card content are enabled by the Global Platform specification. All the applet loading, installing, personalizing, and removing operations are performed for the applet issuer by the Estonian Information System Authority. Also, the government side provides the tools to enable third-party applet issuers. Basic processing, quality, and security checks will be performed by the Estonian Information System Authority before the applet is added to the applet store for general use. The interface example is in Figure 2.



Use Ühiskaart for buying bus tickets. Ühiskaart is a contactless travelcard which can be used all across Estonian cities. All your tickets safely and conveniently on one card.

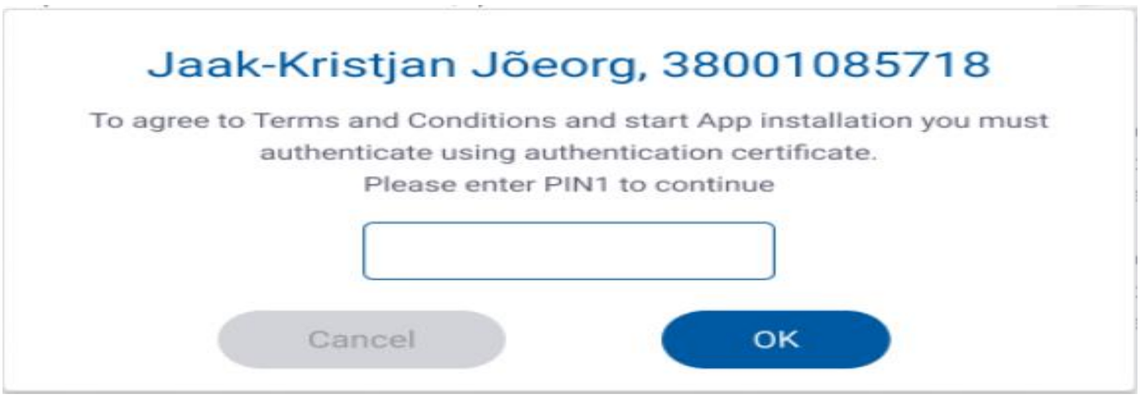
All your passwords safely and conveniently on one card.

Figure 2. Applet store view.

The applets will be installed using the same DigiDoc4 tool that is currently used to read the ID card’s personal data, sign, encrypt, and decrypt documents. The image above shows a prototype update that will add a “My Apps” view where is visible applet store with solutions available for installation. In the same view, it is possible to manage already installed applets.

2.3.4.2 User responsibilities

If users want to get a particular applet for their ID card, they need to visit the Applet Store view in DigiDoc. To use the store, the user needs PIN1 to verify their identity. It is the user's responsibility to ensure that the applets are updated/installed/removed. Due to the limited space on the smart card, the user cards will not be able to install an unlimited number of applications. PIN1 insertion field example is in Figure 3.


Jaak-Kristjan Jõeorg, 38001085718

To agree to Terms and Conditions and start App installation you must authenticate using authentication certificate.
Please enter PIN1 to continue

Figure 3. Authentication of the user.

The PIN1 document will only be used for the installation process, which means that the applet must have a separate authentication method. Along with the identity verification for the ID card, there may also be additional password fields that provide security for the installed applet. With password managers as an example, it is important to have a separate user PIN that will be used every time there is a need to access stored data.

2.3.4.3 Third-party responsibilities

The idea behind the project is to make applet development more attractive to third parties. Companies will develop applets that can promote their solutions or help sell certain products. Some examples of ideas could be a crypto wallet, a discount card vault, a public transport card, and a password manager. The latter is already the focus of the current thesis. Companies must guarantee the security of their applets, or they will not be allowed to add their product to the store.

Test coverage should be part of quality assurance. Because applet security is an important topic due to it potentially being part of a citizen's document, some mandatory rules must be laid down for developers. These rules will help companies have confidence in the quality of their product, as well as a responsible party for ID cards, such as RIA, who will review an applet before it is allowed into the applet store. The topic of testing will be covered in more detail in the following paragraphs.

2.3.4.4 Manager Applet

The manager applet holds two purposes on the card:

1. Each third-party applet on the smart card will have its AID. The ecosystem does not know which applets are third-party applets. The manager applet contains a

registry of currently installed third-party applets and their AIDs. During applet installation and removal, the third-party applet uses the common interface provided by the manager applet to notify the installation and removal of AIDs.

2. To optimize memory usage and enable communication integrity and confidentiality between the issuers.

The personalization process is performed during the installation of the manager applet, which must be completed before installing any new applet. During the personalization of the manager applet, an elliptic curve key pair is generated. The public key is provided to the personalized, and its authenticity is verified. confirmed by the RIA by issuing a certificate. The private key never leaves the manager applet. Access to the private key object is not available, but it can be used through objects derived from a Sharable Interface Object (SIO). Key pair generation may be performed by the RIA during personalization and after personalization. The latter is required if the private key is somehow exposed to the public or hacked in any other way. This means that the key pair can change during the life cycle of the ARMIS applet ecosystem.

2.3.4.5 Difference in the creation of the applet

From a technical point of view, when implementing an applet for JavaCard, it is necessary to use certain libraries and classes. ARMIS ecosystem uses them, it extends `Applet.class` and implements `AppletEvent.class` from `JavaCard.framework` and implements `Personalization.class` from `org.globalplatform`. To develop an applet that will be compatible with ARMIS, instead of extending the `Applet.class`, there is a need to extend `AbstractApplet.class` from `ee.openeid.armis.applet.ecosystem.libs`. This approach is not limited to that class, but each class from `JavaCard.framework` needs to be replaced with an equivalent and extended class from the ARMIS applet ecosystem.

Also associated with these changes is the requirement to provide keys to sign the applet that will be used by the ARMIS ecosystem to sign the CAP file before load and installation. Also, some additional functionality like methods `isAuthenticated()`can be used to check authentication before continuing with a new function like `unwrap()` that takes input keys and cipher and unwraps the incoming APDU.

3 JavaCard Technology

JavaCard is a platform with many security features for smart cards for various types of applications such as payments, electronic documents, and telecommunications. Although it is popular in the telecommunications sector for SIM cards, it is not as popular in applications requiring new standardized crypto algorithms, open-source projects, or research projects. Historically, the main reasons for infrequent use may be limited access to low-level cryptographic primitives, which is not the case in newer versions, and the lack of data types such as lists or integers that are commonly used in Java [22].

JavaCards expose their functionality through various specialized packages that are part of the standard API. Their list includes high-level cryptographic operations, biometric authentication, and remote method invocation. Due to the limitations that smart cards have, most of these packages are implemented in a dedicated crypto coprocessor. This solution helps improve the performance of the smart card.

Some examples of JavaCard usage in technologies are Europay, MasterCard and Visa cards, Subscriber Identity Modules, Identity Mapping Modules, Smart ID Passes, Medical Cards, and as mentioned before government-issued documents. Also, it can give access to 5G [29].

The benefits of JavaCard technology include reduced hardware costs, easier and more secure installation and upgrades due to the open OS architecture, and the ability to host multiple applets on a single device. Thanks to the simple design of the JavaCard, fast production can be achieved. In addition, the security of cards is ensured by the strong protection of the Java language [29].

3.1.1 JavaCard 3.0.5

For current research work, JavaCard version 3.0.5, also known as Classic Edition, which was released in 2015, will be used. It was an evolution of version 2, released in 2006. The positive side of the update is that it is backward compatible with the old version, and there is the possibility to use the newer version with the old one, as long as a link to the new library functions is not used. Smart cards using the JavaCard Classic Edition are considered secure enough for widespread use. This version of the security profile has been certified by the “Bundesamt für Sicherheit in der Informationstechnik“ to CC

EAL4+, ALC_DVS.2, AVA_VAN.5 certification levels can be used to achieve EAL4+ and higher certification levels for JavaCard products [21].

In total, there are only 19 packages defined by the JavaCard 3.0.5 API. That includes the next packages for cryptography: key agreement algorithms, key pair storage, key generation, random number generators, symmetric encryption schemes, asymmetric signing schemes, and cryptographic hash functions [22].

3.1.2 JavaCard 3.1.0

In 2019, JavaCard 3.1 introduced an enhanced file format, static resource management, binary compatibility improvements, and support for array representations. These features improve application deployment and upgrades and provide better modularity and security.

Although this version of JavaCard was not considered when implementing the solution for the current thesis, because it is not yet supported on smart cards, it may help future applet developers learn about new features that will be added. It can also add to confidence in the technology.

The main things that this version brought to JavaCard are the establishment of a direct channel between the security element and device peripherals to ensure security at the very edge of the device, secure communication with IoT cloud serv, ices and performance of various device attestation mechanisms to prove device authenticity to a third party.

3.1.2.1 New features

The main improvement for applet developers will be the enhanced CAP format. With changes to static resource management, improvements to binary compatibility, and array support, applet development not only becomes easier but provides more options for applet security. Also, a huge advantage is that these changes provide backward compatibility.

An extended CAP file can contain multiple application packages and Java libraries. CAP file sizes can exceed 64KB. This removes design constraints and reduces the number of CAP files deployed by a trusted service manager. A Java package can be private to a specific application, as opposed to public and shared packages that can be accessed by different applications available on the platform. This allows for an improved design and more precise access control [19].

3.1.2.2 Security improvements

With new features for the applet, development came also new security features. In the list of upgrades are new encryption algorithms, support for configurable asymmetric key-pair generation, and support for elliptic curve cryptography named curves. In short, this version has updated cryptographic packages to treat keys as trusted entities and support modern algorithms and related operations [19].

The certificate API allows one to verify the signature of a certificate, select and verify some of its fields and extensions, and access its public key without having to create a dedicated certificate object. Cryptographic certificates are the main part of the public key infrastructure. This helps establish trust between several different parts. One of the most well-known examples is the chain of trust, which is established before the client and server can authenticate each other through the certificate infrastructure [20].

The next one will be the API for getting the keys. It is used in cryptography with pseudo-random functions to obtain data like a private key. There are several uses for that technology, such as deriving a password to store a derived value without having to store the original key value, deriving a shared secret in an Elliptic Curve Diffie-Hellman key exchange, and TLS handshake protocol negotiation. Currently, eight algorithms are proposed, which, in particular, provide support for the International ICAO or TLS protocols [20].

To prevent replay attacks, a monotonic API counter was introduced. When a monotonic counter is used, it increments its value by some amount, and when attached to a timestamped payload, the counter is protected from being reused and attached to the payload later. The entity consuming the payload will know if the payload has been consumed by checking the value of the counter. One example would be external secure storage. DRM licenses or the number of PIN attempts are typical sensitive data protected by the JavaCard security element [20].

The last one is the system time API. The timestamp is an important feature because it allows for keeping track of the time of events. Knowing the time intervals, it is possible to improve, change or limit the duration of transactions. Some way to learn about algorithms and hack systems is to keep track of how long an operation takes by entering various data into the application. Some data may take longer to verify than others, which may indicate that it may be correct. The two new operations are SysTime, which shows the time since loading, and TimeDuration, which shows the length of time in microseconds [20].

3.2 Security concerns

Like any technology, JavaCard has its drawbacks. Each new version tries to fix vulnerabilities and introduce new protection methods. There are studies dedicated to testing the security limitations of the JavaCard platform. Smart card attacks use a combination of software and hardware. One approach that has shown results is using the JavaCard API, which aims to obfuscate the JavaCard applet and gain access to another object, including cryptographic keys. The next step is to access the return address of methods in JavaCard with the countermeasure of stack splitting. Several vulnerability studies have shown that it is possible to modify legitimate applications on the card to use it. It is possible to create an innocent-looking code that, under certain conditions, can become hostile [23].

3.3 JavaCard Usage in ID cards

The European Union (EU) regulates which ID card documents are considered trustworthy. All countries whose electronic documents pass eIDAS are considered safe and trustworthy by other EU countries. This allows, for example, the Estonian ID card to be used as a traveling document in the same way as a passport throughout the EU. The rules set by eIDAS ensure that citizens of different countries can use their ID cards to access other countries' online government services, and also ensure that the document will work abroad and have legal rights such as a passport [42].

To achieve this level of trust, national document-producing companies have chosen to use JavaCard due to its security and accessibility innovations. When it comes to electronic document production, government organizations are trying to find a manufacturer capable of producing and personalizing smart cards. Some such smart card companies known in Europe are IDEMIA, Giesecke + Devrient GmBH, and the Thales Groups [41]. Contracts with these companies are concluded for at least one life cycle of the document, in the example of the Estonian ID card it would be 5 years.

3.4 Applets

Java applets were introduced for use with the Java language in 1995. These are legacy applications that were small and were written in Java or another programming language, then compiled into Java bytecode and served to users in Java bytecode. They ran through web browsers and required a Java virtual machine to run. Starting in 2013, major browsers started dropping support for Java applets, and in 2019, with Java 9, these applets have been deprecated [25].

JavaCard applets have the same design as Java applets. JavaCard applets also work with the JavaCard virtual machine, but with a different encoding to make them smaller to accommodate smart card size limitations. So, the same Java applet will be larger than the JavaCard applet. This encoding method and the resulting smaller size have the side effect of missing some of the Java language libraries and features. Although there are methods to overcome size limitations by splitting the application code into packages smaller than 64 KiB [26].

As mentioned earlier the limitations of JavaCard technology compared to Java, include the absence of some object types that are commonly used when writing Java code. This list includes char, double, float, long, enums, multi-dimensional arrays, threads, final objects, and object cloning. Most smart cards also do not support int-s and garbage collection objects in runtime. This also includes the Estonian ID card.

Differences from a Java applet are not limited to encryption. Library and runtime support in JavaCard technology differs from normal Java support. One security difference is that the JavaCard cannot use a class like the Java Security Manager and the same security policies are enforced by the JavaCard virtual machine. Also, through the JavaCard class library, fast RAM variables are supported, which are in regular Java without import.

Some features are unique to the JavaCard runtime and this platform. One of the features is the isolation of the applet, which acts as a firewall. It creates isolation of applets from other applets. But in addition, these same functions make it possible to create unique access to applets for other applets. The big difference with a regular Java application is that in a JavaCard, objects are stored in persistent memory. The RAM on a smart card is really limited, so it is only used when there is a need to store sensitive or temporary data. One last thing is that the JavaCard is written individually, and the bytecode instructions are atomic. Transaction mechanisms are limited. This is because the smart card is externally powered and dependent on permanent memory [26].

From a security perspective, JavaCard was developed to protect sensitive data on smart cards. For this, several technologies were introduced. The first is that the data in the applets and JavaCard application is executed separately from the operating system in the JavaCard virtual machine. Next, as mentioned before the firewall, separates the applets. The most important for an ID card would be cryptographic algorithms. These include key generation, key exchange, and signing. This means using symmetric and asymmetric key algorithms. Advanced Encryption Standard, Data Encryption Standard, and Triple Data Encryption Standard for symmetric. And the asymmetric algorithms are Rivest-Shamir-Adelman, also known as RSA and elliptic curve cryptography [26].

3.4.1 JavaCard Development Kit

For applet development, Oracle introduced the JavaCard Development Kit tool, which uses the GlobalPlatform to manage the load and installation of applets. In the context of smart cards, it is correct to use the terms "load" when explaining the process of placing an applet on a smart card, and "install" when installing a loaded file. The development kit makes it easier to configure these processes, which helps develop applets. The kit also has a function to create an applet that is added to the read-only memory of the smart card. For example, in Estonian, the ID card applet that manages the user's personal data is read-only, so a new document must be requested to change personal data [27].

Development Kit provides a JavaCard framework for applet development. That includes functionality like simulating JavaCard runtime. It is also known as JavaCard Workstation Development Environment. For communication with the card itself is a tool that sends to this virtual environment APDU commands. As mentioned before there is functionality to convert the applet to a format to make it read-only and to mask it. The tool also helps to check the integrity of the files and to install it on smart cards [27].

The development kit provides a JavaCard platform for applet development. This includes features such as simulating the JavaCard runtime. It is also known as the JavaCard workstation development environment. To communicate with the card itself, there is a tool that sends APDU commands to this virtual environment which can be used also with the virtual environment. As mentioned earlier, there is a function to convert the applet to a format that allows it to make it read-only and/or mask it. The tool also helps to check the integrity of the files and install it on the smart card [27].

Summarizing the functionality of the JavaCard Development Kit. The tool allows one to compile Java files and create applets in CAP format from them, which can be placed on a smart card. It is also possible to test the applet in a JavaCard virtual environment using the APDU sending tool. In addition, the development kit provides the ability to mask the applet and make it read-only on the card.

3.4.2 CAP format

In previous chapters, the CAP file has been referred to in the context of applets. CAP is a converted applet. This format is similar to a *.jar* file that is generated from regular Java code with all the classes and libraries. Each CAP file in the same way contains all the classes that were defined in the Java code but in the context of smart cards.

In this format, multiple files are stored in byte order, with high-order bytes first. Each stream consists of 8-bit bytes, and in the case of large sizes, such as 16-bit, 32-bit, and 64-bit, reading is performed sequentially from two, four, or eight 8-bit bytes [28].

CAP files consist of several parts such as validation and linking information, information about all classes including executable bytecodes, and some others. The instructions for the bytecode are defined with a small size in mind, so the generated data is compact with limited indirection. After the installation procedure on the smart card, the installed applet will have a unique ID called the Application Identifier or AID [28].

CAP file is in reality zipped file that if unpacked has files that hold information about applet components. This includes:

- **Import** and **Export** files containing a list of imported classes and elements that can be exported, respectively.
- **Header**, **Directory**, and **Applet** files, which contain, respectively, general information, component sizes, and an entry for each applet in the CAP.
- **Class** and **Method** files contain information about classes and methods.

- **RefLocation** contains information about the data that is associated with methods, while **ConstantPool** contains information about fields, classes, and methods.

- The **StaticField** is useful for initializing because it has all the information about all the fields it needed for that.

- For debugging, a **Debug** file with metadata about the package and a **Descriptor** that contains all the necessary information to check all elements is used.

Sometimes a CAP file malfunction can only be detected after it is installed on a smart card. This may be the result of file corruption in the CAP file [28].

4 Case study: Password Manager

When choosing an application to implement as a case study, the following criteria were stated:

1) The application needs to demonstrate the full development cycle of the 3rd party application.

2) The application needs to demonstrate storage capabilities of the card.

3) The development of the application needs to be as simple as possible to keep the focus on the development of 3rd party applications for the ID card.

4) The application needs to demonstrate two-way communication with the card.

When considering different applications for demonstration, several alternatives were considered. For example, a FIDO2 authentication and one-time password (OTP) application were considered but were left for future work due to the added complexity of handling the appropriate communication protocols between the smart card and the browser. A password manager allows us to:

1) Demonstrate the full development cycle of an ID card application.

2) Allows demonstrating how to utilize storage on the card and how to handle the scenario when there is no more free space for the application.

3) The development is self-contained without additional protocols and complexity involved, and

4) Demonstrates two-way communication with the card, i.e. storing passwords and then retrieving them in a way protected by a PIN.

In short, a password manager is a solution that allows users to store, generate and manage their passwords for local applications and online services. Password management technologies are already widely used in many environments where there is a need to remember a bunch of passwords at the same time. In addition, it is a more secure way to exchange passwords between company employees. Currently, using only passwords is not considered the most secure way to effectively protect accounts or data. One of the weaknesses is remembering and creating complex passwords and updating them on regular basis. Password managers are trying to solve these problems, although they still have issues that need to be fixed. While some companies are already trying to move away from passwords and use more passwordless solutions such as FIDO2, PKI, biometrics, etc. using passwords is still relevant.

4.1 Logic Overview

The general logic of password managers is that they allow the storage of the data in one place, in protected and encrypted storage. Instead of having to remember the password for each service, one can store all the data in one storage and remember only one "master" password. As the name of the password manager suggests, it is designed to store passwords, but with the development of services, it is now also used to securely store credit cards, access keys, tokens, important notes, and similar sensitive data. This encrypted database can be implemented and used in two different ways: either stored locally, such as on a PC, on a separate disk, or, as the current document implies, on a smart card or JavaCard, or in a cloud storage, such as OnePassword, Keepass, and Lastpass.

4.2 Implementation Overview

As mentioned earlier, there are different approaches to using password managers. Each has its benefits, and which one to use depends on the use case. These solutions are locally stored passwords, passwords stored in the cloud, and a token-based approach. Current work will focus on the combination of a locally stored solution and a token-based approach.

4.2.1 Locally Stored Passwords

These types of password managers are usually found on some device that is owned and operated by the owner of the sensitive data. As mentioned earlier, it could be some kind of electronic device with encrypted database storage. This usually requires the installation of additional software. Some solutions may require an Internet connection to retrieve data from the Internet, and then locally installed software handles sensitive information. Although in this case a clear example of such solutions would be stand-alone software, eliminating the possibility of data leakage during data transmission over the network.

Unfortunately, this comes with the inconvenience of not always having access to it everywhere if the storage device is unavailable [31].

4.2.2 Passwords Stored in the Cloud

In this context, cloud storages are password managers, which are websites where data is stored. The two main differences between cloud and locally stored solutions are portability and the fact that usually there is no need to install additional software to use an online service. Online solutions minimize the risk of damage to the owner’s storage device and relieve users of the responsibility of backing up their data [32]. Although using a web service is much more convenient, users must trust the hosting site that it is protected from cyberattacks and that they follow best practices to secure user’s data. These services can also sync the password vault to the user's mobile devices which is secured with encryption.

4.2.3 Token-based Approach

The security token mechanism uses hardware devices to authenticate the user and is in addition to passwords. Hardware devices can be, for example, smart cards or USB sticks. To make the data in the token secure, it must also be encrypted on the device. This solution may require additional software, special hardware, and drivers to read and decode data.

4.3 Security

The use of password managers is regarded by security experts as a safe solution that provides a more convenient way to hide sensitive information from unwanted eyes. But this is not a consistent approach. Some companies deliberately break password managers because they believe it is beneficial for the user to remember their passwords and not trust this work to some service. For some, the benefits are not good enough to justify ignoring the concerns [35][36].

4.3.1 Benefits

Password managers offer several enhancements and features for creating and using secure passwords. This includes the following things:

1. **Create secure passwords for the user**. This option offers to create highly secure passwords that will require a lot of time and resources to crack using brute force. Automatically generated passwords are long and alphanumeric, but the downside is that they are difficult to remember. We can calculate the entropy of a password by first finding the entropy of one character in the set of R characters, which is equal to $\log_2 R$, and then multiplying it by the number of characters in the password. So, the formula to calculate entropy for passwords would be $E = L * \log_2(R)$ [30].

2. **One master password to remember**. It is much easier to maintain only one password. Easier to remember, update and create complex ones. Some problems with updating passwords are coming up with new ones, especially if a user has too many environments and passwords to handle. This can result in the reuse or minor modification of an existing password, such as adding a new number, making it more likely to be cracked. Although this approach has some drawbacks. They will be discussed in subsequent topics [34].

3. **Password manager data can be synchronized across multiple devices.** The big advantage of a cloud-based password managers service is that users can access them from multiple devices without transferring the data by themselves. This includes synchronization between different operating systems such as Windows, macOS, Linux, Android, etc. In the cloud, these benefits are available as long as users have access to the Internet. Unfortunately, there is no access to such benefits with offline password managers.

4.3.2 Concerns

With every convenient solution comes security flaws. For example, the following:

1. **One master password to crack or forget.** Since it is convenient to have only one password, it also poses a security risk in case it is brute forced or if the owner forgets it. If the attackers guess the password, they will know all the secrets hidden in the vault, which will require the owner to update all passwords. And if the master password is lost, the owner may lose access to all services.

2. **Not secure password managers.** If the password manager of choice does not store the data in encrypted form or the cloud storage provider has a poor data transfer implementation, these passwords can be leaked. Several popular password manager providers have had security issues. For example, LastPass had a vulnerability in their plugin, OneLogin had their AWS keys stolen, and Keeper exposed passwords for untrusted web pages [33].

3. **Data corruption and need for backups.** A similar problem with losing the master password occurs when data is lost due to disk corruption. To protect data from this, a backup copy needs to be created. This either means extra effort on the part of the owner if it is local storage, or trust in online services to take care of it, which goes back to choosing the right provider [34].

4.4 Password Manager Examples

There are many different examples of working solutions, but they have similar implementations of security systems. The main difference lies in the monetization methods used and whether they are open source or not.

4.4.1 Keepass

This type of manager is free to use and stores data on the owner's device. One of the benefits of Keepass is that it is completely open source and certified by the Open Source Initiative. This allows users to see what and how was implemented in this tool. Users also have the option to make some corrections or implementations themselves, and if these changes are approved by the project administrators, they may make it into the next update. Open source prevents backdoors by compiling the code in the personal environment if necessary and making changes for its convenience [47].

Keepass supports strong encryption using Advanced Encryption Standard and Twofish algorithm for database encryption. A single master password is used to encrypt the database, but key files can be used as an alternative. It is possible to generate strong random passwords using user input or some actions such as mouse movement. One of the features is that it is portable and works without installation, but can only be used on the Windows operating system. In addition, passwords can be exported to various file formats such as TXT, HTML, XML, and CSV [47].

4.4.2 OnePassword vs Lastpass

The other side of password managers is those that hold data in the cloud and sell their products. They are designed to provide a simple and secure way to store sensitive information and use it wherever needed. This is especially useful in companies where there is a need to share passwords or keys between employees. Lastpass and OnePassword are currently known as examples. Both have advantages and offer security for data but have different approaches to some solutions that come down to ease of use.

LastPass is the only app that offers a free version. But for more money, OnePassword offers local apps, a better user experience, and a less stringent security policy, which is probably best for large groups. OnePassword relies heavily on a proactive crash management style, which means that if recovery data is lost, it is usually impossible to log in again to the account. LastPass, on the other hand, offers simple recovery methods that are less secure [48].

LastPass and 1Password work pretty much the same on mobile platforms. Both services also have browser extensions for the most popular browsers. The bigger difference comes in desktop applications. OnePassword has options to use apps on Windows, Linux, and Mac users; LastPass is more browser-plugin-oriented. Both solutions are viable and widely used, but from a business standpoint, it is not easy to say which one is better. [48].

4.5 Passwordless Solutions

Passwordless authentication solutions, as the name suggests, are authentication methods that do not require passwords. Some of them right now are in addition to passwords as an additional layer of protection. In recent years, industries have begun to use these solutions much more widely because they are safer, more reliable, and with lower maintenance costs. They can be used as standalone solutions but require some additional implementations. The most famous examples are one-time passwords, biometrics, magic links, and push notifications [37].

While this approach is modern and saves money on maintenance, it may require additional investment during the implementation phase. This may include new software and hardware. If to go with a hardware installation, then this means buying, for example, cards or tokens. The software may be cheaper to implement but may increase the cost of maintenance and administration. Even with passwordless authentication, attackers can use "man in the browser" malware to compromise the security of a tool or application. Hackers can install malware designed to intercept one-time passwords [37].

5 Practical Approach

To show the viability of the concept of a working password manager as an applet for ID cards, in the current thesis will be presented as a proof of concept. The work will be divided into several parts. Each part will represent a working component of the solution. The current part of the work will combine practical and theoretical approaches. In order not to make the scope of the current paper too large, the only fully practically implemented part will be the development of the applet.

The first part will focus on designing a password manager and making it compatible with the limitations of JavaCard applets. The next step is to test the developed solution as an application, its behavior on the test Estonian ID card, and whether it breaks something that already existed on the test JavaCard before. The last part is a browser extension that accepts data from the applet and can use to enter a webpage.

For this practical part, a JavaCard with an Estonian ID card profile and the ARMIS applet installed is required. The Estonian identity card will be the test card that was used to check the validity of the document before issuing it. After testing the test card, ARMIS was installed on it. The Estonian Information System Authority provided a test Estonian identity card and the necessary documentation about ARMIS.

5.1 Development of Password Manager

Current work will use Java as the programming language, Apache Ant as the software tool to create the CAP file, and Apache Maven to create the JAR file that will install the CAP in JavaCard. Gradle was considered as an alternative for generating JAR and CAP files, but research on tools concluded that Maven and Ant are easier to use with Martin Paljak's applet installation software. Also, Maven has better examples to achieve the goals of the current thesis than Gradle. In order not to increase the scope of the paper, it was decided not to use Gradle.

To communicate with the smart card, Martin Paljak's software was used, which interacts with cards that comply with the GlobalPlatform specification, called GlobalPlatformPro. It allows sending specific APDU commands. It can be used to test an applet or manually select certain things on the card. Knowing the structure of the card, it is easy to find specific saved fields.

5.1.1 Limitations

As mentioned earlier, JavaCard has very limited support for various types, e.g. there is no support for collections such as a lists and even the 4 byte signed integer value type is implemented only on some cards. An additional problem in applet development is limited memory. Before the addition of biometrics to the ID card, it had approximately 100 551 bytes of memory, and after adding the image and fingerprints, only approximately 50 232 bytes remained. In this regard, the question arises as to how much it is possible to achieve with this amount of memory and whether this will become a problem in the future.

Limited memory becomes an even bigger problem after the installation of the applet manager. With ARMIS installed, the card will need to manage with even less room. To give an example of memory differences, have been used GlobalPlatformPro software to read the amount of memory remaining on a test Estonian ID card. This gave the following APDU responses.

Table 1. ARMIS memory usage on the card without biometrics.

Before ARMIS installation	**After ARMIS installation**

`A>> T=1 (4+0000) 00CADF64 06`
`A<< (0003+2) (86ms) 0188C7 9000`
	`A>> T=1 (4+0000) 00CADF64 06`
	`A<< (0003+2) (25ms) 015099 9000`

Hex `01 88 C7` in this context is equal to 100 551 bytes. With biometrics 50 232 bytes.	Hex `01 50 99` in this context is equal to 86 169 bytes. With biometrics 35 850 bytes.

Table 1 shows the result of sending the `java -jar gp.jar -a 00CADF6406 -dv` command, which sends an APDU response with memory amount in bytes. As can be seen in Table 1, after installing ARMIS, the test card has 14 382 bytes less than before. This means that there is less space left to install more applets. At the same time, it will be more difficult to manage applets without ARMIS, so for now, this is the only approach. The development of ARMIS is not yet complete, so the amount of space it takes up may change in the future.

With this knowledge in mind, when developing applets, developers should be careful about what they want to store in memory. If the applet is supposed to store some extra information, as in the password manager example, it must also store extra data after installation, then the applet should have a separate limit on the amount of extra memory it is allowed to use.

5.1.2 List of the Applets

First, to start communicating with the JavaCard applet must be established connection between the application and the card. Before using the developed password manager, it must be selected from the applets available for selection on the card. All applets that were not placed during the personalization phase at the factory are in the Supplementary Domain. To find selectable applets, have been used GlobalPlatformPro. Correct domain AID must be specified, in this case, it is `D233000000444F4D` and the encryption key to get access to data. In the current example, the correct value of the key has been changed to XXX. An example is shown in Table 2.

Table 2. Selectable APDU-s

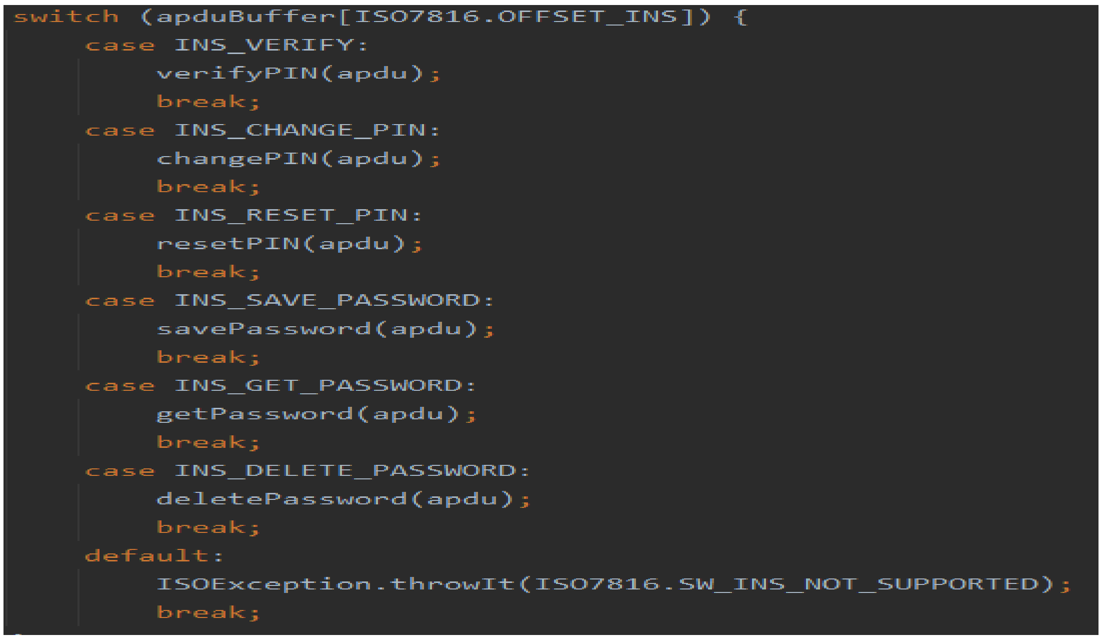
Command	`gp -dvli -key XXX -kdf3 --sdaid D233000000444F4D`

Output	`DOM: D233000000444F4D (PERSONALIZED) (|.3...DOM|)`
	`Privs: SecurityDomain, DelegatedManagement, CardLock, TrustedPath`
	`APP: 4D616E61676572417070 (SELECTABLE) (|ManagerApp|)`
	`Privs:`
	`APP: 0123456789000006 (SELECTABLE) (|........|)`
	`Privs:`
	`APP: 4D616E61676572417071 (SELECTABLE) (|ManagerApq|)`
	`Privs:`
	`PKG: 41524D49532D6C6962 (LOADED) (|ARMIS-lib|)`
	`PKG: 4D616E61676572 (LOADED) (|Manager|)`
	`Applet: 4D616E61676572417070 (|ManagerApp|)`
	`PKG: 4D616E6167657241 (LOADED) (|ManagerA|)`
	`Applet: 4D616E61676572417071 (|ManagerApq|)`
	`PKG: 0123456789 (LOADED) (|.....|)`
	`Applet: 0123456789000006 (|........|)`

5.1.3 Structure of the Password Manager Applet

The example above shows all available applets during the test. In this particular case, selectable ARMIS applets, a loaded and installed password manager with AID 0123456789, and applet AID 0123456789000006 are visible. Loaded applets must be installed before they can be selected. To install, must select LOADED AID and after installation, AID will change to the applet and will be in the SELECTABLE status.

After selecting the correct password manager applet, it needs certain commands that the applet is programmed to react to. The specific combination of CLA and INS in the developed applet will allow calling functions with different tasks. In the examples given, the functions are `verifyPin`, `changePIN`, `resetPIN`, `savePassword`, `getPassword`, and `deletePassword`. Acceptable CLAs and INSs are shown in Figure 4.



```
switch (apduBuffer[ISO7816.OFFSET_INS]) {
    case INS_VERIFY:
        verifyPIN(apdu);
        break;
    case INS_CHANGE_PIN:
        changePIN(apdu);
        break;
    case INS_RESET_PIN:
        resetPIN(apdu);
        break;
    case INS_SAVE_PASSWORD:
        savePassword(apdu);
        break;
    case INS_GET_PASSWORD:
        getPassword(apdu);
        break;
    case INS_DELETE_PASSWORD:
        deletePassword(apdu);
        break;
    default:
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
        break;
```

Figure 4. Password manager acceptable CLA and INS [50].

All functions in the developed applet will receive as input APDU and will use all remaining parts except CLA and INS to read the information. The rest of the parts represent the DATA. When a function that enables a password manager is called, the correct user PIN must be provided, but additional data is not needed when calling for the function that shows the available space. The example in Figure 5 shows how the authorization part works.

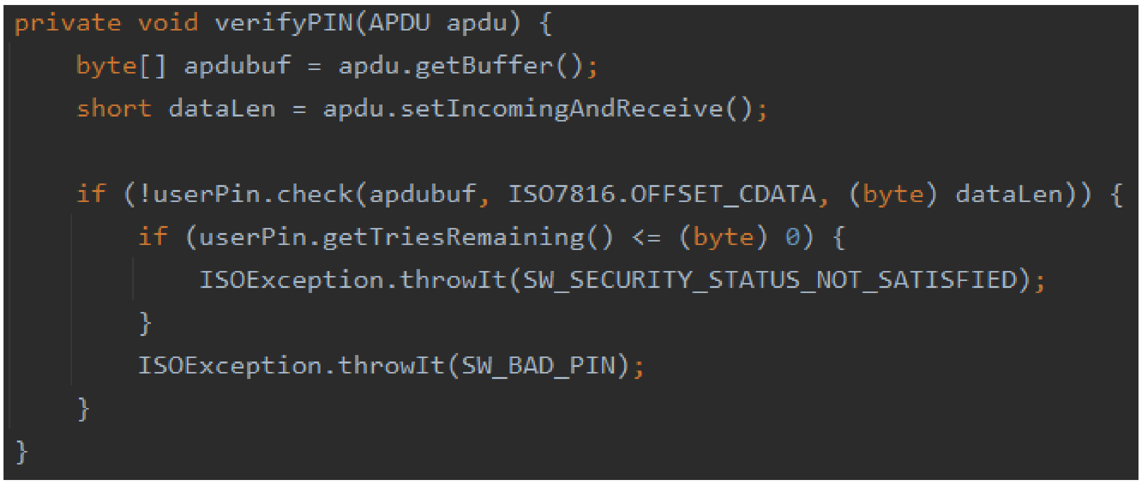

```
private void verifyPIN(APDU apdu) {
    byte[] apdubuf = apdu.getBuffer();
    short dataLen = apdu.setIncomingAndReceive();

    if (!userPin.check(apdubuf, ISO7816.OFFSET_CDATA, (byte) dataLen)) {
        if (userPin.getTriesRemaining() <= (byte) 0) {
            ISOException.throwIt(SW_SECURITY_STATUS_NOT_SATISFIED);
        }
        ISOException.throwIt(SW_BAD_PIN);
    }
}
```

Figure 5. Enable password manager usage function [50].

5.1.4 Ensuring the Security of the Passwords

The password stored in the manager must be protected. For the developed applet, the approach taken was using a custom PIN that needs to be sent to the applet before it returns any useful information about the data stored inside. This is done using the `JavaCard.framework.OwnerPIN` as one of the available libraries from Oracle for JavaCard. It already has built-in logic for checking PIN codes. Figure 6 depicts simple logic for validating a PIN. To pass this verification, the user needs to pass the verifyPIN, which is shown in Figure 5.

The admin and user PINs are created during the installation operation. For the install operation, the APDU command can provide a data field with specified PINs, but for this particular case study, the hardcoded values 0000 and 00000 were used. As an example, the `changePIN` function has also been created, which allows changing the user PIN if the correct admin PIN has been used. The PIN code declaration can be seen in Figure 7.

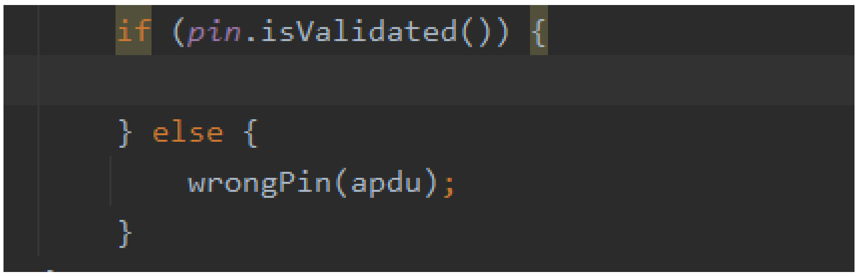
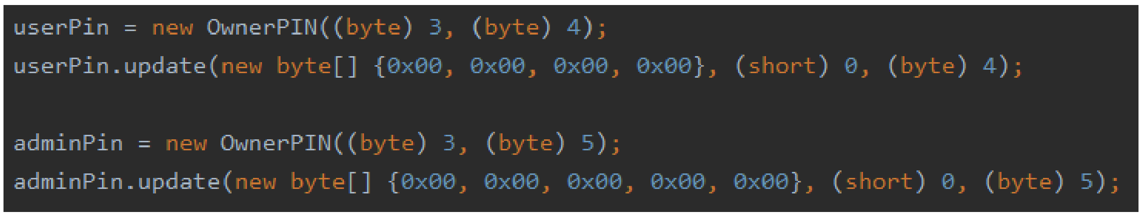

```
        if (pin.isValidated()) {

        } else {
            wrongPin(apdu);
        }
```

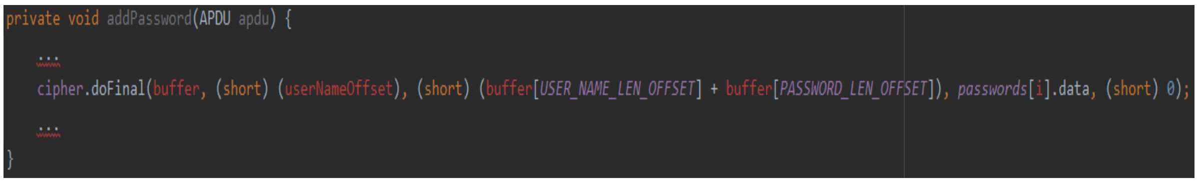
Figure 6. PIN validation [50].



```
userPin = new OwnerPIN((byte) 3, (byte) 4);
userPin.update(new byte[] {0x00, 0x00, 0x00, 0x00}, (short) 0, (byte) 4);

adminPin = new OwnerPIN((byte) 3, (byte) 5);
adminPin.update(new byte[] {0x00, 0x00, 0x00, 0x00, 0x00}, (short) 0, (byte) 5);
```

Figure 7. Admin and user PIN [50].

Also, to ensure the security of operations, another available class from the Oracle JavaCard libraries is `javacardx.crypto.Cipher` is used, which allows the performing of data encryption and decryption operations using a secret key. The developed applet uses an AES cipher and key for this. To process the APDU DATA part, the `doFinal` function is used, which generates encrypted or decrypted output from all or the last input. The admin PIN is generated during the installation phase, as shown in Figure 8. The figure after shows an example of the `addPassword` function.



```
private void addPassword(APDU apdu) {

    ...
    cipher.doFinal(buffer, (short) (userNameOffset), (short) (buffer[USER_NAME_LEN_OFFSET] + buffer[PASSWORD_LEN_OFFSET]), passwords[i].data, (short) 0);

    ...
}
```

Figure 8. Add password function with Cipher [50].

5.1.5 Saving Password

The password save logic will take the input password in HEX format and store it in an array. The password must not be longer than a certain amount, and the array must only accept a certain number of passwords. It is possible to prevent input that is longer than X, or keep only X characters and notify the user that saved is only part of the password. The response APDU will return the location in the array where the password was stored. The example below shows the correct application flow, but it does not represent a sample of the current thesis project. The flowchart of this process is represented in Figure 9.

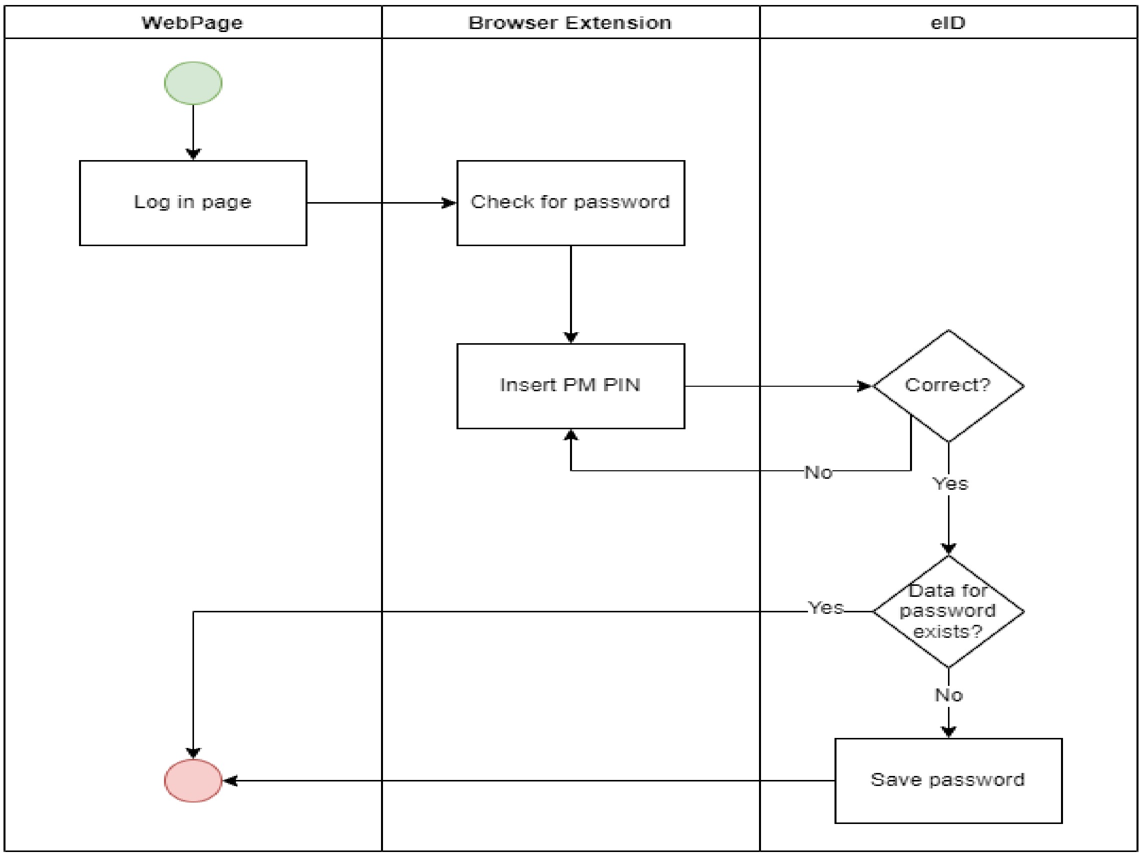
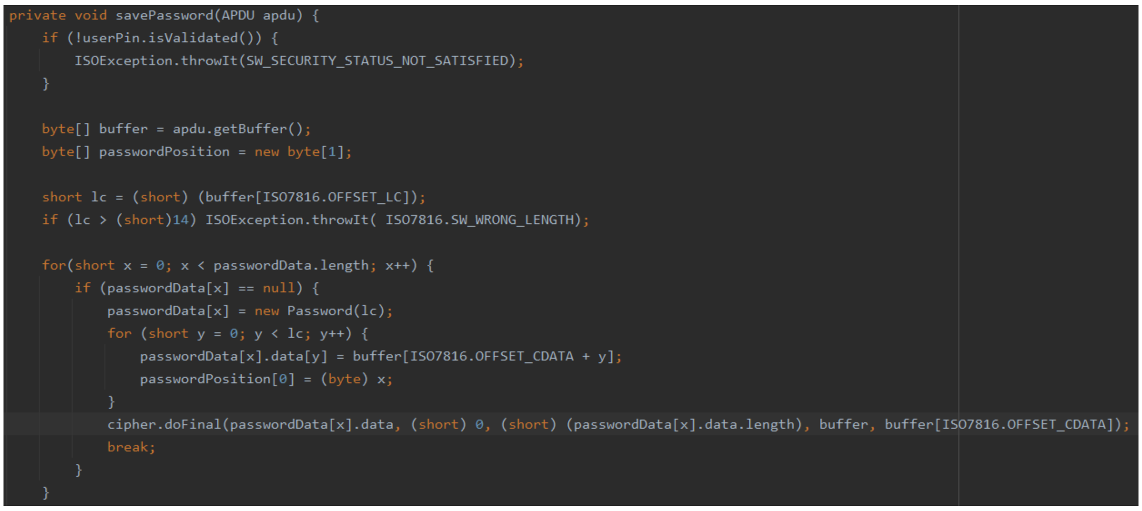


Figure 9. Saving password workflow.

The current thesis would use a simple example of storing the password that is in the APDU data field in an array and returning the positions in the array where that password is stored. The stored position is the first empty position available in the byte array. It is possible to send the browser extension the location of the password so that it can automatically request it for a particular field.



```java
private void savePassword(APDU apdu) {
    if (!userPin.isValidated()) {
        ISOException.throwIt(SW_SECURITY_STATUS_NOT_SATISFIED);
    }

    byte[] buffer = apdu.getBuffer();
    byte[] passwordPosition = new byte[1];

    short lc = (short) (buffer[ISO7816.OFFSET_LC]);
    if (lc > (short)14) ISOException.throwIt( ISO7816.SW_WRONG_LENGTH);

    for(short x = 0; x < passwordData.length; x++) {
        if (passwordData[x] == null) {
            passwordData[x] = new Password(lc);
            for (short y = 0; y < lc; y++) {
                passwordData[x].data[y] = buffer[ISO7816.OFFSET_CDATA + y];
                passwordPosition[0] = (byte) x;
            }
            cipher.doFinal(passwordData[x].data, (short) 0, (short) (passwordData[x].data.length), buffer, buffer[ISO7816.OFFSET_CDATA]);
            break;
        }
    }
```

Figure 10. Save password [50].

5.1.6 Retrieving Password

Password recovery is achieved by sending a command APDU with the position of the password in the array. This will return a specific password from an array. With the right approach, this communication should only be between the browser extension and the card. To simplify the example for the current thesis, the command will return an APDU with data to the terminal. The figure below shows how, theoretically, the relationship between a web page, a browser extension, and a document should be. The flowchart of this process is represented in Figure 11.

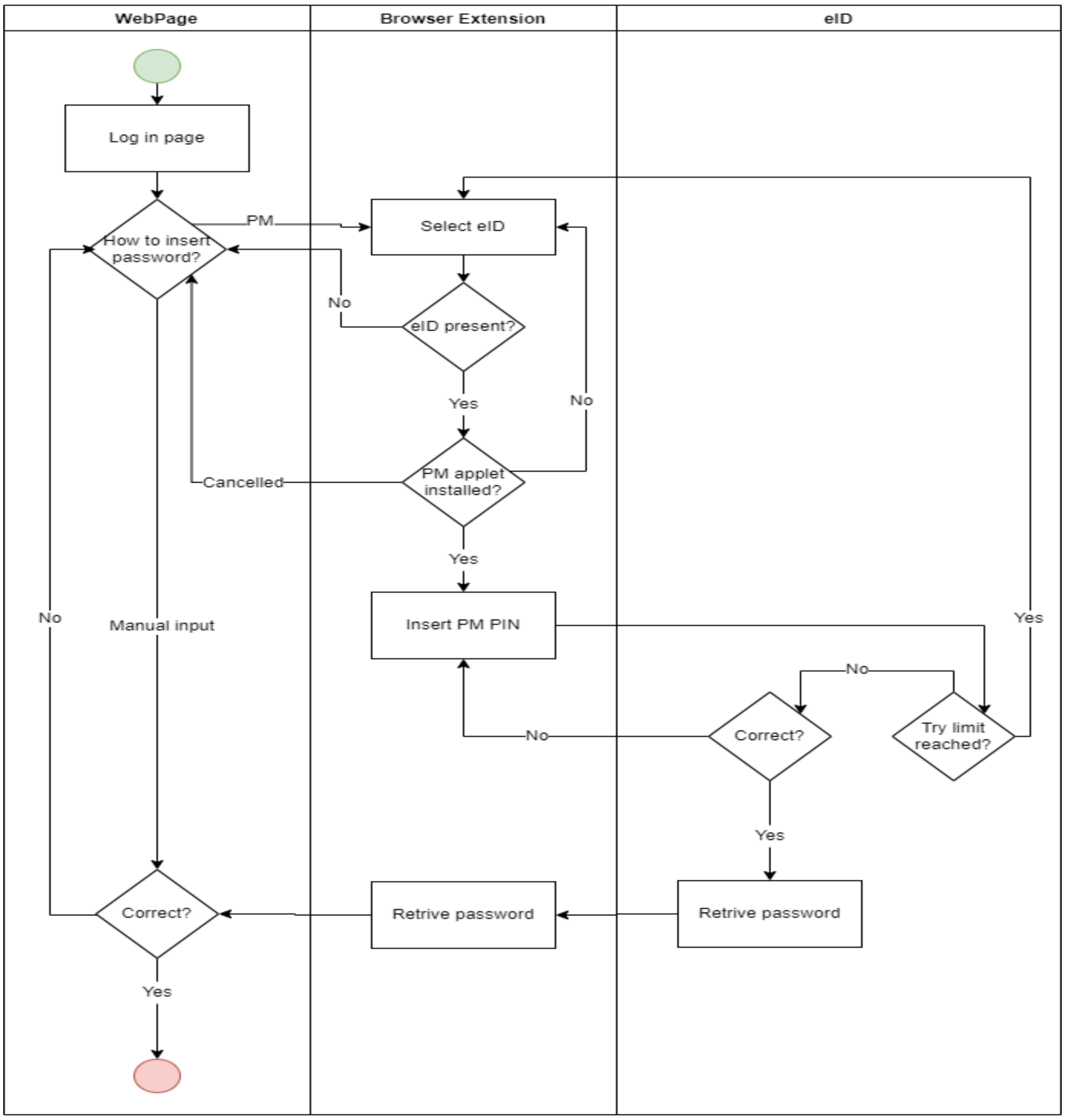


Figure 11. Getting a password from the smart card.

In the current thesis, the password lookup works by asking for a specific position in the array where the data should be stored. If the data is successfully found, then the password and code 9000 will be stored in the response. If a specific position in the array is empty, then the response code will be 6A88, and if the positions are outside the bonds, then the response will indicate that the data field is incorrect.

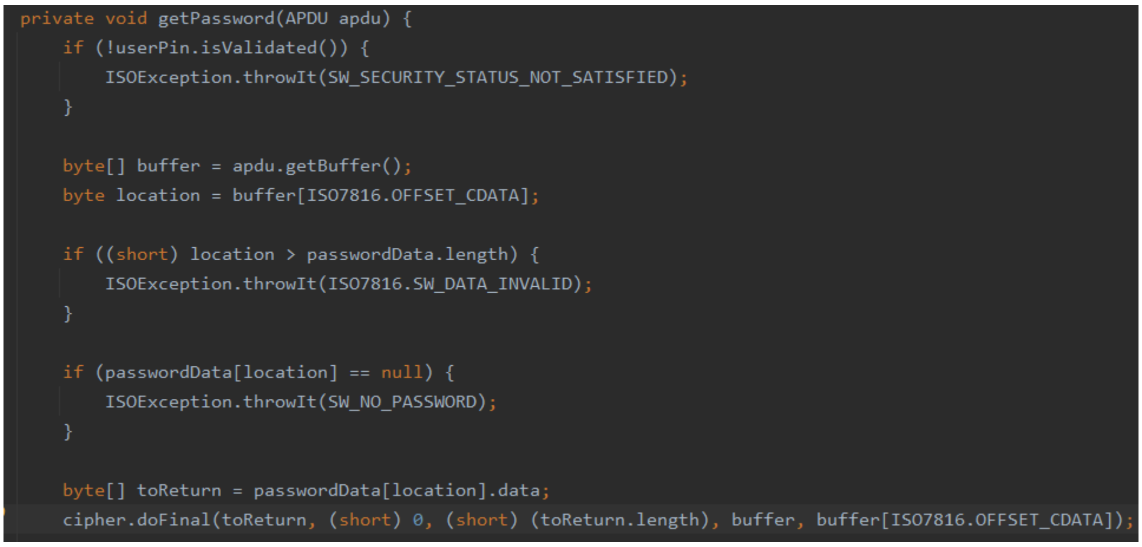

```java
private void getPassword(APDU apdu) {
    if (!userPin.isValidated()) {
        ISOException.throwIt(SW_SECURITY_STATUS_NOT_SATISFIED);
    }

    byte[] buffer = apdu.getBuffer();
    byte location = buffer[ISO7816.OFFSET_CDATA];

    if ((short) location > passwordData.length) {
        ISOException.throwIt(ISO7816.SW_DATA_INVALID);
    }

    if (passwordData[location] == null) {
        ISOException.throwIt(SW_NO_PASSWORD);
    }

    byte[] toReturn = passwordData[location].data;
    cipher.doFinal(toReturn, (short) 0, (short) (toReturn.length), buffer, buffer[ISO7816.OFFSET_CDATA]);
```

Figure 12. Retrieving password [50].

5.1.7 Updating Applet

Part of the application development is also maintenance and addition of the new functionality. Developers may want to improve their solution by fixing some bugs or adding new features. This means that the version installed on the card will need to be replaced with a newer one. That might lead to some potential issues such as data corruption or lack of backwards compatibility. If the password manager is updated, saved passwords may be lost. One possible solution is to create a backup of the repository and applet in case the update goes wrong. If the post-installation check fails, it would be possible to restore an older version. One of potential flows is shown in Figure 13.

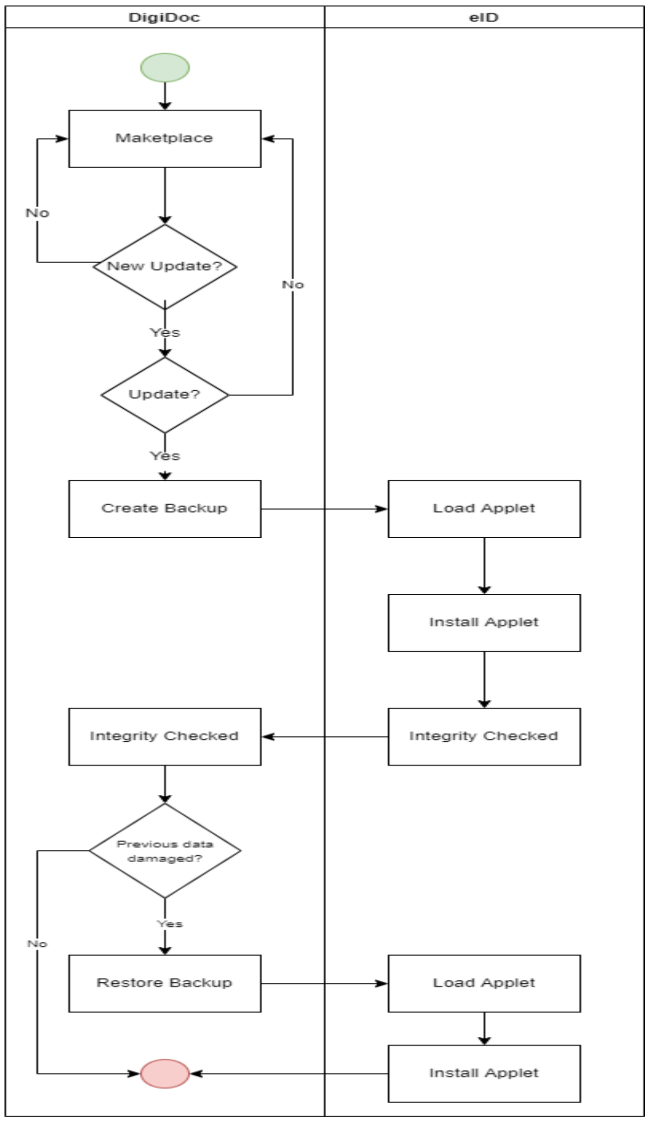


Figure 13. Updating installed applet.

5.1.8 Additional Functionality

Additional functionality allows simplifying testing the example on different data. In this case, operations with a PIN code and password removal are considered bonus functionality. PIN operations, such as change and reset, allow testing the user's ability to verify using values other than hard-coded ones. Resetting the PIN requires an administrator password, which in this example is hardcoded and created during installation. Deleting a password works similarly to getting a password, it needs the password position and returns the data at that position, but removes it from the array, allowing that array position to be used later for new data.

5.2 Development of the Applet

One of the goals of the current study is to show by example how applets can be installed via a trusted service on an Estonian national ID card. In this case, the final solution must be installed through the DigiDoc applet store, which is an application provided by the Estonian Information System Administration for Estonian ID card management. An example of the flow is shown in Figure 14.

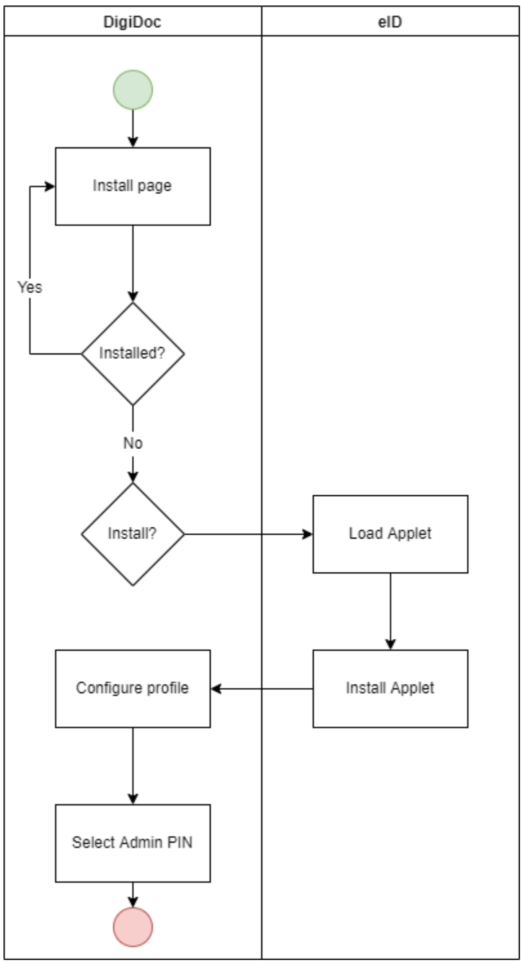


Figure 14. Installing the applet on the smart card.

Loading and installing the applet will be done in the current thesis with ARMIS and GlobalPlatformPro due to the lack of such functionality in DigiDoc. The following steps will be done:

1) Create a CAP file from the password manager Java application.

2) Load the signed CAP file on the smart card.

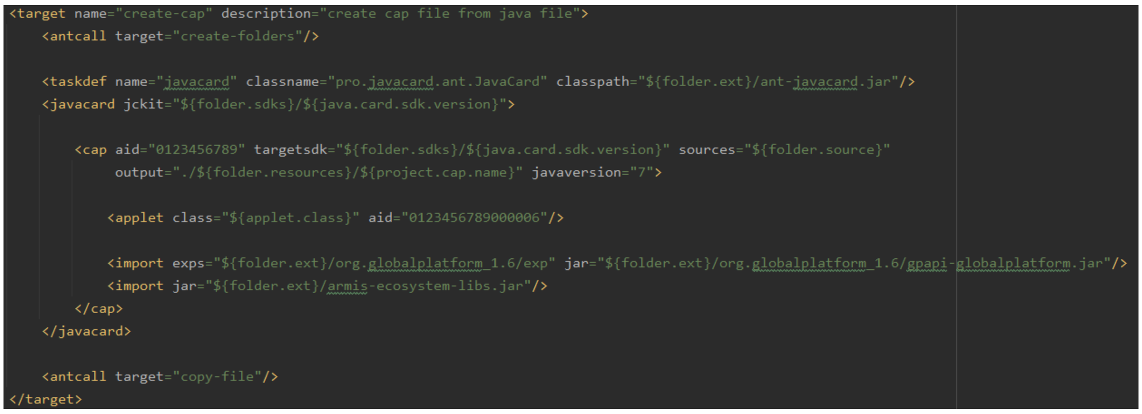
3) Install a loaded CAP file on the smart card.

5.2.1 Converting Java Application to CAP

After creating a working Java application that is compatible with JavaCard limitations, it can be converted to *.cap* format. There are several approaches to converting a Java application to CAP. This can be done with a specialized IDE such as the JavaCard Development Kit [38] or an IDE extension such as the JavaCard 3 Platform Development Kit, which can be installed in Eclipse [39], example.

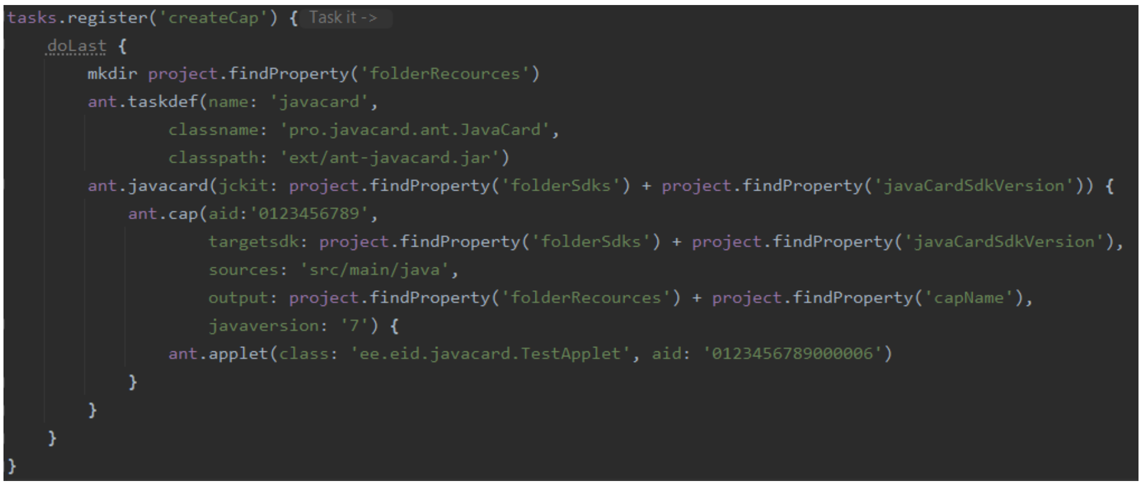
Both of these solutions also allow testing developed applications in a dummy environment without a real physical JavaCard. Another approach is to use an application developed by Martin Paljak called capfile.jar, which is available open source on GitHub under the MIT license. The application mentioned earlier and GlobalPlatformPro [40] will be the main tools used in the current document to load, install and test the JavaCard part of the project. This allows running commands through the terminal and is an easier approach than using a specific IDE.

As mentioned earlier, the main tool for the current thesis will be the GlobalPlatformPro tool. One of the tools associated with it, which is also associated with the same author, is ant-JavaCard.jar, which allows the creation of a CAP file from a Java file. It can be used in any java project, it requires a separate task in either Ant like in Figure 15 or Gradle like in Figure 16. As input, the version of the JavaCard bundle is required, the AID of the CAP file that will be visible after loading to the card, the AID of the applet that will be visible after installing on the card, and optionally the necessary `.jar` files can be imported, as in this example, this is GlobalPlatformPro 1.6 and the ARMIS ecosystem libraries.



```
<target name="create-cap" description="create cap file from java file">
    <antcall target="create-folders"/>

    <taskdef name="javacard" classname="pro.javacard.ant.JavaCard" classpath="${folder.ext}/ant-javacard.jar"/>
    <javacard jckit="${folder.sdks}/${java.card.sdk.version}">

        <cap aid="0123456789" targetsdk="${folder.sdks}/${java.card.sdk.version}" sources="${folder.source}"
             output="./${folder.resources}/${project.cap.name}" javaversion="7">

            <applet class="${applet.class}" aid="0123456789000006"/>

            <import exps="${folder.ext}/org.globalplatform_1.6/exp" jar="${folder.ext}/org.globalplatform_1.6/gpapi-globalplatform.jar"/>
            <import jar="${folder.ext}/armis-ecosystem-libs.jar"/>
        </cap>
    </javacard>

    <antcall target="copy-file"/>
</target>
```

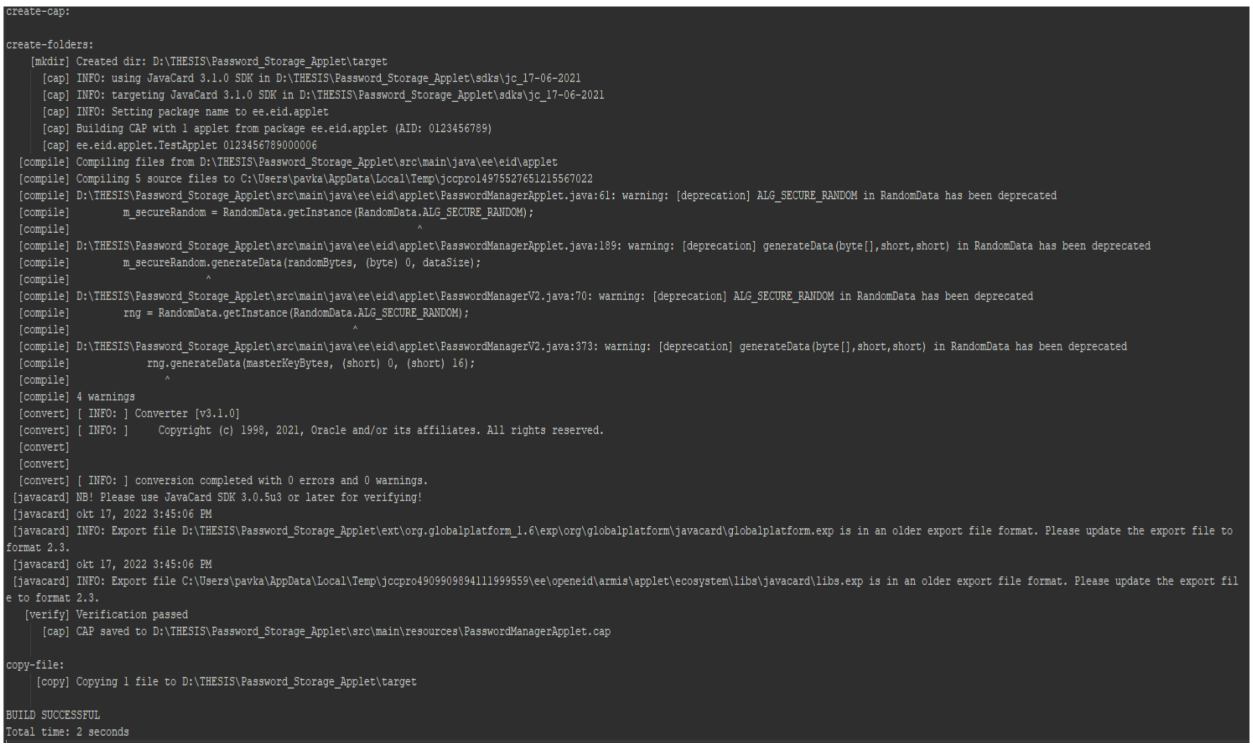
Figure 15. Ant example of creating a CAP file.



```
tasks.register('createCap') { Task it ->
    doLast {
        mkdir project.findProperty('folderRecources')
        ant.taskdef(name: 'javacard',
                classname: 'pro.javacard.ant.JavaCard',
                classpath: 'ext/ant-javacard.jar')
        ant.javacard(jckit: project.findProperty('folderSdks') + project.findProperty('javaCardSdkVersion')) {
            ant.cap(aid:'0123456789',
                    targetsdk: project.findProperty('folderSdks') + project.findProperty('javaCardSdkVersion'),
                    sources: 'src/main/java',
                    output: project.findProperty('folderRecources') + project.findProperty('capName'),
                    javaversion: '7') {
                ant.applet(class: 'ee.eid.javacard.TestApplet', aid: '0123456789000006')
            }
        }
    }
}
```

Figure 16. Gradle example of creating CAP file.

In addition, to make the conversion process easier, this tool also shows potential problems with the Java file being used. The figure below shows some warnings about deprecated functions. Warnings are usually ignored, and the CAP file is still generated, but this may help improve the applet in the future. The tool will also show errors that will block file conversion. An example of an error would be the incorrect use of an object or the import of libraries that are not supported by the JavaCard technology. This includes using lambda expressions or the enchanted switch statement. Exampled of warnings and errors are in Figure 17.



```
create-cap:

create-folders:
    [mkdir] Created dir: D:\THESIS\Password_Storage_Applet\target
      [cap] INFO: using JavaCard 3.1.0 SDK in D:\THESIS\Password_Storage_Applet\sdks\jc_17-06-2021
      [cap] INFO: targeting JavaCard 3.1.0 SDK in D:\THESIS\Password_Storage_Applet\sdks\jc_17-06-2021
      [cap] INFO: Setting package name to ee.eid.applet
      [cap] Building CAP with 1 applet from package ee.eid.applet (AID: 0123456789)
      [cap] ee.eid.applet.TestApplet 0123456789000006
  [compile] Compiling files from D:\THESIS\Password_Storage_Applet\src\main\java\ee\eid\applet
  [compile] Compiling 5 source files to C:\Users\pavka\AppData\Local\Temp\jccpro14975527651215567022
  [compile] D:\THESIS\Password_Storage_Applet\src\main\java\ee\eid\applet\PasswordManagerApplet.java:61: warning: [deprecation] ALG_SECURE_RANDOM in RandomData has been deprecated
  [compile]         m_secureRandom = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
  [compile]                                                          ^
  [compile] D:\THESIS\Password_Storage_Applet\src\main\java\ee\eid\applet\PasswordManagerApplet.java:189: warning: [deprecation] generateData(byte[],short,short) in RandomData has been deprecated
  [compile]         m_secureRandom.generateData(randomBytes, (byte) 0, dataSize);
  [compile]                       ^
  [compile] D:\THESIS\Password_Storage_Applet\src\main\java\ee\eid\applet\PasswordManagerV2.java:70: warning: [deprecation] ALG_SECURE_RANDOM in RandomData has been deprecated
  [compile]         rng = RandomData.getInstance(RandomData.ALG_SECURE_RANDOM);
  [compile]                                               ^
  [compile] D:\THESIS\Password_Storage_Applet\src\main\java\ee\eid\applet\PasswordManagerV2.java:373: warning: [deprecation] generateData(byte[],short,short) in RandomData has been deprecated
  [compile]             rng.generateData(masterKeyBytes, (short) 0, (short) 16);
  [compile]                ^
  [compile] 4 warnings
  [convert] [ INFO: ] Converter [v3.1.0]
  [convert] [ INFO: ]     Copyright (c) 1998, 2021, Oracle and/or its affiliates. All rights reserved.
  [convert]
  [convert]
  [convert] [ INFO: ] conversion completed with 0 errors and 0 warnings.
 [javacard] NB! Please use JavaCard SDK 3.0.5u3 or later for verifying!
 [javacard] okt 17, 2022 3:45:06 PM
 [javacard] INFO: Export file D:\THESIS\Password_Storage_Applet\ext\org.globalplatform_1.6\exp\org\globalplatform\javacard\globalplatform.exp is in an older export file format. Please update the export file to
format 2.3.
 [javacard] okt 17, 2022 3:45:06 PM
 [javacard] INFO: Export file C:\Users\pavka\AppData\Local\Temp\jccpro4909909894111999559\ee\openeid\armis\applet\ecosystem\libs\javacard\libs.exp is in an older export file format. Please update the export fil
e to format 2.3.
   [verify] Verification passed
      [cap] CAP saved to D:\THESIS\Password_Storage_Applet\src\main\resources\PasswordManagerApplet.cap

copy-file:
     [copy] Copying 1 file to D:\THESIS\Password_Storage_Applet\target

BUILD SUCCESSFUL
Total time: 2 seconds
```

Figure 17. An example of creating a CAP file.

5.2.2 Load, Install and Delete

All CAP files must be signed with a private key before they can be installed on a Supplementary Domain and used on an Estonian ID card. The load operation is possible without this step. The public key is stored on the card in the Issuer Security Domain, and during the installation operation, it checks to see if the correct private key has been used. If the security check fails, the applet will not install. For the test card, it is possible to add own public key to the card and sign the applet with the corresponding private key. In real life, only RIA will have access to the private key. This means that the applet will need to be sent to them first, and after all the necessary tests, it will be signed and added to the DigiDoc marketplace.

The best approach is to use ARMIS [49], the principles of which have been described in previous chapters. This will bypass the need for manual signing but will require placing the ARMIS applet on the smart card and having the ARMIS library for signing. This solution is much more secure as it ensures that the signing is performed by trusted parties. On the negative side, it takes longer to figure out how to use the tool and takes up memory from the card itself. Figure 18 shows the schema of the ARMIS system testing environment. The bottom of Figure 18 is the applet that will be added to the map using ARMIS. The central part represents parts of the ARMIS ecosystem, namely the ARMIS service, certificate management connected to the database and hardware security module, the GobalPaltform key service. The upper part represents the user's interaction after deploying an applet that is represented by DigiDoc4 connected to an Estonian ID card.

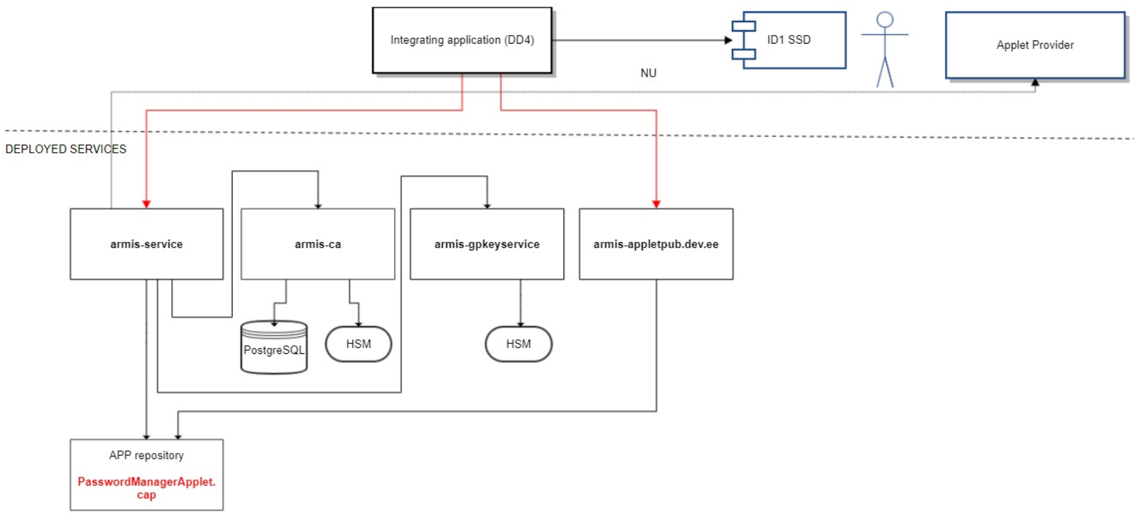


Figure 18. Schema of ARMIS system testing environment.

Since ARMIS is still in development and the documentation is being changed and updated, it is not yet possible to show examples of its use in real life. So in the current thesis, the focus will be to stick to a tool that can be used with a terminal and theoretical approach, which would be with ARMIS. This will limit the possible examples that can be tested in real life.

The applet was installed and added to the demo version of the DigiDoc marketplace demo through the RIA assistant. The theoretical approach to what the process looks like from their side has been described in previous chapters. For example, the current thesis and guide for GlobalPlatformPro and ARMIS used a test card that has the same profile as an Estonian ID. The keys for accessing the domains on the card are publicly available on official RIA web pages such as https://id.ee.

5.2.2.1 Installation with GlobalPlatformPro

As mentioned before to test the applet on the test card user needs to create RSA key and sign the CAP file with a tool like GlobalPlatformPro [40]. In the future, this step will be performed by ARMIS or manually by the RIA side. The current paper as an example will use a random key generated using the `OpenSSL` tool, as shown in Table 3. After that it can be signed with the `capfile.jar` tool due to some limitations, the required key can only be 1024 bytes.

Table 3. CAP file signing.

Commands	`openssl genrsa 1024 > key.pem`
	`java -jar capfile.jar -s key.pem PasswordManagerApplet.cap`

Output	`CAP file (v2.3), contains: applets for JavaCard 3.1.0`
	`Package: ee.eid.applet 0123456789 v0.0`
	`Applet:  ee.eid.applet.PasswordManagerApplet 0123456789000006`
	`Import:  A0000000620001                    v1.0 java.lang`
	`Import:  A0000000620101                    v1.8 JavaCard.framework`
	`Import:  A0000000620102                    v1.7 JavaCard.security`
	`Import:  A0000000620201                    v1.7 javacardx.crypto`
	`Generated by Oracle Corporation converter  [v3.1.0]`
	`On Mon Oct 17 18:34:19 EEST 2022 with JDK 14.0.1 (Oracle Corporation)`
	`Code size 4264 bytes (5319 with debug)`
	`SHA-256 ff91158d1958af3fc7eb7f61116e600d4e335a1dcfe9576b2e68acc500b4f57f`
	`SHA-1   8f29f4a1dbef298582dc8ee9378660eda25f760e`
	`Signed PasswordManagerApplet.cap`

After signing the CAP, the applet can be installed using the GlobalPlatformPro tool. There are two approaches how to do this. The first is to load first and then install. Or load and install at the same time. The result is the same. Starting by loading the CAP can help test how much memory it takes up and whether the load is working. Direct installation still loads the applet to the card, the difference is that there is no need to perform this step manually separately. Table 4 are shown the load and install commands with GlobalPaltformPro.

Table 4. Load and install CAP with GlobalPaltformPro.

Load command	`java -jar gp.jar --load PasswordManagerApplet.cap -to D233000000444F4D -dv -key XXX -kdf3`
Install after load command	`java -jar gp.jar -package 0123456789 -applet 0123456789000006 -create 0123456789000006 -dv -key XXX -kdf3`
Install without separate load command	`java -jar gp.jar --install PasswordManagerApplet.cap -to D233000000444F4D -dv -key XXX -kdf3`

5.2.2.2 Installation with ARMIS

To use ARMIS as a helper tool for installing and signing applets, it must first be placed in a domain where other applets will be installed in the future. In the case of the Estonian ID card, this is the Supplementary Domain, which exists separately from the main domain, where all personal data and certificates are placed. For this part, footage from RIA was used.

Now, if the applet is created with the correct libraries with the ARMIS ecosystems, the correct keys, and packaged in a CAP file, then it can be placed on the ID card. Installation works through the manager and can be achieved with a separately created jar file that uses the GlobalPlatformPro and ARMIS ecosystems to communicate with the manager on the card. Table 5 is shown how the install command with ARMIS looks like if ARMIS is hosted locally.

Table 5. CAP install with ARMIS.

```
java -jar cras-client-1.1.45-exec.jar
--pin1=1234
--cras.url=localhost
--cras.port=30443
--cras.tls.protocols=TLSv1.3
–creator
class=ee.openeid.armis.common.creators.ArmisStartRpcRequestCreator
--cap-file-path=PasswordManagerApplet.cap
--applet-aid=0123456789000006
--operation=INSTALL
```

After installation, it is possible to check the memory and existing applets on the card in the same way as described in Chapter 5.2.2.1 using the GlobalPlatformPro commands. The main difference will be that this time ARMIS will be in the list of installed applets, and there will be much less memory since the manager takes up a decent amount of space.

5.2.3 Communication with the Card

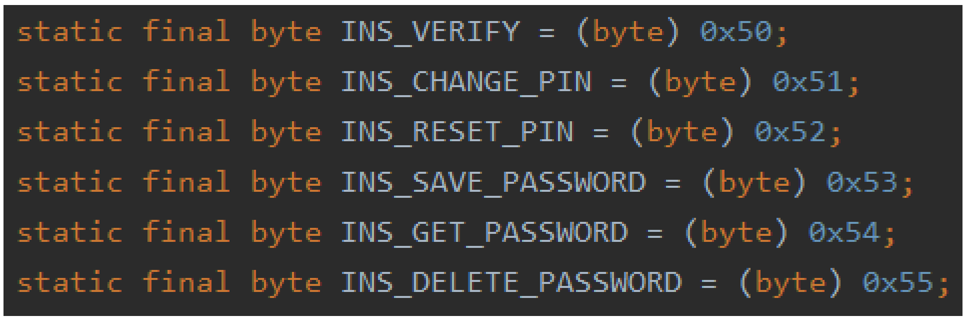
To standardize communication with JavaCard, the ISO 7816 standard is used. It shows the use of APDU commands, which can be of two types: request and response. The base command has the following fields:

1) CLA – class byte
2) INS – instruction byte
3) P1 and P2 – parameter bytes
4) L_C – length of the data field
5) Data Field – data to send to the card or to get from the card

In the case of communication between a Java application and a smart card, drivers are required to establish communication between the card reader and the computer. The best example of a Java library might be smart cardio which solves these problems. A similar approach is used in the case of communication with the browser. Solutions such as the DigiDoc browser extension are one such example.

5.2.4 Command and Response APDU-s

All commands that an applet responds to are specified during development. As mentioned earlier, INS commands are used to specify which function to use in an applet. In the current project, P1 and P2 are not specified, and CLA cannot be B0, which means that P1 and P2 can be any value. Although the link requires a valid DATA field, if a particular function requires input in the current project, all functions must not be empty in the DATA field. Figure 19 shows what INS are used for the project.



```
static final byte INS_VERIFY = (byte) 0x50;
static final byte INS_CHANGE_PIN = (byte) 0x51;
static final byte INS_RESET_PIN = (byte) 0x52;
static final byte INS_SAVE_PASSWORD = (byte) 0x53;
static final byte INS_GET_PASSWORD = (byte) 0x54;
static final byte INS_DELETE_PASSWORD = (byte) 0x55;
```

Figure 19. Configured INS [50].

Using an applet starts by selecting it on the JavaCard. The selection uses INS, P1, and P2 specified by the Global Platform standard. To select the developed Password Manager must be used the Command APDU `00 A4 04 00 08 01 23 45 67 89 00 00 06 00`. After selecting the correct applet, it is possible to send customized commands to it. Table 6 is shown the flow of commands to test the operability of the developed project.

Table 6. Password manager communication flow.

Verify PIN	`00 50 00 00 04 00 00 00 00 00`	`Response 9000`
Change PIN	`00 51 00 00 04 01 02 03 04 00`	`Response 9000`
Verify New PIN	`00 50 00 00 04 01 02 03 04 00`	`Response 9000`
Reset PIN	`00 52 00 00 05 00 00 00 00 00 00`	`Response 9000`
Verify New PIN	`00 50 00 00 04 01 02 03 04 00`	`Response 6900`
Verify Reset PIN	`00 50 00 00 04 00 00 00 00 00`	`Response 9000`
Save Password	`00 53 00 00 05 12 34 56 78 90 00`	`Response 00 9000`

Retrieve Password 0	`00 54 00 00 01 00 00`	`Response 12 34 56 78 90 9000`
Save Password	`00 53 00 00 05 55 44 33 22 11 00`	`Response 01 9000`
Retrieve Password 1	`00 54 00 00 01 01 00`	`Response 55 44 33 22 11 9000`
Retrieve Password 0	`00 54 00 00 01 00 00`	`Response 12 34 56 78 90 9000`
Delete Password 0	`00 55 00 00 01 00 00`	`Response 12 34 56 78 90 9000`
Retrieve Password 1	`00 54 00 00 01 01 00`	`Response 55 44 33 22 11 9000`
Retrieve Password 0	`00 54 00 00 01 00 00`	`Response 68AA`

5.3 Testing

To make sure that the installed applet did not damage the contents of the card and works as intended, a series of tests must be carried out. Tests should be conducted by the applet's creators, as well as government officials responsible for handling electronic identifiers. In the current paper, will be provided an example of a three-step testing approach that can help validate a national ID card after installing an applet.

5.3.1 Applet Functionality

Applet owners must ensure that their applet works as intended. This includes unit tests that control the behavior of the applet itself not only in a mocked environment but also on a smart card with the same profile as the Estonian national identity card. In the password manager applet example, it must complete the following steps:

1. Install on a smart card.
2. Store passwords in a secure vault created by the applet.
3. Get the passwords from the vault to use them on the web page.
4. Passwords cannot be accessed without a user PIN.

It is the responsibility of the applet owners to complete these steps, as they need to know better how their applet behaves. The main security issue in this situation will be the use of technology and the approach to storing data on a smart card. With incompetent implementations, some problems may show up later. Some of these issues may be noticed in the later stages of testing.

5.3.2 Electronic Document Functionality

Government officials responsible for electronic IDs must ensure that features that worked before the applet was installed continue to work. The basic functionality of an ID card is the reason for its usefulness and security and cannot be traded for any nice extras. One potential problem that may arise during installation is the triggering of security features that can wipe data from the card, which might cause problems with existing functionality. Also, users must be protected in case the developed applet has hidden malicious functionality. Here are some of those important parts:

1. Encryption and decryption procedures.
2. Memory status before and after installation.
3. Usage of the authentication methods that include the usage of PIN1, PIN2, and PUK.

The RIA ID card department should already have the knowledge and experience to check documents before they can be used. Using the example of the Estonian ID card, there are already test tools that check the cards against the basic profile to see any differences in the structure or workflow of the basic procedures.

5.3.3 Electronic Document Integrity

Finally we look at the the integrity of the document part. It can be tested in parallel with the functionality, but it should be a separate checkmark in the list. Since the electronic document also contains information about the owner, this information should not be changed in any way. Changes to some data on the card can be made in some special cases, but this should not be done using a third-party application. The steps to be followed will check:

1. Personal data integrity.

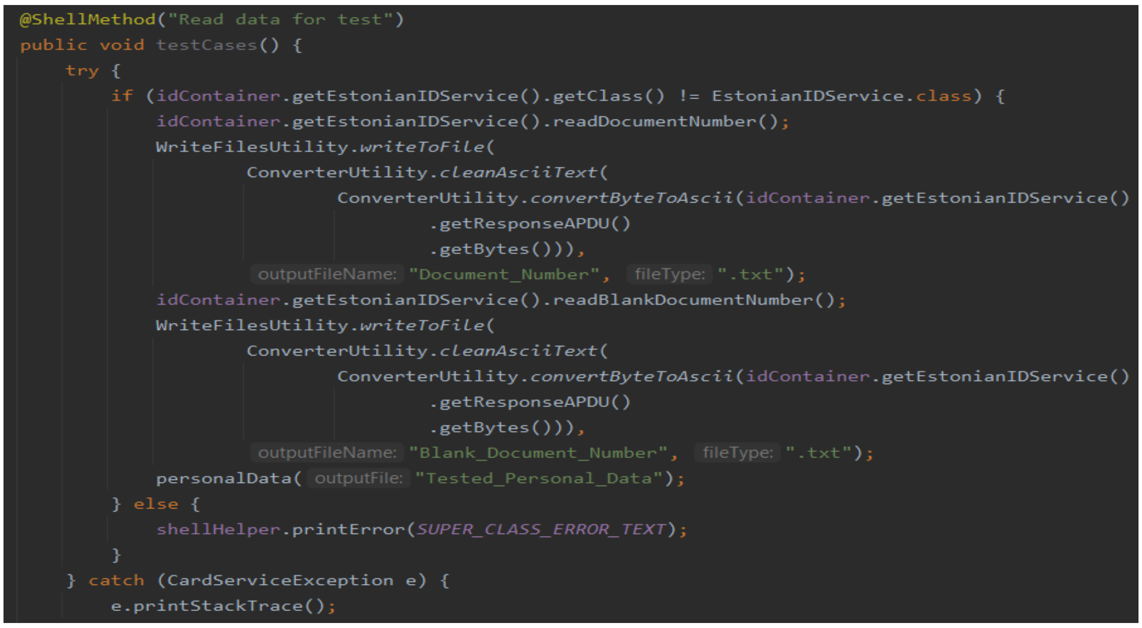
2. Authentication and signature certificate integrity.

3. Existence of previously installed applets.

As a rule, data that was already in the document should remain there unchanged. The electronic identity provider may be responsible for this part. This part can also potentially be tested by the DigiDoc application after the applet is installed.

5.3.4 Identity Testing Tool

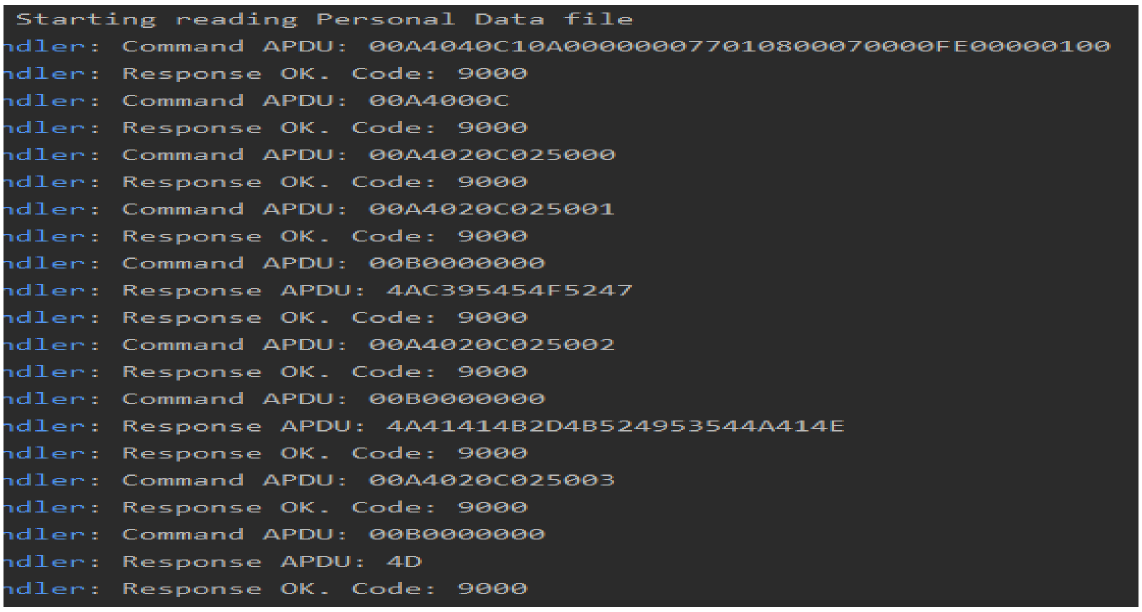
The best approach for testing is to use ready-made tools. One of the tools that was created to check the reading of data on a smart card is the Identity Testing Tool. It was a Bachelor’s thesis project. It is capable of reading data through contact and non-contact interfaces. The project was created for Estonian ID card versions before and after 2018. The tool automatically checks if the card's ATR is recognized and creates a special profile for the correct card. Commands for communication through the integrated shell [45]. Figure 20 shows an example test that reads data and saves the results to separate files.



```java
@ShellMethod("Read data for test")
public void testCases() {
    try {
        if (idContainer.getEstonianIDService().getClass() != EstonianIDService.class) {
            idContainer.getEstonianIDService().readDocumentNumber();
            WriteFilesUtility.writeToFile(
                    ConverterUtility.cleanAsciiText(
                            ConverterUtility.convertByteToAscii(idContainer.getEstonianIDService()
                                    .getResponseAPDU()
                                    .getBytes())),
                     outputFileName: "Document_Number",  fileType: ".txt");
            idContainer.getEstonianIDService().readBlankDocumentNumber();
            WriteFilesUtility.writeToFile(
                    ConverterUtility.cleanAsciiText(
                            ConverterUtility.convertByteToAscii(idContainer.getEstonianIDService()
                                    .getResponseAPDU()
                                    .getBytes())),
                     outputFileName: "Blank_Document_Number",  fileType: ".txt");
            personalData( outputFile: "Tested_Personal_Data");
        } else {
            shellHelper.printError(SUPER_CLASS_ERROR_TEXT);
        }
    } catch (CardServiceException e) {
        e.printStackTrace();
```

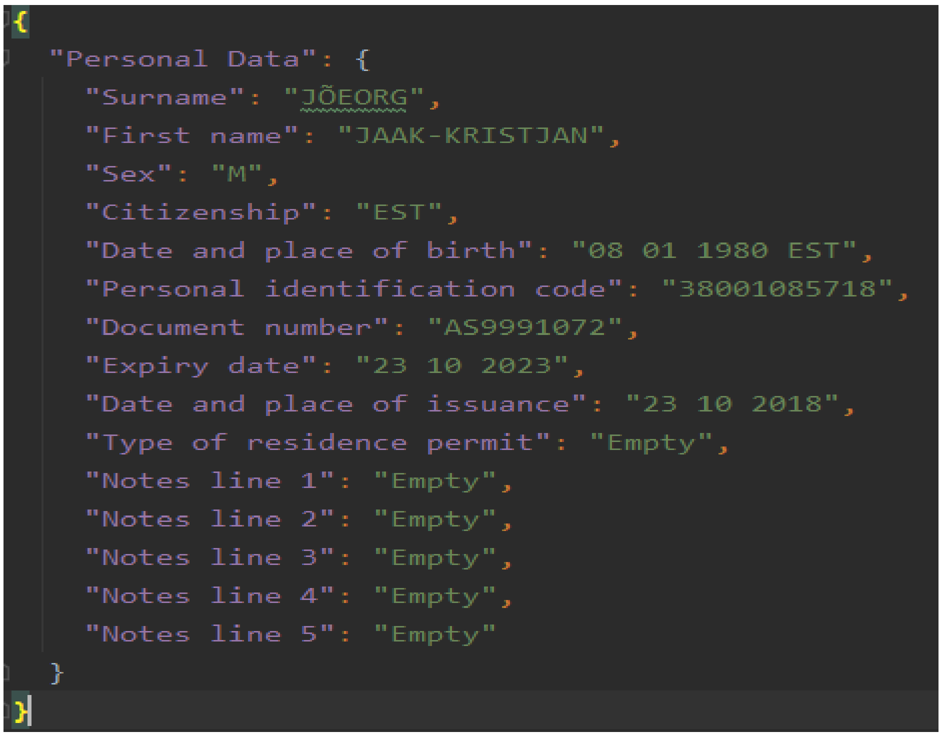
Figure 20. Sample test in Identity Testing Tool.

This will show the output of the connection between the card and the tool in the form of APDUs in HEX format as shown in Figure 21. Also, if the test is successful, the results are converted to ASCII and saved to a separate JSON file as shown in Figure 22. Communication with the card is continuous and allows us to perform other tests, such as certificates check or signature performance. The examples below were run with the same card that hosted the applet.



```
 Starting reading Personal Data file
ndler: Command APDU: 00A4040C10A000000077010800070000FE00000100
ndler: Response OK. Code: 9000
ndler: Command APDU: 00A4000C
ndler: Response OK. Code: 9000
ndler: Command APDU: 00A4020C025000
ndler: Response OK. Code: 9000
ndler: Command APDU: 00A4020C025001
ndler: Response OK. Code: 9000
ndler: Command APDU: 00B0000000
ndler: Response APDU: 4AC395454F5247
ndler: Response OK. Code: 9000
ndler: Command APDU: 00A4020C025002
ndler: Response OK. Code: 9000
ndler: Command APDU: 00B0000000
ndler: Response APDU: 4A41414B2D4B524953544A414E
ndler: Response OK. Code: 9000
ndler: Command APDU: 00A4020C025003
ndler: Response OK. Code: 9000
ndler: Command APDU: 00B0000000
ndler: Response APDU: 4D
ndler: Response OK. Code: 9000
```

Figure 21. HEX format output.



```
{
  "Personal Data": {
    "Surname": "JÕEORG",
    "First name": "JAAK-KRISTJAN",
    "Sex": "M",
    "Citizenship": "EST",
    "Date and place of birth": "08 01 1980 EST",
    "Personal identification code": "38001085718",
    "Document number": "AS9991072",
    "Expiry date": "23 10 2023",
    "Date and place of issuance": "23 10 2018",
    "Type of residence permit": "Empty",
    "Notes line 1": "Empty",
    "Notes line 2": "Empty",
    "Notes line 3": "Empty",
    "Notes line 4": "Empty",
    "Notes line 5": "Empty"
  }
}
```

Figure 22. JSON format example.

5.3.5 Test Generation with Verification Technology

Another option to test the functionality of the applet is to use tools like TGV. Test Generation with Verification technology in short TGV is a prototype for creating conformance test suites for protocols. It is based on the I/O Transition Systems (IOLTS) model and uses algorithms derived from validation technology. The specification and test objectives are required as input. The specification can be expressed as a Labelled Transition System (LTS) and it is used to generate an abstract test case [43].

JavaCard requires that all features be valid and that all cases of misuse are covered. For testing purposes, each expected behavior and abuse should be tested. The test cases generated by TGV are in a non-executable text format and are as abstract as the specification. There are two steps to perform these tests. The first step is to reduce the level of abstraction and then convert the test cases to a programming language like Java [43].

The goal of testing is an abstract definition of a subset of the definition that allows behavioral choice to be tested and thus allows for reduced specification testing. The final states of the test goal graph are acceptance or failure states. If the test goal is valid the TGV produces an LTS, which is a description of the test graph in BCG or AUT format [44].

5.4 Browser Extension and Backend logic

A browser extension is required to establish a link between the smart card and the web page. The extension will work as a bridge between the two. This should solve a few issues, such as the designed input fields, but the main one is the connection between the backend and the web browser. The most common approach is to a use Chromium browser extension.

Developing an extension requires implementing HTML and CSS for the input fields. This includes a field for entering a user PIN, a field to check available passwords, and other fields to perform functionality that was implemented for the applet like storing the password on the smart card, and the ability to retrieve the password from the smart card. Figure 23 shows examples of the fields.

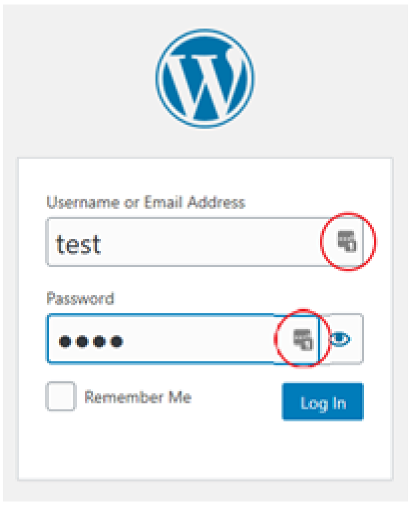
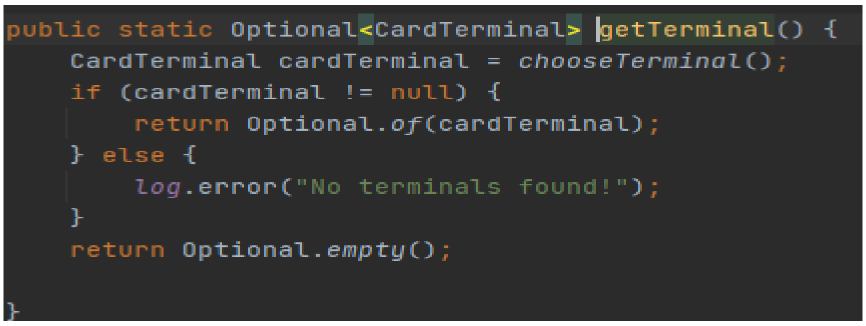


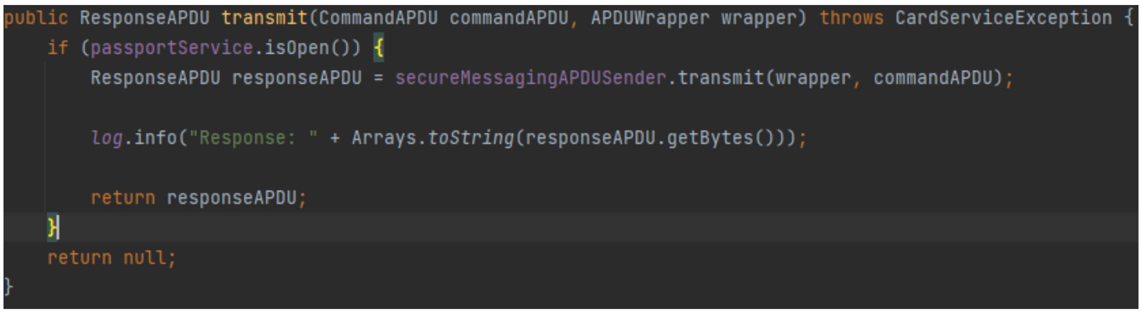
Figure 23. LastPass password selection.

The backend program will receive data from the input fields of the browser extension. The program must be installed along with a browser extension to be able to communicate with the card. In the current example, it was created in Java with basic functionality to connect a terminal, as shown in Figure 24, send command APDU to the card as shown in Figure 25, and respond via REST API calls configured in the application controller that performs card operations such as sending an APDU, as shown in Figure 26. The controller will be able to receive GET, POST, and DELETE commands.



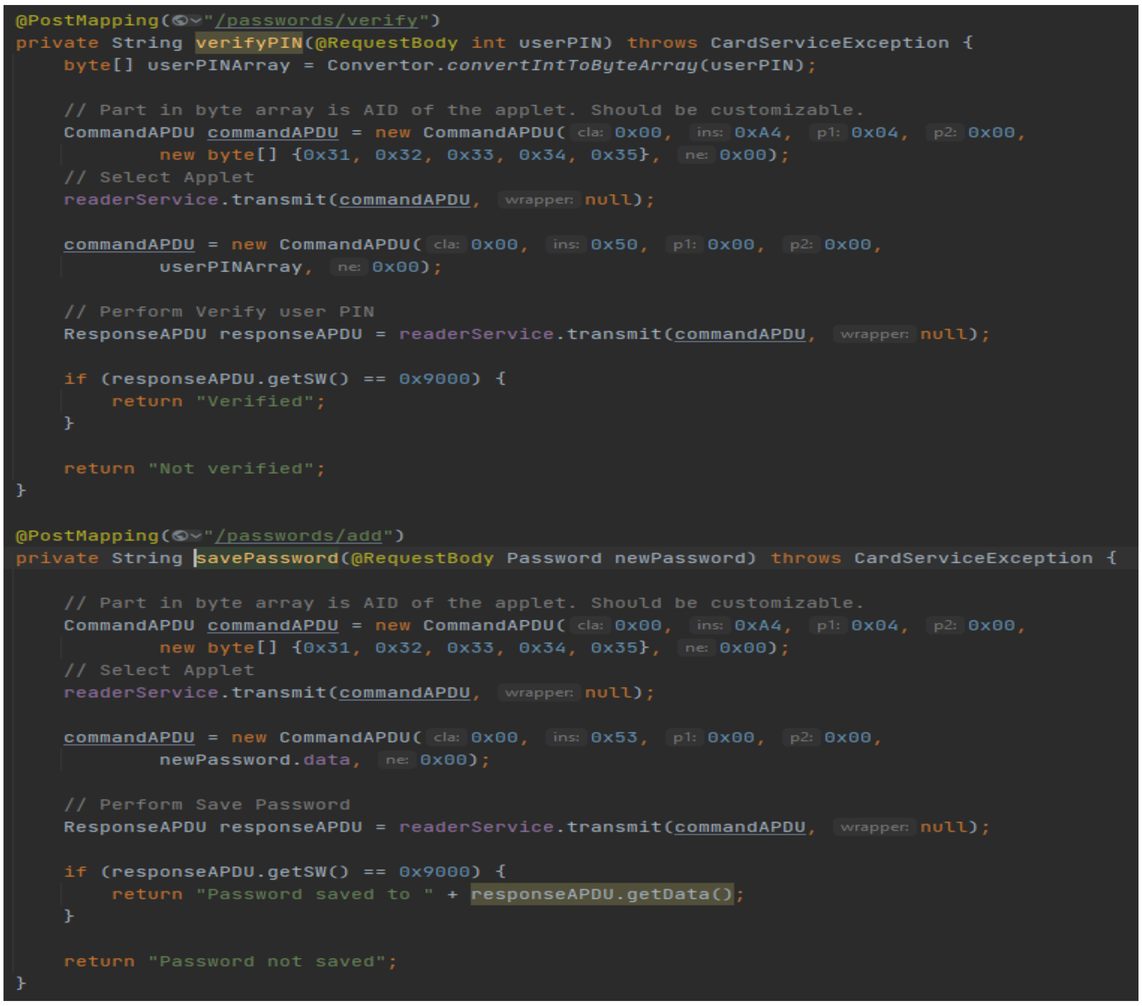
```java
public static Optional<CardTerminal> getTerminal() {
    CardTerminal cardTerminal = chooseTerminal();
    if (cardTerminal != null) {
        return Optional.of(cardTerminal);
    } else {
        log.error("No terminals found!");
    }
    return Optional.empty();

}
```

Figure 24. Selecting card terminal [53].



```java
public ResponseAPDU transmit(CommandAPDU commandAPDU, APDUWrapper wrapper) throws CardServiceException {
    if (passportService.isOpen()) {
        ResponseAPDU responseAPDU = secureMessagingAPDUSender.transmit(wrapper, commandAPDU);

        log.info("Response: " + Arrays.toString(responseAPDU.getBytes()));

        return responseAPDU;
    }
    return null;
}
```

Figure 25. Transmitting command APDU [53].



```java
@PostMapping("/passwords/verify")
private String verifyPIN(@RequestBody int userPIN) throws CardServiceException {
    byte[] userPINArray = Convertor.convertIntToByteArray(userPIN);

    // Part in byte array is AID of the applet. Should be customizable.
    CommandAPDU commandAPDU = new CommandAPDU( cla: 0x00,  ins: 0xA4,  p1: 0x04,  p2: 0x00,
            new byte[] {0x31, 0x32, 0x33, 0x34, 0x35},  ne: 0x00);
    // Select Applet
    readerService.transmit(commandAPDU,  wrapper: null);

    commandAPDU = new CommandAPDU( cla: 0x00,  ins: 0x50,  p1: 0x00,  p2: 0x00,
            userPINArray,  ne: 0x00);

    // Perform Verify user PIN
    ResponseAPDU responseAPDU = readerService.transmit(commandAPDU,  wrapper: null);

    if (responseAPDU.getSW() == 0x9000) {
        return "Verified";
    }

    return "Not verified";
}

@PostMapping("/passwords/add")
private String savePassword(@RequestBody Password newPassword) throws CardServiceException {

    // Part in byte array is AID of the applet. Should be customizable.
    CommandAPDU commandAPDU = new CommandAPDU( cla: 0x00,  ins: 0xA4,  p1: 0x04,  p2: 0x00,
            new byte[] {0x31, 0x32, 0x33, 0x34, 0x35},  ne: 0x00);
    // Select Applet
    readerService.transmit(commandAPDU,  wrapper: null);

    commandAPDU = new CommandAPDU( cla: 0x00,  ins: 0x53,  p1: 0x00,  p2: 0x00,
            newPassword.data,  ne: 0x00);

    // Perform Save Password
    ResponseAPDU responseAPDU = readerService.transmit(commandAPDU,  wrapper: null);

    if (responseAPDU.getSW() == 0x9000) {
        return "Password saved to " + responseAPDU.getData();
    }

    return "Password not saved";
}
```

Figure 26. REST API example [53].

The names of the REST API functions in the controller are the same as those in the applet. Each function first selects an applet and then tries to perform a certain operation with the correct APDU where the INS field is changed and the corresponding DATA field input. Figure 25 shows the `verifyPIN` and `savePassword` operations as an example. For simplicity, responses from the API will give general string feedback. Also, each operation has its endpoint. Used endpoints for POST are `/passwords/verify`, `/passwords/change`, `/passwords/reset`, and `/passwords/add`. For GET /passwords/{position}. For DELETE `/passwords/{position}`.

5.5 Outcomes of the Practical Part

The result of the practical part partially corresponds to the objectives of the thesis. Some topics have received more theoretical research from a practical side, but proofs of concepts have also been created. The theoretical approach has received tools that are still in development and cannot be used to their full potential. For example, the ARMIS project is still in the early stages of development and, as at the time of completing the thesis, has slowed down due to the prioritization of various other projects in the RIA.

The first achievement is a working example of an applet installation guide. The guide provides a step-by-step approach to loading, installing, and interacting with the applet. These examples show flows to place software with GlobalPlatformPro and the ARMIS applet manager. Both solutions can be used for testing purposes in the future if the development of the applet is more in demand specifically for the Estonian identity card.

The second result is a working simple Password Manager logic that can be placed on an Estonian ID card. Native JavaCard libraries were used for development, with some examples of how to use the same libraries with the ARMIS ecosystem in the future. The password manager itself can be placed on the Estonian ID card, and its functions are available when communicating with the card.

For browser extension a REST API was developed. A browser extension was required to show how the link between the card password manager applet and the login screen could work. Test cases were simulated in the terminal with APDU commands and analyzed to see if the responses were the same as those encoded in the applet. For the current part, a theoretical approach to what is needed and a potential visualization were mainly provided. The GUI from LastPass was used as an example design. Applets that do not need additional dependencies on additional extensions should be the focus in the future.

The potential flow of testing an applet with a part of a document and how to divide the responsibilities between the parties. The main applet development is intended for private companies that would like to place their product on the Estonian ID card. This means that the main testing of the applet is on the side of the developers. And since the RIA is responsible for the integrity of the electronic document, their side should be testing the functionality of an existing part. The Identity Testing Tool and the Test Generation with Verification were used as examples of potential testing tools.

The current study completed a more hands-on approach to Gregor Johannson's study mentioned in Chapter 2.2. As his work explored more potential future uses and developments of the Estonian ID card, the current thesis also presented an updated version using some known technologies and potential new ones.

5.6 Future Work

For future research, it is possible to develop some kind of solution that may find its way into the market when the development of the ARMIS ecosystem will be completed, and RIA will work out how to support the integration process with their application. With the introduction of biometrics in the ID card, there may be also some research and advancement in that direction too.

The list of potential new developments in the field of applets for ID cards may include the use of the NFC channel. Although there are currently some limitations on how this can be used on an Estonian ID card, the potential is still great. This could include solutions like the bus ticket applet or support of banking operations, which could also include crypto wallets in the future.

Other more suitable JavaCard solutions have already been developed and could be explored for integration with the Estonian ID card. Such examples could be FIDO for multiple devices, One Time Password (OTP) or FIDO2. In October 2022, Martin Paljak released the FIDO2 toolkit along with the X-FIDO application, which could be configured to work also with an Estonian ID card [54]. The advantage of these technologies may also be in the fact that they do not need additional dependencies like browser extensions.

6 Summary

The purpose of the current thesis is to provide 3rd parties with a practical example of how to develop JavaCard applets that can be made to run in a separate security domain of a modern Estonian ID card. The current work can also be seen as a continuation of Gregor Johannson's thesis, which explored application development for the Estonian ID card on a more theoretical level. The thesis builds on published research on existing password manager technologies.

The chapters on the electronic identity card and JavaCard are devoted to the history of technology and real examples of solutions. Electronic identity cards that comply with ICAO and Global Platform standards use JavaCard technologies. This includes also the Estonian ID card, which uses all the security features that JavaCard has to offer. This is true for the Estonian ID card version after the year 2018. The same technologies are used in the chips found in SIM cards and credit cards. Historically, this technology has not been so widely used for a certain period due to limited access to low-level cryptographic primitives and the lack of some common data types. With the development of technology, the development of applets has also increased.

The chapter on password managers briefly introduces the technology and the different implementations. The main purpose of that chapter is to show the importance of password managers and their advantages and disadvantages. That part gives a better idea of what a proper implementation of the password storage applet should look like. While passwords are still widely used, they are starting to become obsolete. The chapter also provides examples of alternative security solutions such as token-based access and one-time passwords. In the current thesis, an applet was created to store passwords, which will be available when the ID card is connected to the device.

The last chapter is devoted to the practical part of how to create a working applet with functionality, load and install it on the card, and the tools that can be used for this, such as ARMIS and GlobalPlatformPro. That chapter also gives a brief overview of how to test these solutions and how to build a proper password manager with a browser extension. That chapter has some limitations because ARMIS is still in development and there is no publicly available documentation for the current project. The reasons for this may be related to the fact that that project is currently on hold. The main value of the chapter is an example of the process of loading and installing an applet with a working solution on an Estonian electronic identity card. The result of that chapter is that it offers a good starting point for simpler projects with JavaCard.

Additional functionality in the form of a password manager, which was developed for the current thesis, is not a needed addition to electronic documents. Although, when considering what applet development looked like and technologies that could help manage these applets in the future, there is potential to develop useful applets that can add value to an electronic document. Some of the examples shown in the current thesis can give a glimpse of the potential that electronic documents can provide in addition to their basic functionality.

In conclusion, all the planned results have been achieved in the current thesis with room for future research and development. The current thesis successfully continued the work of Gregor Johannson's thesis. The research started in the current thesis can be continued by introducing an applet with more commercial uses, development, or integration of the existing solutions with Estonian ID and utilizing the ARMIS ecosystem when its development will be completed.

References

[1] De Guzman, F.E., Gerardo, B.D. & Medina, R.P. 2019, "Implementation of enhanced secure hash algorithm towards a secured web portal", 2019 IEEE 4th International Conference on Computer and Communication Systems, ICCCS 2019, pp. 189.

[2] Polpong, J. & Wuttidittachotti, P. 2020, "Authentication and password storing improvement using SXR algorithm with a hash function", International Journal of Electrical and Computer Engineering, vol. 10, no. 6, pp. 6582-6591.

[3] Singh, A. & Raj, S. 2019, "Securing password using dynamic password policy generator algorithm", Journal of King Saud University - Computer and Information Sciences.

[4] "Smart card programming and security | springerlink." [Online]. Available: https://link.springer.com/book/10.1007/3-540-45418-7?page=2. [Accessed: 05-Nov-2021].

[5] S. K. H. Islam, "Design and analysis of an improved smart card-based Remote User Password Authentication Scheme," Wiley Online Library, 21-Apr-2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/dac.2793. [Accessed: 05-Nov-2021].

[6] Meshram, C., Ibrahim, R.W., Deng, L., Shende, S.W., Meshram, S.G. & Barve, S.K. 2021, "A robust smart card and remote user password-based authentication protocol using extended chaotic maps under smart cities environment", Soft Computing, vol. 25, no. 15, pp. 10037-10051.

[7] S. Robinson, B. Narayanan, N. Toh, and F. Pereira, "Methods for pre-processing smart card data to improve data quality," Transportation Research Part C: Emerging Technologies, 07-Nov-2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X14002988. [Accessed: 05-Nov-2021].

[8] G. Johannson, “Tehnilised eeltingimused Kolmanda osapoole rakenduste lubamiseks uuel Eesti ID-Kaardil,” Master’s thesis, Avaleht - TalTech raamatukogu digikogu, 05-Jun-2019. [Online]. Available: https://digikogu.taltech.ee/et/item/64c83d8f-8f2d-4311-b548-b07c9b58a6cb. [Accessed: 05-Nov-2021].

[9] G. Haugas, “Proaktiivsete Teenuste Võimalikkus ID-Kaardi Näitel,” Master’s thesis, Avaleht - TalTech raamatukogu digikogu, 04-Jun-2020. [Online]. Available: https://digikogu.taltech.ee/et/item/c35892e2-f289-4970-b0e9-6568e6ec6a42. [Accessed: 05-Nov-2021].

[10] M.-L. Palginõmm and E. Karo, “Eesti ID-kaardi ja selel elektroonilise kasutuse Levik - eduloo tagamaad,” Avaleht - TalTech raamatukogu digikogu, 10-Jun-2016. [Online]. Available: https://digikogu.taltech.ee/et/item/0b2c9ad3-db1f-47ac-bca0-4269832b5b59. [Accessed: 05-Nov-2021].

[11] “Card specification V2.3 - globalplatform.” [Online]. Available: https://globalplatform.org/wp-content/uploads/2018/05/GPC_CardSpecification_v2.3.1_PublicRelease_CC.pdf . [Accessed: 05-Nov-2021].

[12] “Elektrooniline identiteet eID,” [Online]. Available: https://www.ria.ee/et/riigi-infosusteem/elektrooniline-identiteet-eid.html. [Accessed: 05-Nov-2021].

[13] “ JavaCard Technology,” [Online]. Available: https://www.oracle.com/technetwork/java/embedded/JavaCard/overview/index.html. [Accessed: 05-Nov-2021].

[14] Icao.int (2015) Doc 9303 machine readable travel documents - *ICAO*. [online] Available at: https://www.icao.int/publications/Documents/9303_p1_cons_en.pdf [Accessed: 05-Nov-2021].

[15] EVS. (n.d.). EVS 827:2004. [online] Available at: https://www.evs.ee/tooted/evs-827-2004 [Accessed: 05-Nov-2021].

[16] Digi- ja väestötietovirasto. 2022. Citizen Certificate and electronic identity | Digital and population data services agency. [online] Available at: <https://dvv.fi/en/citizen-certificate-and-electronic-identity> [Accessed: 05-Nov-2021].

[17] Europa.eu. (2019). Regulation (EU) 2019/1157 of the European Parliament and of the Council of 20 June 2019 on strengthening the security of identity cards of Union citizens and of residence documents issued to Union citizens and their family members exercising their right of free movement (Text with EEA relevance.). [online] Available at: https://eur-lex.europa.eu/legal-content/et/TXT/?uri=CELEX:32019R1157 [Accessed: 31-Nov-2021].

[18] Icao.int (2015) Doc 9303 Machine Readable Travel Documents Part 9: Deployment of Biometric Identification Seventh Edition, 2015 and Electronic Storage of Data in MRTDs. (n.d.). [online] Available at: https://www.icao.int/publications/documents/9303_p9_cons_en.pdf [Accessed: 31-Nov-2021].

[19] Ponsini, N. (2020). Java Card 3.1 Unveiled. [online] oracle.com. Available at: https://blogs.oracle.com/javamagazine/post/java-card-31-unveiled [Accessed 4 Feb. 2022].

[20] Ponsini, N. (2019*)*. Java Card 3.1 explored. [online] oracle.com. Available at: https://blogs.oracle.com/javamagazine/post/java-card-31-explored [Accessed 4 Feb. 2022].

[21] Oracle.com. (2019). Java Card Protection Profile Downloads. [online] Available at: https://www.oracle.com/java/technologies/JavaCard-protection-profile.html [Accessed 4 Feb 2022].

[22] Mavroudis, V. and Svenda, P. (n.d.). Towards Low-level Cryptographic Primitives for JavaCards. [online] Available at: https://arxiv.org/pdf/1810.01662.pdf [Accessed 4 Feb 2022].

[23] Farhadi, M. and Lanet, J.-L. (2016). Chronicle of a Java Card death. Journal of Computer Virology and Hacking Techniques, [online] Available at:

https://www.oracle.com/java/technologies/javacard-protection-profile.html. [Accessed 4 Feb 2022].

[24] Ria.ee (2022) *Estonia resolves its ID-card crisis | Estonian Information System Authority*. [online] Available at: https://www.ria.ee/en/news/estonia-resolves-its-id-card-crisis.html [Accessed 4 Feb 2022].

[25] Oracle.com. (2019). *Deprecated APIs, Features, and Options.* [online] Available at: https://www.oracle.com/java/technologies/javase/9-deprecated-features.html. [Accessed 4 Feb 2022].

[26] Chen, Z. and Internet Archive (2000). *Java Card technology for Smart Cards : architecture and programmer's guide.* [online] Internet Archive. Boston : Addison-Wesley. Available at: https://archive.org/details/javacardtmtechno00zhiq [Accessed 4 Feb 2022].

[27] Oracle.com. (2018). *Developing a Java Card Applet*. [online] Available at: https://www.oracle.com/java/technologies/java-card/developing-JavaCard-applet.html [Accessed 4 Feb 2022].

[28] CardLogix Corporation. (n.d.). *Converted Applet*. [online] Available at: https://www.cardlogix.com/glossary/java-converted-applet-cap/ [Accessed 4 Feb 2022].

[29] Thales Group. (n.d.). *Java Card: definition, use cases and benefits.* [online] Available at: https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/javacard. [Accessed 11 Feb 2022].

[30] Omnicalculator.com. 2022. *Password Entropy Calculator*. [online] Available at: https://www.omnicalculator.com/other/password-entropy#:~:text=E%20%3D%20L%20 [Accessed 15 September 2022].

[31] PCMAG. (2022.). The Best Free Password Managers for 2021. [online] Available at: https://www.pcmag.com/picks/the-best-free-password-managers. [Accessed 3 October 2022].

[32] Aleksandersen, D. (n.d.). How to back up your password manager. [online] www.ctrl.blog. Available at: https://www.ctrl.blog/entry/password-manager-backup.html [Accessed 3 October 2022].

[33] Password Managers Reviews. (2022.). Which Password Managers Have Been Hacked? – Best Reviews. [online] Available at: https://password-managers.bestreviews.net/faq/which-password-managers-have-been-hacked/. [Accessed 4 October 2022].

[34] CyberNews. (2021). Are Password Managers Safe to Use in 2021? [online] Available at: https://cybernews.com/best-password-managers/are-password-managers-safe/. [Accessed 4 October 2022].

[35] Cox, J. (2022.). Websites, Please Stop Blocking Password Managers. It’s 2015. [online] Wired. Available at: https://www.wired.com/2015/07/websites-please-stop-blocking-password-managers-2015 [Accessed 4 October 2022].

[36] Wright, M. (2015). British Gas deliberately breaks password managers. [online] TNW | Insider. Available at: https://thenextweb.com/news/no-pass-on-this-one [Accessed 4 October 2022].

[37] Coggins, J. (2022). What is Passwordless Authentication? Benefits and Challenges. [online] Lepide Blog: A Guide to IT Security, Compliance and IT Operations. Available at: https://www.lepide.com/blog/what-is-passwordless-authentication-benefits-and-challenges [Accessed 4 October 2022].

[38] javacardos.com. (2022). Smart Card Developement Kit| JavaCardOS Tools. [online] Available at: https://www.javacardos.com/tools [Accessed 6 October 2022].

[39] docs.oracle.com. (2022). JavaCard 3 Platform Development Kit User Guide, Classic Edition - Contents. [online] Available at: https://docs.oracle.com/JavaCard/3.0.5/guide/index.htm [Accessed 6 October 2022].

[40] Martin Paljak. (2022). Home · martinpaljak/GlobalPlatformPro Wiki. [online] Available at: https://github.com/martinpaljak/GlobalPlatformPro/wiki [Accessed 6 October 2022].

[41] Mobilityforesights.com. (2022.). Europe Smart Card Market 2022-2027 | December 2022 Updated. [online] Available at: https://mobilityforesights.com/product/europe-smart-card-market [Accessed 7 October 2022].

[42] Digital-strategy.ec.europa.eu. (2022.). eIDAS Regulation | Shaping Europe's digital future. [online] Available at: https://digital-strategy.ec.europa.eu/en/policies/eidas-regulation. [Accessed 7 October 2022].

[43] Martin, Hugues & du Bousquet, Lydie. (2004). Automatic Test Generation for Java Card Applets. Available at: https://www.researchgate.net/publication/2888730_Automatic_Test_Generation_for_Java_Card_Applets [Accessed November 20, 2022].

[44] cadp.inria.fr. (n.d.). TGV manual page. [online] Available at: http://cadp.inria.fr/man/tgv.html#sect2 [Accessed November 20, 2022].

[45] Pavel Kargin (2021) Identity Testing Tool [online] Available at: https://github.com/pavkar/Identity-Testing-Tool [Accessed 28 Nov. 2022].

[46] RIA. (2022). *Install ID-software.* [online] Available at: https://www.id.ee/en/article/install-id-software/ [Accessed 28 Nov. 2022].

[47] Reichl, D. (n.d.). *Features - KeePass*. [online] keepass.info. Available at: https://keepass.info/features.html.

[48] zapier.com. (n.d.). *LastPass vs. 1Password: Which should you use? [2023] | Zapier*. [online] Available at: https://zapier.com/blog/lastpass-vs-1password/ [Accessed 29 Nov. 2022].

[49] Estonian Information System Authority. (2022). *ARMIS development guidelines and future plans.* Internal RIA report. Unpublished.

[50] Pavel Kargin. (2022). [online] *Password Manager* Available at: https://github.com/pavkar/Password-Manager-Applet-Maven [Accessed 28 Nov. 2022].

[51] Markantonakis, K. and Mayes, K. (2003). An overview of the GlobalPlatform smart card specification. Information Security Technical Report, 8(1), pp.17–29. doi:10.1016/s1363-4127(03)00103-1 [Accessed 20 Dec. 2022].

[52] Police and Border Guard Board. (2021). A settlement agreement has been signed between the Police and Border Guard Board and Gemalto AG Tallinn - News. [online] Available at: https://www.politsei.ee/en/news/a-settlement-agreement-has-been-signed-between-the-police-and-border-guard-board-and-gemalto-ag-tallinn-2021 [Accessed 20 Dec. 2022].

[53] Pavel Kargin (2022*). GitHub - pavkar/PasswordManagerServer at master.* [online] Available at: https://github.com/pavkar/PasswordManagerServer/tree/master [Accessed 21 Dec. 2022].

[54] Paljak, M. (2022). *FIDO2*. [online] GitHub. Available at: https://github.com/martinpaljak/FIDO2 [Accessed 27 Dec. 2022].