

# Security Analysis of the Austrian Citizen Card Environment MOCCA and E-Card

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Medizinische Informatik**

eingereicht von

**Thomas Johannes Stipsits**

Matrikelnummer 1025098

an der Fakultät für Informatik  
der Technischen Universität Wien

Betreuung: Dipl.-Ing. Markus Kammerstetter  
Mitwirkung: Prof. Dr. Wolfgang Kastner

Wien, 6. August 2015

---

Thomas Johannes Stipsits

---

Markus Kammerstetter

# Security Analysis of the Austrian Citizen Card Environment MOCCA and E-Card

## BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

### **Bachelor of Science**

in

### **Medical Informatics**

by

**Thomas Johannes Stipsits**

Registration Number 1025098

to the Faculty of Informatics  
at the Vienna University of Technology

Advisor: Dipl.-Ing. Markus Kammerstetter

Assistance: Prof. Dr. Wolfgang Kastner

Vienna, 6<sup>th</sup> August, 2015

---

Thomas Johannes Stipsits

---

Markus Kammerstetter

# Erklärung zur Verfassung der Arbeit

Thomas Johannes Stipsits  
Wiesengasse 4, 7344 Stoob

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 6. August 2015

---

Thomas Johannes Stipsits

# Kurzfassung

Diese Arbeit setzt sich intensiv mit der Verbindung der Bürgerkarte und der Bürgerkartensoftware MOCCA auseinander. Besonderes Augenmerk liegt auf den Sicherheitsmaßnahmen zur Wahrung der Privatsphäre und Schutz der Identität des Nutzers bei Authentifizierungsprozessen mit MOCCA. Hierzu wird eine Klassifizierung nach Angriffen eingeführt, deren mögliche Angriffsarten vorgestellt werden. Es werden beide Varianten von MOCCA vorgestellt, beschrieben und gegenübergestellt. Dabei zeigt sich, dass vor allem MOCCA local derzeit kaum Anwendungsgebiete für den User bereitstellt. Im Gegensatz dazu bietet die online Variante eine Fülle an Anwendungsgebieten und kommt als Authentifizierung vor allem bei E-Government Applikationen oder Signaturerstellung zum Einsatz. Die technische Architektur dieser beiden Varianten wird vorgestellt und der Ablauf einer Signaturerstellung mittels MOCCA online analysiert. Zusätzlich dazu wird kurz das Konzept der österreichischen E-Card erläutert und die auf ihr gespeicherten Daten extrahiert und untersucht. Hierbei ist besonders bemerkenswert, dass die österreichische E-Card nur personenbezogene Daten, wie Name, Geburtsdatum, Sozialversicherungsnummer und ZMR-Nummer speichert. Weiters werden potentielle Sicherheitslücken der Bürgerkartensoftware genauer untersucht und deren Anfälligkeit für Attacks bewertet, wobei sich zeigt, dass MOCCA vor allem im Hinblick auf Authentizität des Applets einige Defizite aufweist. Basierend darauf werden zukünftige Angriffsszenarien erstellt, die einen Ausblick auf mögliche Angriffswege schaffen. Zum Abschluss wird ein möglicher Angriffsweg sowie eine Zusammenfassung und Verbesserungsvorschläge, wie zum Beispiel Sicherung der Authentizität mittels Codesignatur des Applets, präsentiert.

# Abstract

This thesis focuses on the connection between the Austrian citizen card and its environment MOCCA. In particular, the security measures for ensuring the integrity of the user's privacy of identity in terms of authentication are considered and evaluated. A classification of attackers and their possible attack vectors is given. Furthermore, both varieties of MOCCA will be introduced and discussed. Based on this discussion, it is obvious that the applications of MOCCA are currently very limited. In spite of this, the online variant offers a huge amount of possible applications as a solution for authentication, especially based on e-government or signature creation services. In addition, the technical architecture of both, the local and online version, will be introduced, and the procedure of a signature creation will be examined. Next, the concept of the Austrian e-card will be introduced, and the data stored on it will be extracted and analysed. The obtained information indicates that the e-card solely stores personal linked data like name, date of birth, insurance number, and the ZMR-number. Furthermore, potential leaks of MOCCA will be discussed, and their vulnerability will be evaluated. This evaluation shows that MOCCA has several deficiencies in terms of authenticity of the applet. Based on this, some scenarios, that command a view on potential attack vectors, will be introduced. Finally, one of these potential attack vectors will be discussed, a summary of the result and an outlook containing suggested improvements like ensuring authenticity through applet code signature will be given.

# Abbreviations

**MOCCA** - Austrian Citizen Card Environment  
**APDU** - Application Protocol Data Unit  
**GUI** - Graphical User Interface  
**EGIZ** - E-Government Innovationcentre  
**REST** - Representational State Transfer  
**JNLP** - Java Network Launch Protocol  
**JAR** - Java Archive  
**IT** - Information Technology  
**API** - Application Programming Interface  
**SLF4J** - Simple Logging Facade for Java  
**LOG4J** - Java-based Logging Utility  
**IAIK** - Institute for Applied Information Processing and Communications  
**JCE** - Java Cryptography Extension  
**XML** - Extensible Markup Language  
**JAXB** - Java Architecture for XML Binding  
**STARCOS** - Smart Card Chip Operating System  
**TCP** - Transmission Control Protocol  
**STAL** - Security Token Abstraction  
**SMCC** - Smart card Communication  
**ELGA** - Electronic Patient Record, called 'Elektronische Gesundheitsakte'  
**E-Card** - Smart card used by the Social Insurances for handling Electronic Patient Records  
**PIN** - Personal identification number  
**ZMR** - Zentrales Melderegister,  
**ECDSA** - Elliptic Curve Digital Signature Algorithm  
**SHA** - Secure Hash Algorithm  
**RSA** - Cryptosystem, named after its inventors: Rivest, Shamir and Adleman  
**Triple-DES** - Triple-Data Encryption Standard  
**BMF** - Bundesministerium für Finanzen (federal ministry of finance)  
**SSL** - Secure Sockets Layer  
**SSH** - Secure Shell

# Contents

<b>Kurzfassung</b>	<b>iv</b>
<b>Abstract</b>	<b>v</b>
<b>Abbreviations</b>	<b>vi</b>
<b>Contents</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	2
1.2 Basic Concept of the Citizen Card . . . . .	3
1.3 Software using MOCCA/E-Card . . . . .	5
1.4 Problem Statement . . . . .	5
<b>2 Methodology</b>	<b>6</b>
2.1 Used Resources . . . . .	6
2.2 Analysis Methods . . . . .	6
2.3 Used Standards . . . . .	7
2.4 Used Software . . . . .	7
<b>3 Security Analysis</b>	<b>8</b>
3.1 Classification of Attackers . . . . .	10
3.2 MOCCA . . . . .	10
3.2.1 Overview . . . . .	10
3.2.2 Code Architecture . . . . .	11
3.2.3 MOCCA Local . . . . .	13
3.2.4 MOCCA Online . . . . .	14
3.2.5 Disparities of MOCCA Online and Local . . . . .	17
3.3 E-Card . . . . .	18
3.3.1 Overview . . . . .	18
3.3.2 Data Documentation . . . . .	19
Social Insurance Number . . . . .	20
Personal Link . . . . .	22
Certificates . . . . .	23

Signature Hash . . . . .	26
3.4 Possible Vulnerabilities . . . . .	27
3.5 Discussion of Vulnerabilities . . . . .	28
3.6 Potential Scenarios . . . . .	31
3.7 Potential Impact . . . . .	32
<b>4 Attack Example</b>	<b>33</b>
4.1 PIN Spoofing . . . . .	33
4.1.1 Phase 1: Setting up . . . . .	34
4.1.2 Phase 2: Distributing . . . . .	34
4.1.3 Phase 3: User profiling . . . . .	34
4.1.4 Phase 4: Obtaining the card . . . . .	35
4.1.5 Phase 5: Breaking the authentication . . . . .	35
4.2 Additional Findings . . . . .	35
<b>5 Results and Conclusion</b>	<b>37</b>
5.1 Results . . . . .	37
5.2 Conclusion . . . . .	38
5.3 Outlook . . . . .	38
<b>Bibliography</b>	<b>39</b>
<b>6 Appendix</b>	<b>43</b>
6.1 APDU Table . . . . .	44
6.2 Code . . . . .	45



# Introduction

One of the most revolutionary steps in the evolution of online authentication in Austria is the invention of the citizen card, which gives the user the opportunity to use the established electronic health smart card, called e-card, for authenticating, signing documents, and other features provided by different web services, additional software solutions and citizen card environments. This bachelor thesis deals with such an environment called "Modular Open Citizen Card Architecture", in short MOCCA. It was developed by the E-Government Innovationcentre EGIZ in cooperation with the IAIK Institute of the Technical University Graz. [1]

Since these systems are fairly new, there are quite a few problems to be solved. From usability to code quality, major parts of this environment will be reworked in the near future due to new findings in science, known bugs or bad smells in the code. Such modifications need prior analysis in order to achieve the aim of their developers. Since this thesis deals primarily with security domains, it will examine the source code, privacy issues and security features of MOCCA and the e-card.

The aforementioned features are the basis for all citizen card applications and therefore have access to identity-linked information. If the security measures of those features fail, it is likely that an attacker would be capable of hijacking the identity of his victim. In addition to the discussion, this thesis will give an insight on the basic technical architecture of MOCCA, the citizen card data stored on the e-card and the communication between the MOCCA environment and the e-card. Furthermore, there will be a critical reflection on the source code of MOCCA and an estimate of potential leaks and weaknesses of the system. In addition, a conclusion of all those topics will provide an outlook on security and quality of the MOCCA software in relation to the e-card data.

This thesis will not represent a guide for attacking the citizen card environment, but an analysis of the security of the current stable solution. The analysis methods do not consist of quantitative investigations, but of qualitative code and functionality inspections. Instead this thesis will deal with theoretical weaknesses and future threats, such as outdated encryptions, than with concrete attacks on the system. Since it is a security analysis there will be the one or another internal data revealed, but none that is critical for taking attacks.

## 1.1 Motivation

Privacy and information security are one of the biggest concerns in modern software solutions. Every software system today has to deal with various threats and therefore needs an ongoing investigation and maintenance. The most critical leaks and errors are already produced in the planning and implementation phase and lurk in the system, waiting for someone to find and exploit them. Even though software quality assurance and management bring various powerful tools and concepts for finding and correcting these leaks and errors, many of them find their way into the stable version and therefore into deployment of the system. The most common reasons for such leaks are too short time plans or a tight budget.

When it comes to health data or the identity of the user, it is especially crucial to ensure an ongoing quality of privacy in software systems. The previous century changed the IT-solutions for healthcare dramatically. Where doctors and hospitals once had to deal with conventional paper records, most of this data today is created, edited, stored and shared in separate systems. Solutions like hospital information systems, a health smart card for every citizen, like the Austrian e-card, or electronic patient records like "ELGA"(Austrian Electronic Patient Record) have to be designed not only for increasing usability, ensuring data integrity and avoiding downtimes, but also for ensuring privacy and data security.

There are several circumstances which make intense penetration testing on such medical data storage necessary. Besides that, the citizen card environment offers additional functionalities for the user, that have to be considered while dealing with privacy aspects. Since the citizen card environment links the identity of the user with those functionalities the standard for security measures has to be fairly high, actually comparable to cash cards or credit cards.

## 1.2 Basic Concept of the Citizen Card

Since the citizen card provides various possible applications and deals primarily with the identity of its user, there is an urgent need of ensuring privacy and integrity. Thus, the basic concept of preventing unauthorized access is a two factor authentication, realized by PINs and the smart card itself. Such authentication models are not new and therefore, most signature cards are used in a similar way.

The basic idea behind using a smart card for authorization is storing a private key on the card, preventing the extraction of it and therefore ensuring the secrecy of this key. It is urgent that nobody knows this key, not even the user himself. All one needs for accessing the key is a PIN that can be chosen at the citizen card registration. The Austrian citizen card solution uses two independent PINs for authenticating, a card PIN which grants access to the basic data on the card, like the personal link and encrypting data functionality, and a signature PIN for accessing the signature algorithms.

While signing documents, the user authorizes with the signature PIN at the card terminal. After receiving the correct PIN, the card gives access to the security environment, where the signature algorithms are stored. This signature is basically a signed hash of the data, that can be checked by using the associated public key.

The personal link data on the card is used to create a sustainable link between the citizen card and its owner and is the basis for the qualified signature. This personal link consist of the Austrian ZMR-Number, which is unique for every Austrian citizen and therefore suitable for identifying the user. This number is encrypted with a Triple-DES encryption. [2] Figure 3.4 gives a brief overview on the two factor authentication process used by MOCCA's online solution.

Citizen card environments fill the gap between the user's input and the algorithms stored on the card, an e-government system and his users and especially between an anonymous user and his real life identity. Such software needs to handle the PIN verification, processing of APDU commands and ensuring a secure transmission of data between further applications and the card. Besides MOCCA, there are a few environments that were developed by providers like A-SIT. The following list contains some examples of other citizen card environments that are capable of handling the Austrian e-card solution [3]:

- **A-Trust Citizen Card Environment**
- **trustDesk**
- **hotSign**

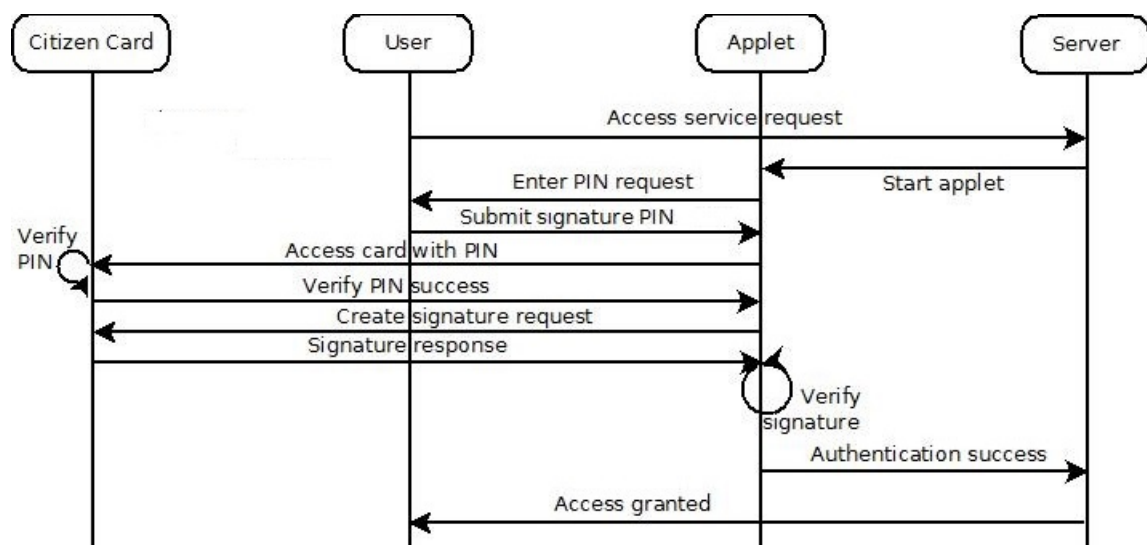


Figure 1.1: Two-factor authentication model for MOCCA

Currently, there are few web services that use MOCCA like FinanzOnline, PrimeSign or other e-government applications. Since the citizen card is a fairly new concept and still has a meagre distribution in Austria, the economy has not yet started to consider the usage of the citizen card applications in a commercial way. Table 1.1 presents an representative survey [4] done by Statistics Austria in 2012 focusing on how many Austrian citizens own a citizen card and their usage of it.

	People that...		
	own a citizen card or a handy signature	do not own a citizen card or a handy signature	
<b>People in 1000</b>	325,3	1873,0	

Table 1.1: Citizen card usage in 2012

### 1.3 Software using MOCCA/E-Card

There are currently few tools available for users to utilise their citizen cards. Most of the current software is hosted by A-SIT [5]. These tools have no access to the data stored on the card, therefore they need a separate environment like MOCCA in order to perform their functionalities. Some of the tools provided by A-SIT are listed below. [6]

- **trustDesk** This tool certifies the correctness and authenticity of the qualified signature used to sign a document. In addition, it can be used as a stand alone citizen card environment.
- **Citizen Card Encrypted** The freeware CCE tool can encrypt given data for communication or storage. It also offers the functionality to grant a predefined group of individuals (their signatures) the access to decrypt the data.
- **secureEFS** The secureEFS software offers the user a tool to encrypt whole directories like parts of the cache, configuration or password storage directories.

### 1.4 Problem Statement

Fortunately, the trend in developing and maintaining bigger systems used in health care and medicine, like hospital information systems, shows that most of the leading companies focus on improving not only privacy but also concepts on sharing health information only with qualified stakeholders like hospitals or health professionals. Nevertheless, it is crucial to perform intense investigations in order to find and remove errors and leaks.

Even the best tests and efforts can lead to failure, if they are taken too lightly. Such investigations should not only be performed by the developers of those systems, they simply know too much of their own code, get stuck in the same patterns or tend to overlook bad smells or threats. Additional independent tests are a powerful tool for tracking such leaks. Therefore, it is crucial to do such additional test and reviews on this kind of software.

The additional functionalities provided by MOCCA and different applications for the citizen card need to be tested independently since they offer a vast variety of different functionalities like signing and encrypting documents and directories, authenticate at different services and even signing contracts online. Therefore, strong security measures are crucial not only for the operation of the system, but also for creating a wide acceptance and confidence among the users.

# Methodology

## 2.1 Used Resources

The main resource for this thesis was the source code of MOCCA, the MOCCA applet and the stored data on the citizen card itself. The local version of MOCCA, as well as the online applet [7], was decompiled and analysed with the standards of reverse engineering. In addition to the decompiled code, the official homepage of the citizen card and the EGIZ was used. [8][9] Furthermore, the data on the e-card was analysed by executing the original APDU commands, which were extracted from the MOCCA source code. Basic resources for the technical architecture of both, MOCCA and the e-card, was the official documentation, the code and the EGIZ website.

## 2.2 Analysis Methods

MOCCA local was analysed by reviewing the source code after decompiling it with Java decompiler GUI. For additional data, the aforementioned resources were considered. MOCCA online was reviewed by analysing the applet source code and considering the disparities mentioned on the official website and the official documentation. The data and the structure of the e-card were analysed by extracting and executing the official APDU commands from both, the MOCCA local and the MOCCA applet code. The execution of the APDU commands was done with JSmartCardExplorer and self coded python pycard scripts. After extracting the certificates, OpenSSL and the java.security.cert package were used to investigate the structure. The scripts used for extracting the certificates and the personal link data are listed in the Appendix, Listings 6.6, 6.7 and 6.8.

## 2.3 Used Standards

Since the following standards are part of the MOCCA citizen card solution, they were also considered during the preparation of this thesis. In particular, ISO7816, a standard for smart card communication and commands, was critical for analysing the smart cards data:

**PC/SC [10]:** Personal Computer / Smart Card, short PC/SC, is a specification for communication between smart cards and computers. PC/SC was implemented by Microsoft.

**ISO7816-4 [11]:** ISO standard for communication with smart card/ APDU Commands

**ISO7816-8 [12]:** ISO standard for security communication with smart card/ APDU security operational commands

## 2.4 Used Software

The following list contains prefabricated freeware that was used to investigate different resources for this thesis.

**JSmartCardExplorer [13]:** The JSmartCardExplorer is a freeware that offers the user the opportunity to perform APDU commands on a connected smart card. A GUI is included, which allows the user to create, perform and store APDU commands, set up projects to store lists of those commands and study the output that was sent by the card.

**Java Decompiler GUI [14]:** Java Decompiler GUI is a freeware that combines a decompiling function with a simple GUI. The advantage of this GUI is that users can get a brief overlook on complex and large software packages.

**Python:** A high level programming language that is often used as scripting language because of its dynamic typing support. Furthermore, the high legibility of python should be considered.

**Pyscard:** Pyscard is a python library that primarily deals with smart card communication and adds PC/SC support to python scripts. The library consists of two components, SCARD for wrapping the PC/SC specifications in a python usable context and SMARTCARD, a special framework that operates on a high level.

**OpenSSL [15]:** OpenSSL is a freeware that primarily deals with Transport Layer Security. It also comes with a shell program for creating, maintaining and analysing certificates.

**YASCA [16]:** YASCA (Yet Another Source Code Analyzer) is a tool that is capable of performing an automated source code analysis on potential weaknesses of the software.

## Security Analysis

For further discussion, as well as the creation of an adequate security analysis, it is fundamental to introduce a classification of different attack types, their needed resources, and the impact on the security of the citizen card environment. Table 3.1 represents a subdivision of attack types according to the initiator of the attack. The more resources an attacking organisation has, the more advanced a possible attack will be. The classification introduced by the table will be used for further description of the security measures of MOCCA.

As mentioned in Chapter 1, the citizen card environment MOCCA holds a great potential for misuse since most e-government applications offer their users the ability to authenticate through its online applet solution. Thus, MOCCA's code needs to be reviewed, evaluated and continuously corrected and reworked. Furthermore, different types of attackers may focus on different datasets of the user. Thus, it is urgent to keep a critical view on the citizen card solution itself and its very own architecture that is used for processing its functionalities. Section 3.2 deals with this information in detail.

In addition, a detailed security analysis needs to consider the involved data flow from the solution's components. Section 3.3 takes a closer look at the data stored on the e-card, the storage architecture these smart cards use, and the algorithms used for signing or encrypting data. Furthermore, this section, in addition to Section 3.2, grants the reader basic insight on the data flow of the citizen card solution.

Since analysing code and dealing with sensible data are not sufficient for performing an adequate security analysis, this thesis also examines a range of possible attacks and introduces weaknesses. Sections 3.4, 3.5, 3.6 and 3.7 deal with vulnerabilities, resulting attack scenarios, and their potential impact on the system and its users.



Type	Description
0	<p><b>Initiator:</b> individual professional, involving single actors</p> <p><b>Level of available resources:</b> low, low-mid knowledge of the systems, low-mid time resources</p> <p><b>Potential attacks:</b> low-mid ranged attacks, cross site scripting, spoofing, DDOSing</p> <p><b>Impact on the system:</b> low-mid, mostly attacks that do not affect the system, but rather the interaction between the user and the system</p> <p><b>Impact on a single user:</b> low - high, depends on the type of attack performed</p>
1	<p><b>Initiator:</b> insider, attack performed or supported by an individual with insider knowledge</p> <p><b>Level of available resources:</b> mid, high knowledge of the systems, low-mid time resources</p> <p><b>Potential attacks:</b> mid ranged attacks, spoofing, DDOSing, exploiting attempts</p> <p><b>Impact on the system:</b> mid, especially exploiting or DDOSing the services can affect the whole system</p> <p><b>Impact on a single user:</b> mid , high knowledge of the system but short on time resources</p>
2	<p><b>Initiator:</b> professional groups, involving groups like Anonymous</p> <p><b>Level of available resources:</b> mid, low-mid knowledge of the systems, highly specific attacks</p> <p><b>Potential attacks:</b> mid ranged attacks, like spoofing, DDOSing, exploiting attempts</p> <p><b>Impact on the system:</b> mid-high, especially exploiting or DDOSing the Web server can affect the whole system</p> <p><b>Impact on a single user:</b> mid - high, depends on the type of attack performed, typically more advanced attacks than type 0 and 1</p>
3	<p><b>Initiator:</b> organized groups, involving institutions or even federal attacks</p> <p><b>Level of available resources:</b> high, mid knowledge of the systems, high time and knowledge resources</p> <p><b>Potential attacks:</b> high ranged attacks, exploiting, destruction</p> <p><b>Impact on the system:</b> low-high, when the strategy demands it severe damage or stealthy infiltration of the system</p> <p><b>Impact on a single user:</b> none - high, depends on the type of attack performed, most advanced attacks</p>

Table 3.1: Attack classes

## 3.1 Classification of Attackers

This thesis solely deals with types 0, 1 and 2, since type 3 is very unlikely to happen in the current political situation. Type 0 attacks are often performed by individual criminals and are usually profit orientated. Most of those attacks will aim to hijack the identity of the victim. Those attacks often do not need to be very sophisticated. Furthermore, there will be a discussion about such an attack in Chapter 4.

Type 1 attacks sound rather unlikely, but valuable knowledge about identity linked authentication can be worth much money on the black market. Although it is rather unlikely, this type of attack powered by insider knowledge can be very destructive and critical not only for the victim, but also for the confidence the users have in the system.

Type 2 attacks are the most dangerous of the considered attack types. These attacks are performed by well organised professional groups that managed to infiltrate a wide range of sophisticated systems in the past. These groups spend a great amount of time and knowledge resources and are capable of performing advanced attacks that can cause severe damage to either the system, or the victim.

## 3.2 MOCCA

MOCCA is a citizen card environment driven by the EGIZ in cooperation with the IAIK of the Technical University Graz, which is capable of reading the data stored on the e-card, authenticating the user with the two factor authentication used by the citizen card system, and transmitting it to an online signature creator [1]. In addition to this, MOCCA local offers the user the ability to extract both certificates, processing PIN management and reading further information like the e-card version, the personal binding, and other data from the card [17].

### 3.2.1 Overview

The Austrian citizen card gives its user the opportunity to create a qualified signature which can be used to sign contracts or other electronic documents and therefore authenticates the user. According to Austrian law, the qualified signature replaces the conventional one. Therefore, the software solutions that are used to create those signatures need a high level of security measures. To fulfil this standard of security, every user needs at least a signature pin, a card PIN, certificates and a password for disclaiming the registration. The signature PIN is solely used for signing documents and authenticating at e-government applications, while the card PIN is used for accessing the personal link files, encrypting data and other functionalities offered by different applications. In the latest citizen card version (G4, see Section 3.3), the user has 10 retries for the card PIN and 3 retries for the signature PIN. After exceeding these retries, a security mechanism

blocks the PIN and the card becomes useless for citizen card functions. For the common card operating system in Austria (STARCOS), it is not possible to unblock the PIN. Thus, the user has to order a new card to regain access to the citizen card functionalities.[18]

The MOCCA client, as a local solution, manages to close the gap between the user and the data stored on the e-card, like the personal binding. Additionally, it offers some functionalities regarding PIN management. In the online solution, the MOCCA applet is used as a client part that handles authentication and smart card commands. In addition, the applet transmits the certificate and the signature hash to the online creator. An example of a qualified signature created by the online creator PrimeSign [19] is given in Figure 3.1. The core of the qualified signature, the signature hash, is not shown in the representation of the signature, but stored in the sourcecode of the data.


	<b>Unterzeichner</b>	Thomas Johannes Stipsits <b>1.</b>
	<b>Datum/Zeit-UTC</b>	2015-02-19T19:21:02+01:00 <b>2.</b>
	<b>Prüfinformation</b>	Informationen zur Prüfung der elektronischen Signatur finden Sie unter: <a href="https://www.signaturpruefung.gv.at">https://www.signaturpruefung.gv.at</a> <b>3.</b>
<b>Hinweis</b>	Dieses mit einer qualifizierten elektronischen Signatur versehene Dokument ist gemäß § 4 Abs. 1 Signaturgesetz einem handschriftlich unterschriebenen Dokument rechtlich gleichgestellt. <b>4.</b>	

Figure 3.1: Qualified signature created by PrimeSign

### Signature Description

1. **Name** - Name of the owner
2. **Date** - Date and time of the creation
3. **Verification** - Link to the verification software
4. **Hint** - Hint for the statutory rule that this signature substitutes the conventional signature

### 3.2.2 Code Architecture

The citizen card environment MOCCA can be used in 2 different ways, as a local and an online alternative. The composition of the local version, which comes as a rich client, can be distributed in 4 particular layers [20]:

- **Launching/Transportation:** Handles the launch or autostart of MOCCA and uses REST [21] to call the necessary functions.
- **Request Mapping:** Maps the requests according to their type and creates the corresponding XML-requests, like the signature or personal link requests, used by the online solution.
- **Security Token Abstraction/Service:** STAL serves as a secure transmission of the mapped requests / the responses.
- **Smartcard Communication/Persistence:** The Smartcard Communication layer, in short SMCC, uses the PC/SC specifications and the Smartcard I/O API to perform requests on the card.

The MOCCA online solution consists basically of a MOCCA Web server (e.g. an online signature creator) and a MOCCA Java applet, that administrates user entries like the signature PIN and replaces the MOCCA client used in the local solution. This applet serves as an extension for the used browser and communicates with the chosen online signature creator or other applications that need a connection to the card.

The applet itself only covers the Smartcard Communication and Security Token Abstraction layers, the Web server the Transportation and Request Mapping layers. The communication between the applet and the MOCCA Web server is performed by the Security Token Abstraction Layer, in short STAL. [22] An example of an applet used to authenticate while creating the qualified signature is pictured in Figure 3.2.

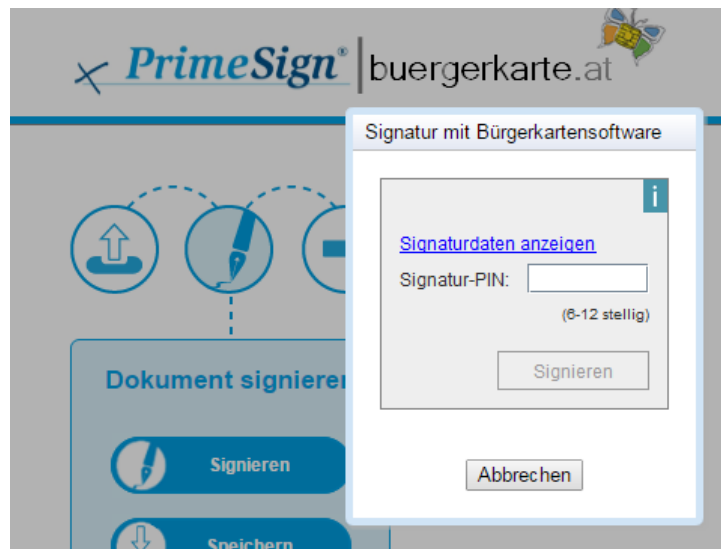


Figure 3.2: The MOCCA online applet used by PrimeSign

### 3.2.3 MOCCA Local

The local version of MOCCA comes as a single JNLP file, that uses an XML-configuration file to check for available updates before starting the software from various JAR files listed below. The up-to-date version of MOCCA is available at the citizen card homepage [23]. In addition to the JNLP file, MOCCA uses Maven to build itself. To download the single JAR archives, one has to append the filenames to the codebase [24] link.

**BKUWebStart-1.3.16.jar:** Contains the MOCCA code

**BKUCertificates-1.11.jar:** Contains a wide range of certificates

**slf4j-api-1.6.4-s.jar:** The simple logging facade for Java provides abstraction for several logging frameworks. This JAR contains the API for SLF4J

**slf4j-log4j12-1.6.4.jar:** This JAR contains the suitable SLF4J Version for LOG4J]

**log4j-1.2.17.jar:** LOG4J is a logging framework for Java

**jetty-6.1.19.jar:** Jetty is a servlet engine and HTTP server

**iaik-jce-full-signed-5.01.jar:** Contains several cryptographic extensions, signed by IAIK

**servlet-api-2.5-20081211.jar:** Basic Java servlet API

The main purpose of the Launching/Transportation layer is to manage the start and the shut down of the software and to display the GUI. The WebStart package contains all the code that belongs to this layer: The launcher, request invoker classes, the GUI and some necessary configuration files. For mapping the invoked requests, the launcher uses REST messages. An example of this is given in Listing 6.5. The WebStart package is designed for running on the three most common operating systems: Windows, Linux, and MacOSX.

The request mapping part of the citizen card environment listens for incoming Secure Layer Requests on TCP ports 3495 and 3496. [25] The local version uses REST for mapping the requests and calls via localhost the corresponding TCP-ports. Therefore, MOCCA can be used simultaneously as a local and as an online tool. After mapping the requests, the citizen card environment calls the STAL functions.

The Security Token Abstraction layer serves as a secure transmission layer between the client (or the Web server) and the card. If the local citizen card environment communicates with a Web application, the whole data transmission is done with those STAL encodings. The local environment, as well as the server, uses only the STAL layer to call functionalities of the SMCC layer.

After receiving the STAL requests, the SMCC layer checks whether the smart card

is already inserted, which configuration the environment has to use for communicating with the card operating system, performs APDU commands to read data from the card, and returns the responses to the STAL layer. The SMCC layer supports a wide range of different smart cards, the most common in Austria uses the STARCOS operating system. Table 3.2 [26] gives an overview of the currently supported cards and their country of origin.

<b>MOCCA notation (Operating system)</b>	<b>Country</b>
BELPIC (Gemalto Cyberflex [27])	Belgium
EstEID (JCOP [28])	Estonia
FIN eID	Finland
IS VISA electron	Iceland
IS Maestro	Iceland
ITCards	Italy
LT eID	Lithuania
E-card (STARCOS [29])	Austria
a-sign premium EPA (ACOS [29])	Austria
a-sign premium MCA (ACOS [29])	Austria
Portugal	Portugal
SE eID	Sweden
SwissSign	Switzerland
QuoVadis	Switzerland
ES DNLe	Spain
Cypriot EID	Cyprus
Gemalto .NET V2.0	Cyprus

Table 3.2: Currently supported smart cards [26]

### 3.2.4 MOCCA Online

As mentioned in Section 3.2.1, the online version of MOCCA consists of two separate parts, a Web server and a Java applet, that is used to authenticate and transmit the data read from the card. This applet establishes a secure connection between the Web server and the citizen card provided by the user. Basically, it consist of a single JAR-File that is downloaded directly after starting the signature process. In addition, the applet can be downloaded manually. The download link can be obtained by enabling and reviewing the log of the Java console while processing a certain request on a MOCCA supported site. A schematic overview on the basic idea of this applet is shown in Figure 3.3.

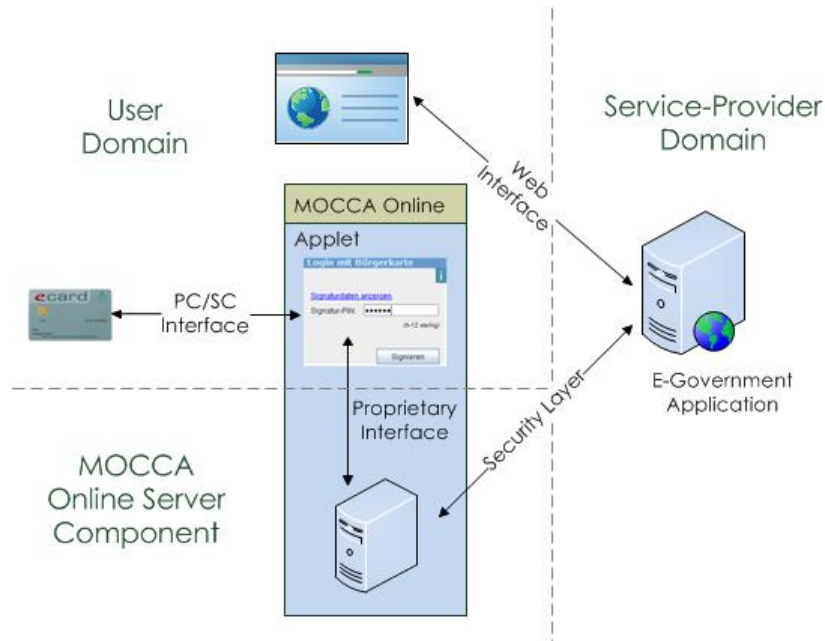


Figure 3.3: Basic idea behind MOCCA online[22]

This basic architecture allows MOCCA to split the signature creation into two separated parts, an online and a local part. The local part is processed by the applet after receiving a signature request from the server. In detail, the applet receives an XML-request containing a hashed value of the document that needs to be signed. Next, the applet validates the given PIN by transmitting it to the card and calling the validate function. After that, the hashed data is transmitted to the card, which uses a security environment for creating the signature hash. Past to receiving the response from the card, the applet transmits the signature hash to the Web server, which performs further processing. The online part of the signature creation calls the applet, transmits the data hash to the applet and creates the final signature out of the user's certificate and the signature hash received from the applet. A summary of these actions is given in Figure 3.4.

As mentioned in the MOCCA Architecture (Section 3.2.2), both the online and local version consist of 4 basic layers. Since the online solution is separated into 2 independent parts, the applet does not need to implement all 4 layers. This ensures a secure communication with the card and transmit the responses in a secure channel to the Web server for further processing, the applet only needs to implement the Smart Card Communication layer and the Security Token Abstraction layer. Both, the Transportation and the Request Mapping layers are implemented by the server. The communication between the applet and the Web server is processed by the STAL layer. Figure 3.5 provides a brief overview on the separation of the MOCCA layers in the online solution.

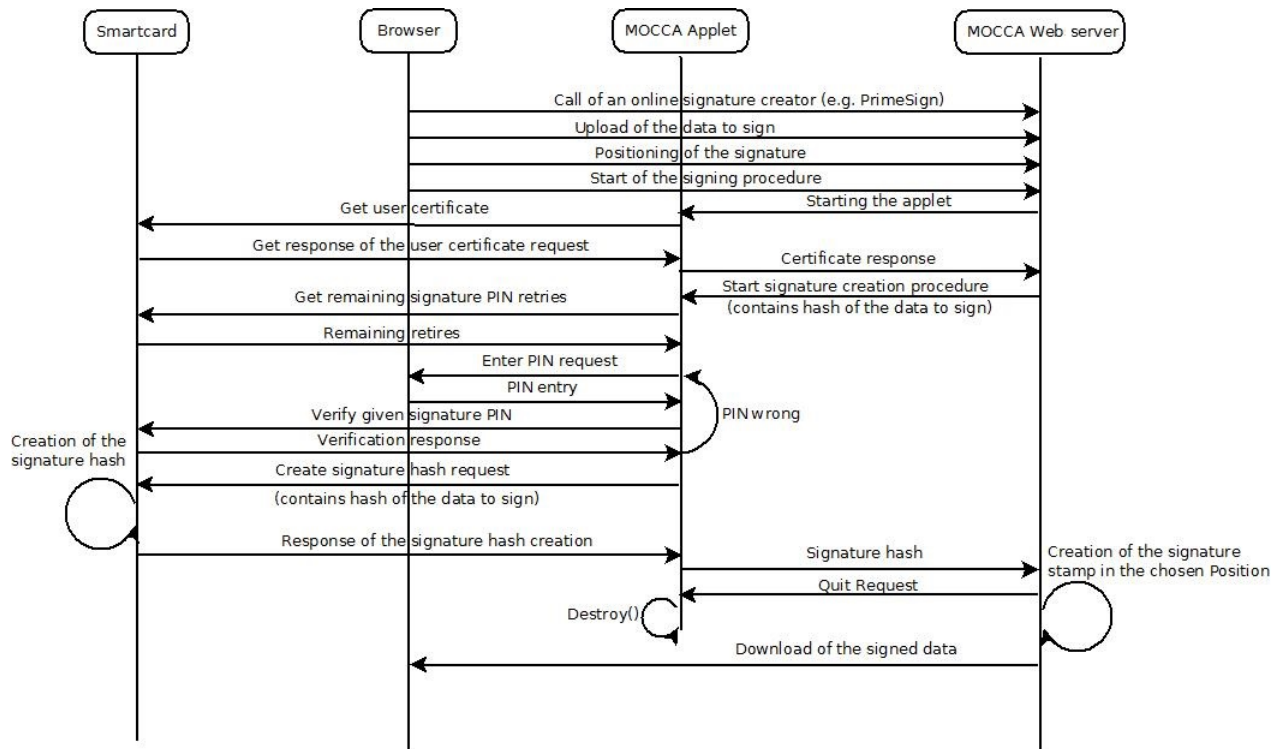


Figure 3.4: MOCCA online applet actions

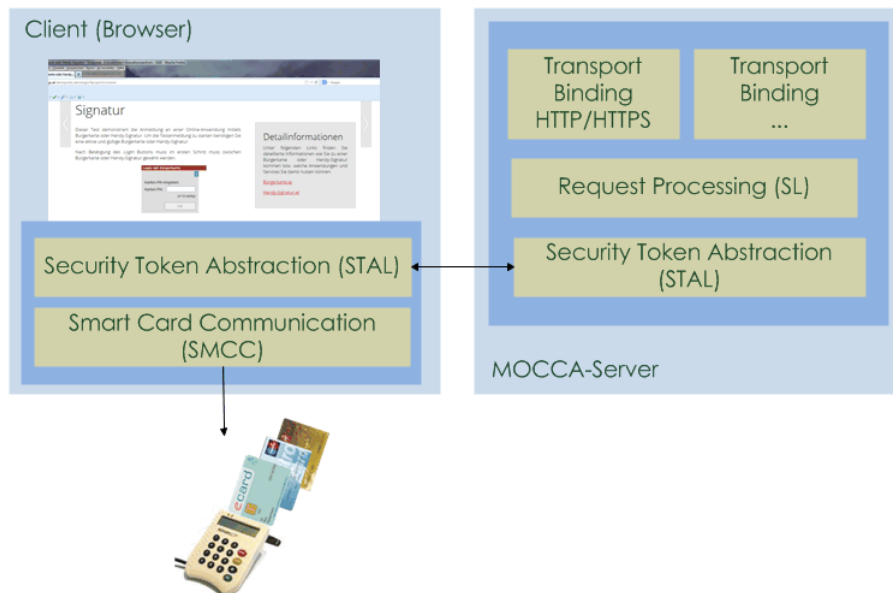


Figure 3.5: Implementation of the MOCCA layers[22]



### 3.2.5 Disparities of MOCCA Online and Local

Both solutions provide their own strengths and weaknesses and differ, at least at the moment, significantly in regard of their functionalities. The local citizen card environment provides access to the user's certificates, PIN management functionalities and an overview on the personal link and the hardware of the citizen card solution. The signature creation however, has not been implemented yet. On the other hand, MOCCA online provides a powerful and secure alternative for signature creation and authentication. A disadvantage of the applet is its high specificity for a certain task. The applet neither gives any feedback to the user, except for blocked/wrong PIN entries, nor are the certificates or the personal link accessible for the user. Thus, the local solution can be used as a "citizen card office", while the online solution provides more advanced functionalities like signature creation and authentication due to its specificity. In defiance to their disparities, both solutions implement the basic MOCCA architecture through its 4 main layers. The disparities of MOCCA online and local in regard of the implementation of these layers are shown in Figure 3.6. [20]

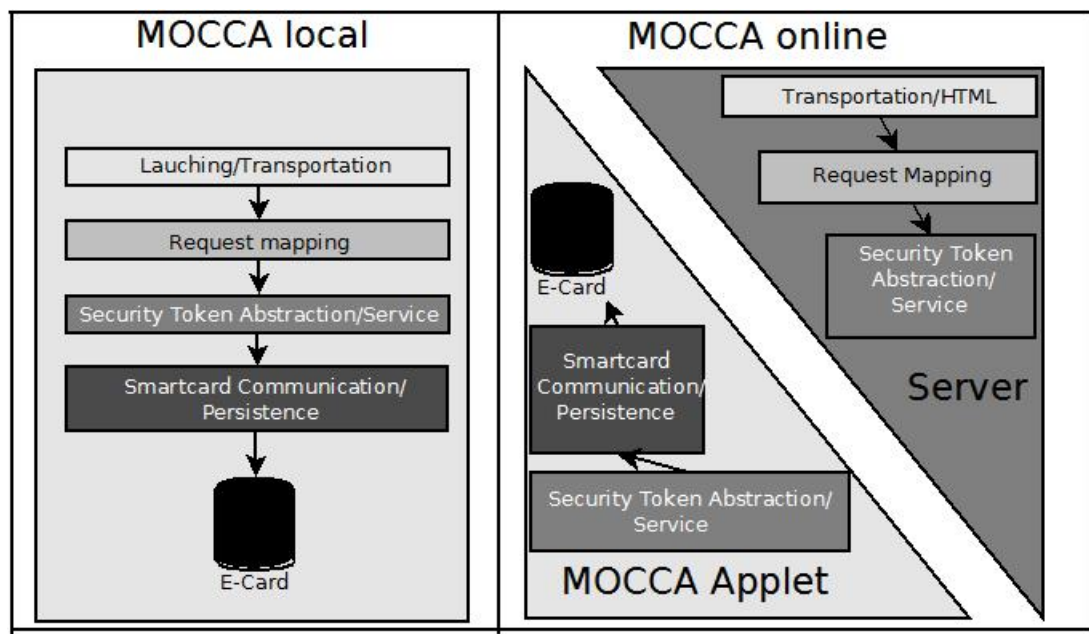


Figure 3.6: Disparities in regard of the layer implementation of MOCCA local and online

### 3.3 E-Card

The e-card is a smart card used by Austrian social insurance to relate medical treatment to the patient. In 2004, the Austrian parliament decided that the e-card should be extended by the citizen card software as an opt-in solution [30]. Since then, three generations of e-cards, used for the citizen card solution, have been developed and released. Table 3.3 holds an overview on the e-card generations that were used for the Austrian citizen card solution. Figure 3.7 gives a brief overview of the e-card.



Figure 3.7: An example for an e-card generation 4 [31]

#### E-Card Description

1. Social insurance logo
2. Braille
3. Social security number
4. Name of the owner
5. Card number

#### 3.3.1 Overview

The Austrian e-card includes two different types of data storage, a public part that is visible to anyone, and a private one that can only be accessed after authenticating with a PIN. The public part includes the user's social insurance number, name and card number. This data is printed on the front side of the card. The private part is stored in small files inside the smart card's chip. In general, this chip stores identifiers, like the ZMR-Number, certificates, user-related data, like social insurance number, date of birth, e.g., and signature-related data, like hash algorithms. One of these files, called the personal link file, is used to store the ZMR-number, the user's name, and other user data. This file is used in most citizen card applications to create a personal-link and authenticate the user. In addition, two certificates are stored on the card. [32] It is also used to store different algorithms like ECDSA-SHA, RSA-SHA and others that are needed for creating the signature hash. The creation of this hash is done by a signature application stored in a separated security environment on the card. This environment can only be accessed by authenticating with the signature PIN and executing special APDU (Application Protocol Data Unit) [33] commands like MSE (Manage Security Environment) or PSO (Perform Security Operation). Additionally, there are some PIN management applications stored on the card for validating, changing and unblocking the user's PIN.




Generation	Release Date	Short Description	Image
Generation 2 (G2)	2004	G2 had no Braille on it and contained just two ECDSA-key pairs. In a smooth transition, the G3 card was released in 2009. With 2014, the G2 support of the citizen card activation ultimately expired.	 The image shows a Generation 2 (G2) e-card. It is a light blue card with the 'ecard' logo at the top left and a small green logo at the top right. A red arrow points to a red 'X' mark on the card, indicating a feature or status. The card displays the number '001' and '1234 567890'. The name 'Dr. Magdalena Musterfrau' and the website 'www.sozialversicherung.at, Serviceline 00124 33 11' are printed at the bottom.
Generation 3 (G3)	December 2009	Braille on current e-cards was inserted with the G3 cards. This generation contains one ECDSA-key pair, used for a qualified certificate, and a RSA-key pair, used for a secure signature certificate. The G3 support of the citizen card activation is still being utilized.	 The image shows a Generation 3 (G3) e-card. It is a light blue card with the 'ecard' logo at the top left and a small green logo at the top right. A red arrow points to a red circle around a Braille character on the card. The card displays the number '001' and '1234 567890'. The name 'Magdalena Musterfrau' and the website 'www.sozialversicherung.at, Serviceline 00124 33 11' are printed at the bottom.
Generation 4	November 2014	The G4 cards have a slightly different color than the G3 cards and a new social insurance logo.	 The image shows a Generation 4 e-card. It is a light green card with the 'ecard' logo at the top left and a new green social insurance logo at the top right. The card displays the number '001' and '2481 290289'. The name 'Max Mustermann' and the website 'www.sozialversicherung.at, Serviceline 00124 33 11' are printed at the bottom.

Table 3.3: E-Card generations overview [31]

### 3.3.2 Data Documentation

Each data file on the card can be accessed in a similar way. First, one has to select the Master File (MF), which serves as a directory on the card. Next, one needs to know the location and the offset of the Elementary File (EF) one wants to read. Some special files however need to be processed by an application stored on the card. The XML-file containing the ZMR-number can be read from the card by simply using a few APDU commands in a specific order. One only needs to install PC/SC and either pycard for reading the data with python or other frameworks that are capable of communicating with a smart card. In addition, a smart card reader is required to establish a connection between the computer and the card.

Since APDU commands are the only way to communicate with the card, Table 3.4 gives little insight to the basic structure of those commands. In the following sections, it is presumed that the citizen card software was activated on the e-card. In general, such commands are given as bytes in hexadecimal numbers.

Cl	INS	P1	P2	Lc	Data	Le
00	A4	00	0C	00	–	00

Table 3.4: Structure of an APDU command

### APDU Structure Description [34]

- **Cl** - Class, defines the commands class of the APDU command, 1 byte
- **INS** - Instruction, specifies the executable command, 1 byte
- **P1** - Parameter 1, specifies the location of the file, 1 byte
- **P2** - Parameter 2, defines the offset between the start and the end of the file, 1 byte
- **Lc** - Command data length, 0 to 3 byte
- **Data** - Command data, Lc bytes
- **Le** - Response data length, 0 to 3 bytes

### Social Insurance Number

After transmission and execution of such commands, the card returns at least two bytes. These two bytes are commonly known as status word 1 (sw1) and 2 (sw2). A simple concatenation of sw1 and sw2 returns the given status word, for example "9000" for a given sw1=0x90 and sw2=0x00. This status word stands for a successful processed command. Prefabricated lists of those status words can be accessed in the ISO7816 standard for smart card communication. [34]

A fairly easy example of extracting personal linked data from the e-card is accessing the social insurance data file containing the social insurance number, the date of birth and the card holder name. Table 3.5 shows the necessary commands used to access and extract the social insurance file.

Nr	Cl	INS	P1	P2	Lc	Data	Le
1	00	A4	00	0C	00	–	00
2	00	A4	04	00	08	D0 40 00 00 17 01 01 01	FF
3	00	A4	02	04	02	EF 01	FF
4	00	B0	00	00	00	–	FF

Table 3.5: APDU list used to access the social insurance XML-file

## Command Description

1. **selectMF** - MF = Master File, the Master File for smart cards is equal to the root directory for computers
2. **execSELECT-AID-SV-PERSONENDATEN** - AID = Application Identifier, identifier for applications stored on the card. This command selects the AID of the personal data application.
3. **execSELECT-FID-GRUNDDATEN** - This command selects the FID of the EF "Grunddaten"
4. **Inputstream-Infobox** - The Inputstream reads the data stored on the selected file, in this case of the EF "Grunddaten".

After executing these commands, the returned byte output can be converted to the string shown in Figure 3.8.

```
1  0.0...*(.....1...XXXXXXXXXX0...*(.....1....0.. .U.*1...Thomas0. ..U.. 1
    ...Stipsits0...+.....1...19920731120000Z0...+.....1...M
```

Figure 3.8: Output Document XML-file Social Insurance Number

The "XXXXXXXXXX" string in Figure 3.8 substitutes the social insurance number stored on a real e-card. As it is evident from Figure 3.8, the date of birth of the card holder is also stored within the file. In this example it would be contained in the string "**19920731120000Z0**".

As indicated, it is quite simple to extract data from the files stored on the e-card. The following sections deal with further extraction of data from the citizen card.

## Personal Link

Table 3.6 show the APDU-commands used to read the XML-file from the e-card. It is important to use the commands in the given order, since they base on each other. Listing 6.8 contains a python script that can be used for reading the personal link data from the card. The python script is solely designed for use upon a G4 e-card.

Nr	Cl	INS	P1	P2	Lc	Data	Le
1	00	A4	00	0C	00	–	00
2	00	A4	02	04	02	00 32	FF
3	00	A4	04	00	08	D0 40 00 00 17 00 18 01	FF
4	00	A4	02	04	02	C0 02	FF
5	00	B0	00	00	00	–	FF
6	00	B0	01	00	00	–	FF
7	00	B0	02	00	00	–	FF
...	continue inputstream commands as long as the card returns sw1 = 0x90 sw2 = 0x00						...

Table 3.6: APDU list used to read the citizen card XML-file

## Command Description

1. **selectMF** - MF = selects the Master File
2. **selectFID-EF-Version** - selects the EF "Version"
3. **selectAID-Infobox** - selects the AID "Infobox"
4. **selectFID-EF-Infobox** - selects the EF "Infobox"
5. **Inputstream-Infobox** - The Inputstream reads the data stored on the selected file, in this case of the EF "Infobox".

Finally, the Inputstream command reads the selected file and produces the desired output, a personal link file that contains the user's name and the date of birth, a link to the user's profile in the ZMR system, the date and time of the citizen card registration and the user's ZMR number encrypted with Triple-DES (for example: "FndBNWQDBQ oNKB rXWwX5qA=="). The Triple-DES number is the core of the personal link file, since it is unique for every Austrian citizen and serves as authentication ID. It is not possible for an attacker to decrypt the Triple-DES number to the ZMR number. [31] An example for the output document is given in Figure 3.9.

```

1 AIK.....0.....&http://www.a-trust.at/zmr/persb205.xsl.*szr.bmi.gv.at-
AssertionID14182133052331744..2014-12-10T13:08:25+01:00.C0A..FndBNWQDBQ
3 oNKBrXWwX5qA==..Thomas.Johannes..Stipsits..1992-07-310.....
..*.....%.....9.....-..v0>..\d....A..<...*.Z.....h...I_....R@
5 .C.{N.*.n%.....4..x.I61J..U..Ru4J.....;..n_.^..(.....'.0.....
.W.QF.J...M....HM.....%...H.A.(Q...P.!..g6.....{.d_.X.}.I10....M...d.
7 ;Q..1...;...)..c7..uBj....O....B.7.#..a.....@h.;..c.F.k~M.dC.....
..~.nY.....^..%U.....AJ.u..|.Nc.o.Z.....
9 .....
.....
11 .....

```

Figure 3.9: Output document XML-file e-card

## Certificates

In addition to the personal link XML-file, there are two separate certificates stored on the E-Card. These certificates are activated with the citizen card registration and are valid until the card is disabled. Normally, these cards become disabled by requesting a new one due to the loss or expiration of the old card.

## Secure Signature Certificate

Table 3.7 shows the commands used to read the secure signature certificate. An example of the structure of such a secure signature certificate is given in Figure 3.10. Furthermore, Listing 6.6 contains a python script that is able to extract the secure signature certificate from a connected card.

Nr	Cl	INS	P1	P2	Lc	Data	Le
1	00	A4	00	0C	00	—	00
2	00	A4	04	00	08	D0 40 00 00 17 00 12 01	00
3	00	A4	02	04	02	C0 00	FF
4	00	B0	00	00	00	—	FF
4	00	B0	01	00	00	—	FF
4	00	B0	02	00	00	—	FF
...	continue inputstream commands as long as the card returns 0x90 0x00						...

Table 3.7: APDU commands used to read the secure signature certificate

## Command Description

1. **selectMF** - selects the Master File
2. **selectAID-AID-DF-SS** - selects the AID of the secure signature certificate application
3. **selectFID-EF-C-X509-CH-DS** - selects the FID of the EF-file that contains the secure signature certificate.
4. **readTransparentFile** - reads the EF with the given FID

```
1 0...0.....S.0...*.H.....0..1.0...U....AT1H0F..U...?A-Trust.Ges
..f..Sicherheitssysteme.im.elektr..Datenverkehr.GmbH1.0...U....a-sign-P
3 remium-Sig-021.0...U....a-sign-Premium-Sig-020...141210120829Z..1912101
10829Z0t1.0...U....AT1!0...U....Thomas.Johannes.Stipsits1.0...U....Stip
5 sits1.0...U...*.Thomas.Johannes1.0...U....1027492128110Y0...*.H.=...*.H
.=...B..yXB./O&....hh..;..[.M....Y.*...&..X....V..Y%.81%.L*...<.Z8)%
7 .....0...0...U.....B.*...8g0...U.....0...U.#..0...M...K...0...U
....0.0{...+.....o0m0B...+.....0..6http://www.a-trust.at/certs/a-sign-
9 Premium-Sig-02a.crt0'..+.....0...http://ocsp.a-trust.at/ocsp0Y..U...R0P
0D.*(...0:08...+.....,http://www.a-trust.at/docs/cp/a-sign-Premium0
11 .....0..0'..+.....0.0.....F..0...+.....0....U.....0..0.....
....ldap://ldap.a-trust.at/ou=a-sign-Premium-Sig-02,o=A-Trust,c=AT?cer
13 tificaterevocationlist?base?objectclass=eidCertificationAuthority0...*.
H....."Ev..H.....L.b....-..f.N.m..Z....5&m#x...!.&?...+J.....
15 ...<^|...5.l.k....^>s:...O].q0.2..d..%.VqniL..i...A..M]...q.39.....
F.j{..(....R..ud...Q/.2R....dvR.P....ww.I_$.x.3W.....W...Zb...lR...
17 ..BY.iJe....C....].0sly..i....J.....m...L|..!{k...':s..ji
```

Figure 3.10: Output document secure signature certificate e-card

## Qualified Certificate

In addition to the secure signature certificate, there is also a qualified one stored on the citizen card. The qualified certificate can be accessed in a similar way. Table 3.8 shows the corresponding APDU commands, Figure 3.11 gives an example of the structure of such a qualified certificate. Listing 6.7 contains a python script that reads the qualified certificate from a connected card.



Nr	Cl	INS	P1	P2	Lc	Data	Le
1	00	A4	00	0C	00	–	00
2	00	A4	04	00	08	D0 40 00 00 17 00 13 01	FF
3	00	A4	02	04	02	2F 01	FF
4	00	B0	00	00	00	–	FF
4	00	B0	01	00	00	–	FF
4	00	B0	03	00	00	–	FF
...	continue inputstream commands as long as the card returns 0x90 0x00						...

Table 3.8: Commands used to read the qualified certificate

### Command Description

1. **selectMF** - selects the Master File
2. **selectAID-AID-DF-GS** - selects the AID of the qualified certificate application
3. **selectFID-EF-C-X509-CH-AUT** - selects the FID of the EF-file that contains the secure signature certificate.
4. **readTransparentFile** - reads the EF with the given FID

```

1 0...0.....S.0...*.H.....0..1.0...U....AT1H0F..U...?A-Trust.Ges
..f..Sicherheitssysteme.im.elektr..Datenverkehr.GmbH1.0...U....a-sign-T
3 oken-031.0...U....s1.0...U....Stipsits1.0...U.*..Thomas.Johannes1.0...U
....1027492128110.."0...*.H.....0.....j.....l#],....K...
5 ...3[uxbW.\.o}m....x.p.<V...E...j..d"I....k...&.G...6.l....t...A.*.4
..(..T~0M0K.N1....T.k.%.....f.r.5.?.\.....#.G....tx2SEM.....9&
7 ]s;.....&|j.....=>..x8G..6<]>...1<..._.....W...q....H.....
.O...vXK,g...#.M..vb.....0...0t..+.....h0f0;..+.....0../http://
9 www.a-trust.at/certs/a-sign-Token-03.crt0'..+.....0...http://ocsp.a-tru
st.at/ocsp0....U.....0..0.....}ldap://ldap.a-trust.at/ou=a-sign-toke
11 n-03,o=A-Trust,c=AT?certificaterevocationlist?base?objectclass=eidCerti
ficationAuthority0...U.#...0...E.R..Uee0...U...0.0...U.....G....1.A0..
13 .U.....0M..U...F0D0B.*((...0806..+.....*http://www.a-trust.at
/docs/cp/a-sign-token0...*.H.....'p...&....C.c^w,..9...F....
15 ..k.!...G)f.E..fL.gL_v....;..Ft.^...T.....&.>...(..9t.."u..z...3..S.%
.._...a...{N.d...x.{.iD&....{.....~.6]h..^...sG.y.I..p.2.b.xH..p
17 l....W'.K....+3..i.....i..'\&<.....(s....Vm:r|!...3....&..-^./4
".H...T.)...?

```

Figure 3.11: Output document qualified signature e-card

## Signature Hash

In addition to the data, the e-card also contains a variety of different hash algorithms which can be used to create the digital signature. These algorithms, in combination with the private key stored on the card, are used for creating a signature hash value, which is the essence of the qualified signature. Currently, the only way to create such hash values is to use an online signature creation in combination with the aforementioned MOCCA online solution. To create this signature value, the Web server transmits a hash of data to be signed to the applet, which passes it along to the card. After receiving such a request, the applet basically performs the commands shown in Table 3.9 on the card. This signature hash varies each time the operation is performed, even for equivalent inputs.

Nr	Cl	INS	P1	P2	Lc	Data	Le
1	00	A4	00	0C	00	–	00
2	00	A4	04	00	08	D0 40 00 00 17 00 12 01	FF
3	00	20	00	81	08	?? ?? ?? ?? ?? ?? ?? ??	00
4	00	22	41	B6	08	84 03 80 02 00 80 01 04	00
5	00	22	41	AA	03	80 01 40	FF
6	00	2A	9E	9A	??	??	FF

Table 3.9: Commands used to create the signature hash of the given hash value

### Command Description

1. **selectMF** - selects the Master file
2. **selectAID-AID-DF-SS** - selects the AID of the secure signature certificate application
3. **validatePINEntry** - validates the given PIN, enables the MSE-Operations, the insertion of the PIN into the APDU is not trivial and needs computing. "??" stands for the inserted PIN Bytes
4. **execMSE** - manages the security environment, initializes the hash algorithm
5. **execMSE** - manages the security environment
6. **PSOcreateSignature** - performs security operation, creates the signature, that signs (hashes) a hash value, the ?? stands for the data bytes (the given hash value) and the length of data

Listing 6.9 implements these APDU commands and is capable of recognizing popular errors like a blocked PIN. The script can handle different lengths of hash values between 0 and 256 bytes. In addition, the validation of the PINs needs to be considered, because both, the signature PIN is blocked after 3 failed retries, while the card PIN is blocked after 10 failed retries [18]. Furthermore, it is not possible to unblock a PIN on cards using the STARCOS operating system. Therefore, the citizen card becomes useless for authentication and signature creation applications once the PIN has been blocked.

### 3.4 Possible Vulnerabilities

Before analysing the specific weaknesses of MOCCA, this thesis aims to investigate the basic structure of MOCCA's architecture for its very own possible weak points. Since most of the citizen card applications are online solutions, the basic structure of the citizen card offers some possible weak points where an attacker may possibly infiltrate. MOCCA bases on a common client-server architecture where the server is determined by the chosen application.

Furthermore, the client (the applet) accepts only one user input, the PIN, which could also possibly be a weak point of the implementation. The connection between the Web server and the client is another vulnerable point that may be attacked. Other attacks however could possibly try to substitute the client part with a fake applet in order to smurf keys or gain access to the data stored on the card.

The MOCCA Web servers may also be another vulnerable point. These servers could not be investigated during the work on this thesis. An insider could leak valuable information on how to attack these parts of the citizen card solution.

Another approach could center on getting hold of the e-card, either by obtaining the physical card or by phishing attacks that pipeline the data read from the card to a modified applet. If this succeeds, an attacker could replace the common citizen card applications with the stolen card in order to deceive the system and gain access to the applications.

The following list contains a summary of possible weak points that were considered while working on this thesis. Figure 3.12 illustrates these points during an authentication process. In addition, they will be discussed in the following section.

#### **Possible weak points:**

- a. User Input**
- b. Applet**
- c. Client-server connection**
- d. MOCCA servers**
- e. E-Card**

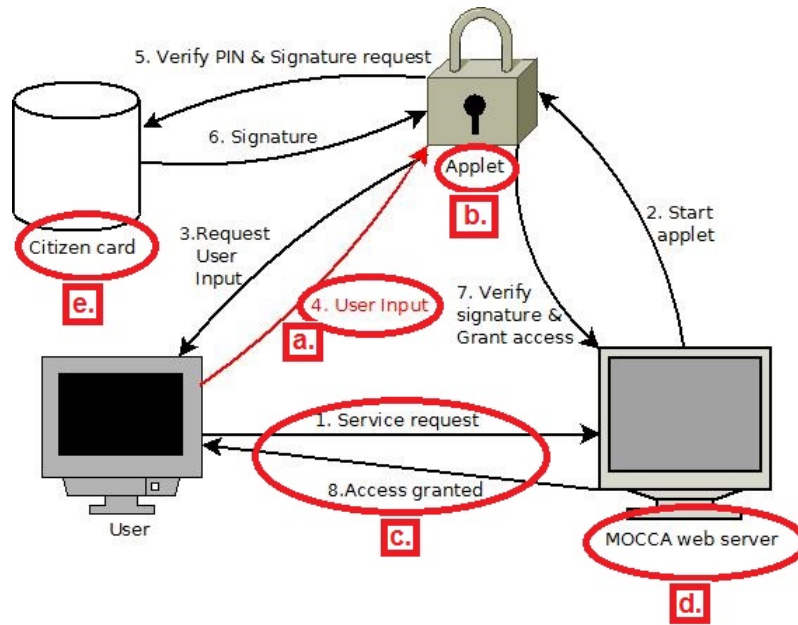


Figure 3.12: Vulnerabilities of MOCCA's online applet authentication

### 3.5 Discussion of Vulnerabilities

After identifying possible weak points, it is urgent to determine whether MOCCA is vulnerable for attacks that target these points. Foremost, the structure of the services and their level of security need to be determined.

- a. **User Input:** MOCCA online is used as authentication software and therefore only accepts one single input, the PIN. This PIN is queried by the applet and needs to be a minimum of 6 and a maximum of 12 characters long. The PIN entry label only accepts numbers from 1 to 9 in decimal number format. Thus, the MOCCA online solution restricts the input to a limit of characters following a whitelist concept for ensuring the secure evaluation of the user input. This input is only used for authenticating with an application that is stored inside the security environment of the card. Thus, MOCCA online can not be corrupted by any input based attacks. Whether the card system itself could be corrupted by any input based attacks could not be determined during the work on this thesis because there is neither a way for extracting the authentication algorithm, nor any documentation available on the Internet.

**b. Applet:** Since MOCCA online is presented as an open source project, any attacker can gather information of specific appearance and functionality of the applet. In addition, the JAR-file of the applet can be accessed easily, since it is downloaded automatically during processing any authentication with MOCCA online. According to this information, it is possible for type 0, 1 and 2 attackers to set up an applet that basically appears to be the same as the original applet. Furthermore, any attacker can extract all data except for the private key from the card, since all APDU commands used by the applet were hard coded and therefore accessible by decompiling the code. Thus, once the attacker decompiles the applet JAR he has all the information he needs to set up a fake applet which is able to extract the personal link data, the certificates and even the public keys if the user is tricked into entering his pin into the fake applet. Taken en masse, the MOCCA online solution is vulnerable to fake applet attacks, although the success of such an attack depends on the knowledge and the attention of the user.

**c. Client-Server connection:**

Basically, the client-server connection is driven by XML-documents that are passed between the client and the server. The binding of the connection is done by a "SOAP" binding, and the requests are mapped according to predefined names on the correct request class. Listing 6.1 in the Appendix shows the mapping of the request types. Such communication can be tracked by enabling the Java console during an authentication process. A snippet of such a connection is given in Listing 6.2. In addition, there are a few official documents available on the Internet that explain the interface of the applet and the server. [35] Thus, an attacker is capable of obtaining the knowledge needed for implementing an applet that can handle server requests. Listing 6.3 shows an example of such XML request documents specified in an official tutorial for application developers offered by the EGIZ. [35] These XML documents are transmitted through a SSL connection to the server. While creating this connection with a SSL-socket, the applet checks whether the server, to which it is connected, is a government server by verifying the peer certificate used to establish the session. Thus, it is not possible for the attacker to perform a simple man in the middle attack. Listing 6.4 shows the method used to determine whether the connecting server is a government server or not. Therefore it is neither possible for an attacker to perform a simple MITM attack, nor a SSLStripping attack.

**d. MOCCA Servers:**

Compromising an entire server would be a severe attack to the citizen card authentication and would cause substantial damage to the current authentication solution. Although the leak of hundreds or even thousands of public keys might be rather benign, since the key is just one of the two factors used to authenticate, such an incident would cause severe damage to the confidence the users maintain to this service. Furthermore, hundreds of documents may get leaked and attackers could manage to disrupt the whole authentication process to gain access without dealing with the two factor authentication. Thus, the most fatal attacks would aim on getting hold of one of these servers. Since there is no information about the server part of MOCCA available, an attacker would need some insider information, what makes such attacks performable by just a narrow group of professionals and insiders. In our classification such attacks could just be performed by type 1 and possibly type 2 attackers, if they have access to insider information about the server structure.

- e. E-Card:** Since most data on the e-card is useless after breaking the authentication, it is not profitable to focus on hacking just the smart card. However, this does not mean that the card has no weakness in the overall MOCCA solution. Since MOCCA heavily relies on the two factor authentication, it is urgent to consider scenarios where the attacker manages to get hold of the card and the PIN. After getting hold of these 2 factors of the authentication process, the attacker gains the ability to hijack the identity of his victim without damaging the system or being more suspicious for the system than a normal user. Such attacks have a very specific scope and do not affect other users or the system.

### 3.6 Potential Scenarios

Since the EGIZ offers an open-source platform for MOCCA, it is likely that an attacker can at least gather information about the MOCCA client and the applet. Thus, an attacker could possibly gain low-mid knowledge of the system. Although the MOCCA Web servers are not accessible for reverse engineering, a possible attacker could use the platform for developing a variety of attacks that aim to spoof the signature PIN of a specific user.

The biggest strength and weakness of MOCCA is that the system relies heavily on the two factor authentication of the card and the PIN. For applying the functions of MOCCA in a useful way, one needs to know the PIN and obtain physical hold of the card. Thus, a possible attack has to cover both parts, the knowledge (the PIN) and the physical component (the card). Such attacks therefore do not need abundant of resources and can be performed by all three types of attackers.

Most current Internet users do not use available services with necessary diligence and therefore put themselves or their data into danger. Since a chain is always as strong as its weakest link, the user himself is a big weakness of MOCCA. It is possible that computers of different users become infected by exploits that were developed for gathering information, passwords and other data. In the past, many exploits, like Trojans or Key-loggers circulated. Once the citizen card environment gains currency, it is possible that organized groups of professionals develop Trojans or other exploits that are capable of gathering the data stored on the card. Once the attackers retrieve the citizen card data, the only thing preventing them from hijacking the identity is the physical component in the two factor authentication. Since the private key is not extractable from the card, it is not possible for the attackers to break this security measure with digital attacks. Thus, the attackers need to obtain hold of the physical component or trick their victims into a phishing attack. This kind of attack is likely to be performed by type 2 attackers, since it is a highly resource-consuming process to develop such an exploit and distribute it.

Another weak point could be the communication between the applet and the Web server. If one could manage to perform a man in the middle attack or even some sort of SSLStripping attack, the attacker could hijack the connection of a victim that is communicating with the MOCCA Web server and neither needs a PIN, nor the physical component. This attack might be the worst case scenario for the EGIZ, since it is not very elaborate to perform, but one only needs the knowledge of the server interface and possible countermeasures that could be implemented on the server side. Thus, such an attack will most likely be performed by a type 1 attacker, because without insider knowledge the attack could be detected and fail.

### 3.7 Potential Impact

Unlike the German health card [36], the Austrian e-card contains no sensitive data like medical records. Even though the files contained in the e-card solely store personal link data like name, the date of birth, and the ZMR-Number, it is very likely that an attacker can use this dataset for tracking his victim in order to get hold of the physical component. After successfully obtaining access to the victims identity, however, this dataset becomes rather useless. The real goal is to gain access to citizen card enabled e-government services, and possibly future commercial applications, in order to obtain way more data about the victim. A good example of such service would be the FinanzOnline tool [37] hosted by the BMF. This application offers the user a citizen card only login, which enables a possible attacker to gather all financial information that the victim had to share with the federal ministry of finance. This is a specific example of a security breach in which an attacker infiltrates the two factor authentication.

Currently, it is possible to manage every administrative work in regard of e-government with the citizen card authentication. Thus, there is a wide range of usage an attacker gains access to. In addition to the e-government applications the attacker can possibly abuse the signature creation methods for malicious purposes. Signing contracts is only the tip of the iceberg, the range of applicability of the citizen card signature is vast. Possible aims can cover defamation, deception, pecuniary loss, fraud and last but not least other criminal activity performed with the obtained identity. Such attacks can cause severe damage to the image of the victim, lead to financial collapse, aspersions, or even wrong judgement. Thus, such specific attacks can eventually destroy the whole life of the victim. If we apply these threats to a scenario where the victim holds office of a leading position in a national or international operating corporation, these threats gain an even bigger dimension, since an attacker could be able to counterfeit sensitive documents, and gain access to business-related data of the subject corporation and other high risk data.



## Attack Example

This chapter will introduce a possible attack vector on the current citizen card authentication solution. Table 4.1 describes the characteristics of the attacks.

Description	Impact on the system	Impact on a single User	Probability	Complexity
PIN Spoofing	none-low	mid-high	low-mid	low

Table 4.1: Attack Vector Description

### 4.1 PIN Spoofing

The PIN Spoofing attack aims to spoof the victim's PIN, locate the victim and get hold of his e-card. Since the entire MOCCA applet can easily be analysed by reverse engineering, it is possible, and very likely, that an attacker could imitate the original applet and therefore easily gain access to the signature PIN, and even all the files stored on the e-card. With the obtained data, an attacker could create a fake lookalike of the victim's real e-card. Furthermore, an attacker could be capable of determining the identity of his victim and locating him with the data stored on the card through social platforms, websites of employers or other data sources. Next, the attacker could possibly use different measures like social engineering or even pickpocketing to get hold of the card of his victim. After accomplishing this, the attacker has everything he needs for signing documents with the identity of his victim, or even worse, through the authenticating possibility access to certain services like FinanzOnline (an e-government application used for handling finances).

This attack can be distributed in 5 different phases described in the following sections.

#### **4.1.1 Phase 1: Setting up**

Phase 1 analyses the look of the original applet, obtaining knowledge of the card communication algorithms, and setting up the attack environment and the fake applet. The basic environment of such an attack may consist of a fake citizen card application website, a server for storing the website, and the data read from the card and the fake applet itself.

Since the only functionality this fake applet contains is reading all of the data stored on the card and saving it together with the given PIN, the implementation effort is fairly low. It is urgent for the attacker to maintain the look of both, the applet and the website. For deceiving as many victims as possible into entering their information into the applet, it is necessary to give the impression of a trustworthy application. After integrating the malicious applet into the fake authentication possibility on his website, the trap is set up, and the attacker may proceed with phase 2.

#### **4.1.2 Phase 2: Distributing**

The distributing phase primarily deals with spreading the website to a mass of potential victims. An attacker may use different distribution channels like forums, spam mails, bot nets or maybe even social platforms. There are various ways an attacker can distribute his trap over the Internet. Since many of the addressed users will not fall into his trap it is likely that an attacker will try to spread his link as far as possible.

Since distributing the link to his trap is the essential part for addressing potential victims, an attacker may never stop with phase 2. Thus, phase 2 can be considered a continuous phase throughout executing this attack. As soon as the first datasets are obtained, the attacker may proceed with phase 3.

#### **4.1.3 Phase 3: User profiling**

User profiling, in terms of the PIN spoofing attack, can be accomplished by analysing the obtained datasets from the e-card. Since the G4 e-card contains user linked data like forename and surname, social insurance number, date of birth and academic degree, it is very likely that an attacker can track the victim down using social platforms like Facebook or even the website of the victim's employer. This enables an attacker to perform phase 4 of the PIN spoofing attack.

In addition to phase 2, an attacker may execute the user profiling as an continuous phase. After tracking the victim, an attacker may proceed with phase 5.

#### **4.1.4 Phase 4: Obtaining the card**

The goal of Phase 4 is to obtain access to the physical component of the two factor authentication process, the victim's citizen card. This phase is key for infiltrating the authentication and can be performed in different ways. Some possible procedures could be pickpocketing, social engineering or other types of fraud or theft.

Since the theft of the e-card can be noticed early on, an attacker has to ensure that he has enough time to perform further attacks before the victim notices the theft of his citizen card. One way to accomplish a deception of the victim, is to replace the real citizen card with a fake dummy card that has the same appearance as the victims e-card. Since phase 2 can provide the attacker with enough data about the victim's card like version number, card expiration date and personal linked data mentioned in phase 3, it is possible that an attacker could create such a dummy smart card. Such a dummy card can not be distinguished from the original by a layman. Thus, the attacker gains significantly more time to abuse access to the user's citizen card features. After getting hold of the physical card, an attacker holds all credentials he needs for continuing with phase 5.

#### **4.1.5 Phase 5: Breaking the authentication**

Phase 5 spells the end for the PIN spoofing attack and the beginning for acquisition and saturation of the victim's data that can be accessed by the citizen card framework. Since the attacker achieved to acquire the PIN in phase 2 and the smart card in phase 4 there is no further barrier to the attackers' attempts. Furthermore, it gives the attacker access to a wide spectrum of the citizen's e-government data and communication and access to a variety of different citizen card applications.

### **4.2 Additional Findings**

In addition to the PIN Spoofing attack, there is another possible attack vector that bases on a phishing attack. Since the MOCCA online environment is designed for a wide variety of applications, a possible attacker could set up a fake applet, rework the official applet and therefore gain access to the user's ID. Since there is no applet validation implemented on the physical component, i.e. the e-card, it is possible for any attacker to access the personal binding data. Based on this principle, an attacker could modify the official applet to receive the physical data obtained by his faked applet and transmit it to the MOCCA Web server. During the research on this thesis, it could not be determined whether or not the Web server performs an applet hash check or not. If not, an attacker would be able to replace the official applet with the modified one, and therefore obtain access to the victims citizen card through setting up a pipeline between his fake applet, which just reads the desired data from the victims card, and computer. Figure 4.1 shows a graphical interpretation of the mentioned fake applet attack.

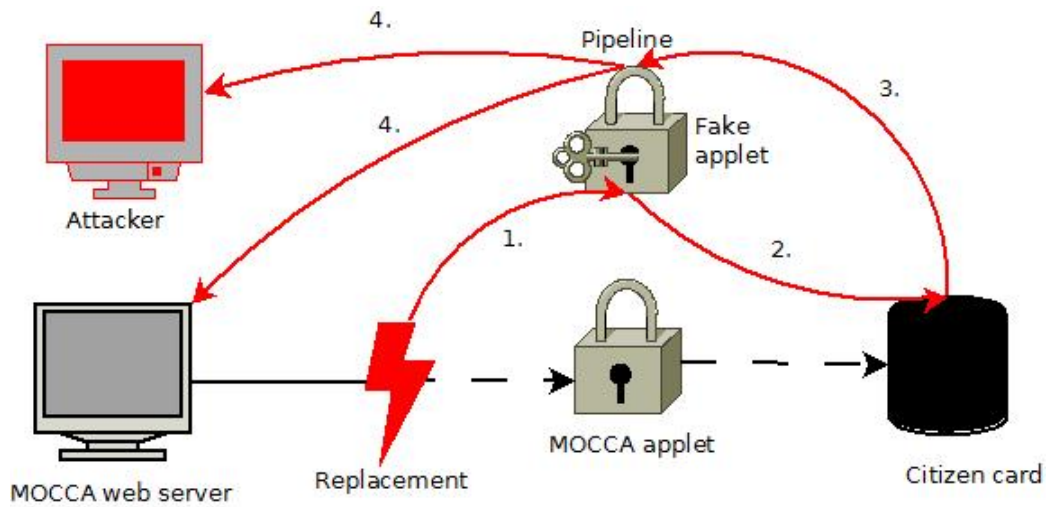


Figure 4.1: Chaining of the fake applet attack

1. **Replacement:** Replacement of the original MOCCA applet with the attacker's modified one.
2. **Authentication request:** Since the attack uses a real authentication request from a MOCCA Web server, the attacker easily gains access to the personal binding files stored on the victim's citizen card.
3. **Authentication response:** Due to the government certificate, that is encapsulated in a MOCCA Web server request, the citizen card treats the request like a normal one and sends an authentication response.
4. **Applet response:** The attacker could modify the applet in different ways, for example simultaneously sending a response to the server and the attacker.

# Results and Conclusion

## 5.1 Results

The Austrian citizen card solution relies primarily on the two factor authentication combined with a private key secured by two different PINs. Since the PINs are not extractable, an attacker needs to obtain access to the physical component. This access can be established in two different ways, either by getting hold of the card or by modifying the official applet for creating a data pipeline with a fake applet. Since it was not possible to determine whether the MOCCA Web servers use a code hash of the applet to determine their authenticity, no comment on this attack vector can be given.

In comparison to common client-server based online applications, MOCCA offers a fairly secure environment for his users. As mentioned in Section 3.5, MOCCA deals quite well with the most common vulnerabilities of comparable systems. Since the citizen card still has a meagre distribution in Austria and is only used by few citizens the potential threat level for the user seems to be fairly low. In addition, there is still a long way to go until the development of MOCCA will be finished.

The local version, for example, is not able to perform signature creation with G4 cards, although the most necessary code parts are already integrated. Furthermore, most of the APDU-commands implemented in the local version are outdated and a huge part of this system is unusable at the moment. Basically all functionality it offers is to check the personal binding and extracting the certificates from the card.

In spite of the unusable local version, the online applet is up to date and can perform a variety of different tasks. The applet receives a government certificate from the server that ensures the authenticity of the connection and uses TLS to transmit the data. Therefore, the applet-server connection can be considered secure. The connection between the card and the applet, however, appears vulnerable. Since the applet can be downloaded, decompiled and therefore reworked by anyone, the modification of this applet is not a difficult task.

The card itself is divided into two separate parts, a security environment and the basic data. The security environment includes the private key and the signature algorithms while the basic data consists of certificates, personal binding and basic user data. Since there is no access validation, anyone can access the basic data except for the personal binding and the security environment. For accessing the personal binding one needs the card PIN of the user. The security environment needs to be managed by an application stored on the card. This application can only be run after authenticating with the signature PIN of the user. Both, the card and the signature PIN, are chosen at the registration of the citizen card.

## 5.2 Conclusion

Since anyone who decompiles the applet code may obtain knowledge of the APDU commands and the card itself has no security checks for incoming commands, it is possible to gain access to the basic data on each citizen card. With phishing matters like setting up a fake applet, anyone can gather PIN data and therefore obtain access to all data and functionality stored on the card. As mentioned in the previous chapter, there are possible attack vectors which could enable an attacker to hijack a victim's identity.

## 5.3 Outlook

A suggestion for improvement of the current solution is to implement a code hash check on both, the server and the card side. Such a code hash could be transmitted at the start of the communication between the applet and the card, and could serve as a type of an applet PIN for accessing the cards citizen card data and functionalities. This security measure would prevent attackers from accessing the personal binding data and signature creation algorithms. Furthermore, a server sided code hash check of the applet could indicate whether someone uses a modified applet for accessing the citizen card functionalities online.

# Bibliography

- [1] **EGIZ. MOCCA - Modular Open Citizen Card Architecture.** <https://www.egiz.gv.at/de/schwerpunkte/9-MOCCA>, 2015. [Online; accessed 30-April-2015].
- [2] **Österreichische Stammzahlenregisterbehörde. Bildung von Stammzahlen, Ersatzstammzahlen und davon abgeleiteten Personenkennzeichen.** <https://www.stammzahlenregister.gv.at/site/6001/default.aspx>, 2015. [Online; accessed 02-June-2015].
- [3] **A-SIT Zentrum für sichere Informationstechnologie - Austria. Bürgerkartenumgebungen.** <https://www.buergerkarte.at/downloads-karte.html>, 2015. [Online; accessed 04-April-2015].
- [4] **Statistik Austria. IKT-EINSATZ IN HAUSHALTEN, Einsatz von Informations- und Kommunikationstechnologien in Haushalten 2012.** [http://www.e-government.steiermark.at/cms/dokumente/10103295\\_34808287/8f9aabca/IKT-Einsatz\\_in\\_Haushalten\\_2012.pdf](http://www.e-government.steiermark.at/cms/dokumente/10103295_34808287/8f9aabca/IKT-Einsatz_in_Haushalten_2012.pdf), 2012. [Online; accessed 30-April-2015].
- [5] **A-SIT Zentrum für sichere Informationstechnologie - Austria. Über A-SIT.** <https://www.a-sit.at/de/allgemein/asit.php>, 2015. [Online; accessed 02-June-2015].
- [6] **A-SIT Zentrum für sichere Informationstechnologie - Austria. Die vielfältigen Einsatzmöglichkeiten der Bürgerkarte.** [http://www.a-sit.at/de/dokumente\\_publicationen/flyer/bk\\_tools\\_private.php](http://www.a-sit.at/de/dokumente_publicationen/flyer/bk_tools_private.php), 2015. [Online; accessed 30-April-2015].
- [7] <https://www.prime-sign.com/mocca/applet/BKUApplet-single.jar>, 2015. [Online; accessed 30-April-2015].
- [8] **A-SIT Zentrum für sichere Informationstechnologie - Austria. Handy-Signatur und Bürgerkarte.** <https://www.buergerkarte.at/anwendungen-karte.html>, 2015. [Online; accessed 24-May-2015].
- [9] **EGIZ. MOCCA.** <https://www.egiz.gv.at/de/schwerpunkte/9-MOCCA>, 2015. [Online; accessed 24-May-2015].

- [10] **PC/SC Workgroup. PC/SC Workgroup Specifications Overview.** <http://www.pcscworkgroup.com/specifications/overview.php>, 2015. [Online; accessed 30-April-2015].
- [11] **ISO, IEC. Identification cards – Integrated circuit cards – Part 4: Organization, security and commands for interchange.** [http://www.cardwerk.com/smartcards/smartcard\\_standard\\_ISO7816-4.aspx](http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4.aspx), 2013. [Online; accessed 30-April-2015].
- [12] **ISO, IEC. Identification cards – Integrated circuit cards – Part 8: Commands for security operations.** [http://www.iso.org/iso/home/store/catalogue\\_tc/catalogue\\_detail.htm?csnumber=66092](http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=66092), 2004. [Online; accessed 30-April-2015].
- [13] **PhD. Primiano Tucci. JSmartCardExplorer: Cross-platform, low level APDU smart card development.** <https://www.primianotucci.com/os/smartcard-explorer>, 2015. [Online; accessed 12-May-2015].
- [14] **Emmanuel Dupuy. Java Decompiler - Yet another fast Java decompiler.** <http://jd.benow.ca/>, 2015. [Online; accessed 12-May-2015].
- [15] **Eric Young and Tim Hudson. OpenSSL - Cryptography and SSL/TLS Toolkit.** <https://www.openssl.org/>, 2015. [Online; accessed 17-May-2015].
- [16] **Michael V. Scovetta. Yasca - "Yet Another Source Code Analyzer".** <http://www.scovetta.com/yasca.html>, 2015. [Online; accessed 17-May-2015].
- [17] **EGIZ. MOCCA local.** <https://www.egiz.gv.at/de/schwerpunkte/9-MOCCA#sub-MOCCA-Lokal>, 2015. [Online; accessed 30-April-2015].
- [18] **IT Solution. PINs of the E-Card.** <https://www.itsolution.at/ecard-pins.html>, 2015. [Online; accessed 30-April-2015].
- [19] **PrimeSign GmbH und Datentechnik Innovation GmbH. PrimeSign.** <https://www.prime-sign.com/>, 2015. [Online; accessed 01-June-2015].
- [20] **Martin Centner, Clemens Orthacker, Wolfgang Bauer. Workshop zur Einführung der MOCCA Online-BKU.** <https://www.digitales.oesterreich.gv.at/DocView.axd?CobId=33421>, 2009. [Online; accessed 10-May-2015].
- [21] **Pivotal Software. Spring - Understanding REST.** <https://spring.io/understanding/REST>, 2015. [Online; accessed 01-June-2015].
- [22] **EGIZ. MOCCA online.** <https://www.egiz.gv.at/de/schwerpunkte/9-MOCCA#sub-MOCCA-Online>, 2015. [Online; accessed 30-April-2015].



- [23] **A-SIT Zentrum für sichere Informationstechnologie - Austria. BÜRGERKARTENUMGEBUNG - MOCCA** . <http://webstart.buergerkarte.at/mocca/>, 2015. [Online; accessed 30-April-2015].
- [24] **A-SIT Zentrum für sichere Informationstechnologie - Austria. Codebase for Mocca JNLP**. <http://webstart.buergerkarte.at/mocca/webstart/>, 2015. [Online; accessed 02-June-2015].
- [25] **EGIZ. MOCCA Ports**. <https://apps.egiz.gv.at/bkuonline/help/de/help.ports.html>, 2015. [Online; accessed 30-April-2015].
- [26] **EGIZ. MOCCA FAQ**. <https://www.egiz.gv.at/de/schwerpunkte/9-MOCCA#sub-FAQ>, 2015. [Online; accessed 30-April-2015].
- [27] **Frank Cornelis. eID Security**. <http://www.oecd.org/mena/governance/41842762.pdf>, 2008. [Online; accessed 02-June-2015].
- [28] **Trueb Baltic AS . EstEID v. 3.5 - Estonian Electronic ID-Card application specification** . [http://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3\\_5-20140327.pdf](http://www.id.ee/public/TB-SPEC-EstEID-Chip-App-v3_5-20140327.pdf), 2013. [Online; accessed 02-June-2015].
- [29] **A-Trust Gesellschaft für Sicherheitssysteme im elektronischen Datenverkehr GmbH. a.trust Liste der empfohlenen Komponenten und Verfahren**. <http://www.signatur.rtr.at/repository/csp-atrust-products-1712-20111010-de.pdf>, 2013. [Online; accessed 03-June-2015].
- [30] **Parlament Republik Österreich. Parlamentskorrespondenz Nr. 40 vom 22.01.2004**. [http://www.parlament.gv.at/PAKT/PR/JAHR\\_2004/PK0040/](http://www.parlament.gv.at/PAKT/PR/JAHR_2004/PK0040/), 2015. [Online; accessed 30-April-2015].
- [31] **A-SIT Zentrum für sichere Informationstechnologie - Austria. MOCCA Glossar**. <https://www.buergerkarte.at/glossar.html>, 2015. [Online; accessed 30-April-2015].
- [32] **SVC - Sozialversicherungs-Chipkarten Betriebs- und Errichtungsgesellschaft m.b.H. The Austrian e-card System**. <http://www.chipkarte.at/portal27/portal/ecardportal/content/contentWindow?contentid=10007.678587&action=2>, 2015. [Online; accessed 30-April-2015].
- [33] **DATACOM Buchverlag GmbH. APDU (application protocol data unit)**. <http://www.itwissen.info/definition/lexikon/application-protocol-data-unit-APDU-Anwendungsprotokoll-Dateneinheit.html>, 2015. [Online; accessed 30-April-2015].

- [34] **Jacquinot Consulting, Inc. . CardWer - Smarter Card Solutions: ISO 7816-4: Interindustry Commands for Interchange).**  
[http://www.cardwerk.com/smartcards/smartcard\\_standard\\_ISO7816-4\\_5\\_basic\\_organizations.aspx](http://www.cardwerk.com/smartcards/smartcard_standard_ISO7816-4_5_basic_organizations.aspx), 2015. [Online; accessed 30-April-2015].
- [35] **Arno Hollosi, Gregor Karlinger, Thomas Rössler, Martin Centner, Andreas Fitzek, Tobias Kellner, et al. Tutorium zur österreichischen Bürgerkarte.** <https://www.buergerkarte.at/konzept/securitylayer/spezifikation/20140114/tutorial/tutorial.html>, 2015. [Online; accessed 30-April-2015].
- [36] **Bundesministerium für Gesundheit. Die elektronische Gesundheitskarte.** <http://www.bmg.bund.de/themen/krankenversicherung/elektronische-gesundheitskarte/allgemeine-informationen-egk.html>, 2015. [Online; accessed 30-April-2015].
- [37] **Bundesministerium für Finanzen. FinanzOnline - Login mit Bürgerkarte.** <https://finanzonline.bmf.gv.at/fon/>, 2015. [Online; accessed 30-April-2015].

# CHAPTER 6

## Appendix

## 6.1 APDU Table

Cl	INS	P1	P2	Data	Le	Original Command
00	A4	04	00	d0 40 00 00 17 01 01 01	FF	execSELECT-AID(AID-SV-PERSONENDATEN)
00	A4	04	00	d0 40 00 00 17 00 18 01	FF	execSELECT-AID(AID-INFOBOX)
00	A4	04	00	d0 40 00 00 17 00 10 01	FF	execSELECT-AID(AID-SV-SIG-CERT)
00	A4	04	00	d0 40 00 00 17 00 12 01	FF	execSELECT-AID(AID-DF-SS)
00	A4	04	00	d0 40 00 00 17 00 13 01	FF	execSELECT-AID(AID-DF-GS)
00	A4	02	04	ef 01	FF	execSELECT-FID(FID-GRUNDDATEN)
00	A4	02	04	ef 02	FF	execSELECT-FID(FID-EHIC)
00	A4	02	04	ef 03	FF	execSELECT-FID(FID-SV-PERSONENBINDUNG)
00	A4	02	04	ef 04	FF	execSELECT-FID(FID-STATUS)
00	A4	02	04	d0 40 00 00 17 00 13 01	FF	execSELECT-FID(AID-DF-GS)
00	A4	02	04	00 32	FF	execSELECT-FID(EF-VERSION)
00	A4	02	04	ef 01	FF	execSELECT-FID(EF-INFOBOX-LEGACY)
00	A4	02	04	c0 02	FF	execSELECT-FID(EF-INFOBOX)
00	A4	02	04	2f 01	FF	execSELECT-FID(EF-SV-SIG-CERT-CA)
00	A4	02	04	2f 02	FF	execSELECT-FID(EF-SV-SIG-CERT)
00	A4	02	04	c0 00	FF	execSELECT-FID(EF-C-X509-CH-DS)
00	A4	02	04	c6 08	FF	execSELECT-FID(EF-C-X509-CA-CS-DS)
00	A4	02	04	2f 01	FF	execSELECT-FID(EF-C-X509-CH-AUT)
00	A4	02	04	2f 02	FF	execSELECT-FID(EF-C-X509-CA-CS)
00	22	??	??	84 03 80 02 00 80 01 04	00	execMSE(ecdsa-sha1)
00	22	??	??	84 03 80 03 00 80 01 02	00	execMSE(rsa-sha1)
00	22	??	??	84 03 80 03 00 80 01 02	00	execMSE(rsa-sha256)
00	22	??	??	84 03 80 02 00 80 01 04	00	execMSE(ecdsa-sha256)
00	22	??	??	84 03 80 02 00 80 01 04	00	execMSE(ecdsa-ripemd160)
00	20	00	81	?? ?? ?? ?? ?? ?? ?? ??	00	validatePIN*(?? = PIN)
00	A4	00	0C	3f 00	00	execSELECT-MF(MF)
00	2A	9E	9A	??	FF	PSO-ComputeDigitalSignature (?? = Hash of data to be signed)
...	*The "??" bytes need extra computation and <b>can not</b> be simply inserted					

Table 6.1: APDU commands list

## 6.2 Code

```
WebService(name="STALPortType", targetNamespace="http://www.egiz.gv.at/
wsdl/stal")
2 SOAPBinding(parameterStyle=SOAPBinding.ParameterStyle.BARE)
XmlSeeAlso({ObjectFactory.class})
4 public abstract interface STALPortType {
    WebMethod
6 WebResult(name="GetNextRequestResponse", targetNamespace="http://www.
egiz.gv.at/stal", partName="part1")
    public abstract GetNextRequestResponseType connect(WebParam(name="
SessionId", targetNamespace="http://www.egiz.gv.at/stal", partName="
part1") String paramString);
8
    WebMethod(operationName="nextRequest")
10 WebResult(name="GetNextRequestResponse", targetNamespace="http://www.
egiz.gv.at/stal", partName="part1")
    public abstract GetNextRequestResponseType getNextRequest(WebParam(name=
"GetNextRequest", targetNamespace="http://www.egiz.gv.at/stal",
partName="part1") GetNextRequestType paramGetNextRequestType);
12
    WebMethod
14 WebResult(name="GetHashDataInputResponse", targetNamespace="http://www.
egiz.gv.at/stal", partName="part1")
    public abstract GetHashDataInputResponseType getHashDataInput(WebParam(
name="GetHashDataInput", targetNamespace="http://www.egiz.gv.at/stal",
partName="part1") GetHashDataInputType paramGetHashDataInputType)
16 throws GetHashDataInputFault;
}
```

Listing 6.1: STALPort class, maps XML files from the server according to their request

```
1 PROCESS READ INFOBOX REQUEST*
3 INFORMATION: Received 1 requests: class at.gv.egiz.stal.service.types.
    InfoboxReadRequestType
Maer 19, 2015 5:57:41 PM at.gv.egiz.bku.smcststal.AbstractSMCCSTAL
    handleRequest
5
.....
7
SEND READ INFOBOX RESPONSE*
9
Maer 19, 2015 5:57:41 PM at.gv.egiz.bku.online.applet.AppletBKUWorker run
11 INFORMATION: Sending 1 responses: class at.gv.egiz.stal.service.types.
    InfoboxReadResponseType
Maer 19, 2015 5:57:41 PM at.gv.egiz.bku.online.applet.AppletBKUWorker run
13
*afterwards inserted markers
```

Listing 6.2: Client-server communication log

```

[01] <?xml version="1.0" encoding="UTF-8"?>
2 [02] <sl:CreateXMLSignatureRequest
[03]   xmlns:sl="http://www.buergerkarte.at/namespaces/securitylayer/1.2#"
>
4 [04]   <sl:KeyboxIdentifier>SecureSignatureKeypair</sl:KeyboxIdentifier>
[05]   <sl:DataObjectInfo Structure="enveloping">
6 [06]     <sl:DataObject>
[07]       <sl:XMLContent>Ich bin ein einfacher Text.</sl:XMLContent>
8 [08]     </sl:DataObject>
[09]     <sl:TransformsInfo>
10 [10]       <sl:FinalDataMetaInfo>
[11]         <sl:MimeType>text/plain</sl:MimeType>
12 [12]       </sl:FinalDataMetaInfo>
[13]     </sl:TransformsInfo>
14 [14]   </sl:DataObjectInfo>
[15] </sl:CreateXMLSignatureRequest>

```

Listing 6.3: XML server request

```

1 public class InternalSSLSocketFactory
  extends SSLSocketFactory
3 {
  ..
5 public boolean isEgovAgency ()
  {
7   this.log.info("Checking if server is egov agency.");
   if (this.sslSocket != null) {
9     try
      {
11      X509Certificate localX509Certificate = (X509Certificate)this.
        sslSocket.getSession().getPeerCertificates()[0];

13      this.log.info("Server cert: {}.", localX509Certificate);
        return isGovAgency(localX509Certificate);
15      }
      catch (SSLPeerUnverifiedException localSSLPeerUnverifiedException)
17      {
        this.log.error("Failed to check server cert.",
19      localSSLPeerUnverifiedException);
        return false;
      }
21    }
    this.log.info("Not a SSL connection.");
23    return false;
  }
25 ...

```

Listing 6.4: SSLSocket used to establish a connection

```

1 public class Launcher implements BKUControllerInterface {

```

```

3  static{
    localObject1 = new URL("http://localhost:" + Integer.getInteger("mocca
    .http.port", 3495).intValue() + '/');
5  ...
    localObject4 = new URL((URL) localObject1, "/GetCertificate");
7  ...
    GETCERTIFICATE_URL = (URL) localObject4;
9  ...
    }
11 ...
    }

13

15
17 public class GetCertificateInvoker implements Runnable{
19     public void run(){
21         ...
        localURLConnection = (URLConnection) Launcher.
        GETCERTIFICATE_URL.openConnection();
23
        localURLConnection.setRequestMethod("GET");
        localURLConnection.setReadTimeout(0);
        localURLConnection.connect();
25         ...
    }
}

```

Listing 6.5: Code snipped of the REST services

```

from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnectionObserver import
    ConsoleCardConnectionObserver
from smartcard.Exceptions import CardRequestTimeoutException
from smartcard.util import HexListToBinString, BinStringToHexList
# request any card type
cardtype = AnyCardType()
# define the apdus used in this script
selectMF = [0x00, 0xA4, 0x00, 0x0C]
selectAID_SID_DF_GS = [0x00, 0xA4, 0x04, 0x00, 0x08, 0xD0, 0x40, 0
    x00, 0x00, 0x17, 0x00, 0x13, 0x01, 0x00]
selectFID_EF_C_X509_CH_AUT = [0x00, 0xA4, 0x02, 0x04, 0x02, 0x2F,
    0x01, 0x00]
# define global vars
global apdu
global transFileCounter
global sw1
transFileCounter = 0
sw1 = 144
# methods for setting the right apdu
def masterfile():

```

```

    global apdu
    apdu = selectMF
def certApplication():
    global apdu
    apdu = selectAID_SID_DF_GS
def certElementaryFile():
    global apdu
    apdu = selectFID_EF_C_X509_CH_AUT
def other():
    global apdu
    global transFileCounter
    m = 0x00+transFileCounter
    transFileCounter+=1
    apdu = [0x00, 0xB0, m, 0x00, 0x00]
selectPriorData = {1 : masterfile,
                    2:certApplication,
                    3:certElementaryFile,
}
try:
    global apdu
    binstring = None
    realstring = None
    # request card insertion
    print 'insert a card, if not timeout within 20s'
    cardrequest = CardRequest(timeout=20, cardType=cardtype)
    cardservice = cardrequest.waitforcard()
    # add card insertion observer & connect
    observer = ConsoleCardConnectionObserver()
    cardservice.connection.addObserver(observer)
    cardservice.connection.connect()
    i=1
    m=0
    # while card status = 0x90 0x00 (Successfully executed)
    execute apdus
    while sw1==144:
        print '_____',
if i<4:
    #execute prior selects on EFs/AIDs
    print i
    selectPriorData[i]()
    print apdu
    response, sw1, sw2 = cardservice.connection.transmit(apdu)
else:
    #ReadTransparentFileSection, reads the signature data
    other()
    print 'Read Transparent File - Part' + str(transFileCounter)
    response, sw1, sw2 = cardservice.connection.transmit(apdu)
    if i==4:
        binstring = response

```



```

        realstring = HexListToBinString( response )
    else:
        binstring += response
        realstring += HexListToBinString( response )
if sw1 == 107:
    print '----->Reached the end of the File!'
elif sw1== 144:
    print '----->Command successfully executed!'
else:
    print '----->That should not have happened, seems like the
    script is broken!'
i+=1
secureSignatureByte=open('./secureSignatureByte', 'w+')
secureSignatureString=open('./secureSignatureString', 'w+')
secureSignatureCert=open('./secureSignatureCert.cer', 'w+')
print '_____ '
print '_____ '
print >> secureSignatureByte, binstring
print >> secureSignatureString, realstring
print >> secureSignatureCert, realstring
print '_____ '
print 'secureSignature certificate successfully read!'
print 'Bytes saved to file secureSignatureByte in the current
directory'
print 'Bytes saved to file secureSignatureString in the
current directory'
print ''
except CardRequestTimeoutException:
    print 'time-out: no card inserted during last 20s'

```

Listing 6.6: Read secure signature certificate data - python script

```

from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnectionObserver import
    ConsoleCardConnectionObserver
from smartcard.Exceptions import CardRequestTimeoutException
from smartcard.util import HexListToBinString, BinStringToHexList
# request any card type
cardtype = AnyCardType()
# define the apdus used in this script
selectMF = [0x00, 0xA4, 0x00, 0x0C]
selectAID_SID_DF_SS = [0x00, 0xA4, 0x04, 0x00, 0x08, 0xD0, 0x40, 0
    x00, 0x00, 0x17, 0x00, 0x12, 0x01, 0x00]
selectFID_EF_C_X509_CH_DS = [0x00, 0xA4, 0x02, 0x04, 0x02, 0xC0, 0
    x00, 0x00]
# define global vars
global apdu

```

```

global transFileCounter
global sw1
transFileCounter = 0
sw1 = 144
#methods for setting the right apdu
def masterfile():
    global apdu
    apdu = selectMF
def certApplication():
    global apdu
    apdu = selectAID_SID_DF_SS
def certElementaryFile():
    global apdu
    apdu = selectFID_EF_C_X509_CH_DS
def other():
    global apdu
    global transFileCounter
    m = 0x00+transFileCounter
    transFileCounter+=1
    apdu = [0x00, 0xB0, m, 0x00, 0x00]
selectPriorData = {1 : masterfile,
                    2:certApplication,
                    3:certElementaryFile,
}
try:
    global apdu
    binstring = None
    realstring = None
    # request card insertion
    print 'insert a card, if not timeout within 20s'
    cardrequest = CardRequest(timeout=20, cardType=cardtype)
    cardservice = cardrequest.waitforcard()
    # add card insertion observer & connect
    observer = ConsoleCardConnectionObserver()
    cardservice.connection.addObserver(observer)
    cardservice.connection.connect()
    i=1
    m=0
    # while card status = 0x90 0x00 (Successfully executed)
    execute apdus
    while sw1==144:
        print '_____',
if i<4:
    #execute prior selects on EFs/AIDs
    print i
    selectPriorData[i]()
    print apdu
    response, sw1, sw2 = cardservice.connection.transmit(apdu)
else:

```

```

#ReadTransparentFileSection, reads the signature data
other()
print 'Read Transparent File - Part' + str(transFileCounter)
response, sw1, sw2 = cardservice.connection.transmit(apdu)
if i==4:
    binstring = response
    realstring = HexListToBinString( response )
else:
    binstring += response
    realstring += HexListToBinString( response )
if sw1 == 107:
    print '----->Reached the end of the File!'
elif sw1== 144:
    print '----->Command successfully executed!'
else:
    print '----->That should not have happened, seems like the
    script is broken!'
i+=1
QualifiedSignatureByte=open('./QualifiedSignatureByte', 'w+')
QualifiedSignatureString=open('./QualifiedSignatureString', 'w
+')
QualifiedSignatureCert=open('./QualifiedSignatureCert.cer', 'w
+')
print '_____
print '_____
print >> QualifiedSignatureByte, binstring
print >> QualifiedSignatureString, realstring
print >> QualifiedSignatureCert, realstring
print '_____
print 'Qualified certificate successfully read!'
print 'Bytes saved to file QualifiedSignatureByte in the
current directory'
print 'Bytes saved to file QualifiedSignatureString in the
current directory'
print ''
except CardRequestTimeoutException:
    print 'time-out: no card inserted during last 20s'

```

Listing 6.7: Read qualified certificate data - python script

```

from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnectionObserver import
    ConsoleCardConnectionObserver
from smartcard.Exceptions import CardRequestTimeoutException
from smartcard.util import HexListToBinString, BinStringToHexList
# request any card type
cardtype = AnyCardType()

```

```

# define the apdus used in this script
selectMF = [0x00, 0xA4, 0x00, 0x0C]
selectFID_EF_VERSION = [0x00, 0xA4, 0x02, 0x04, 0x02, 0x00, 0x32,
0xFF]
selectAID_AID_INFOBOX = [0x00, 0xA4, 0x04, 0x00, 0x08, 0xD0, 0x40,
0x00, 0x00, 0x17, 0x00, 0x18, 0x01, 0xFF]
selectFID_EF_INFOBOX = [0x00, 0xA4, 0x02, 0x04, 0x02, 0xC0, 0x02, 0
xFF]
# define global vars
global apdu
global transFileCounter
global sw1
transFileCounter =0
sw1 =144
#methods for setting the right apdu
def masterfile():
    global apdu
    apdu = selectMF
def versionElementaryFile():
    global apdu
    apdu = selectFID_EF_VERSION
def infoboxApplication():
    global apdu
    apdu = selectAID_AID_INFOBOX
def infoboxElementaryFile():
    global apdu
    apdu = selectFID_EF_INFOBOX
def other():
    global apdu
    global transFileCounter
    m = 0x00+transFileCounter
    transFileCounter+=1
    apdu = [0x00, 0xB0, m, 0x00, 0x00]
selectPriorData = {1 : masterfile,
2:versionElementaryFile,
3:infoboxApplication,
4:infoboxElementaryFile,
}
try:
    global apdu
    binstring = None
    realstring = None
    # request card insertion
    print 'insert a card, if not timeout within 20s'
    cardrequest = CardRequest(timeout=20, cardType=cardtype)
    cardservice = cardrequest.waitforcard()
    # add card insertion observer & connect
    observer = ConsoleCardConnectionObserver()
    cardservice.connection.addObserver(observer)

```

```

cardservice.connection.connect()
i=1
m=0
# while card status = 0x90 0x00 (Successfully executed)
execute apdu
while sw1==144:
    print '_____','
if i<5:
    #execute prior selects on EFs/AIDs
    print i
    selectPriorData[i]()
    print apdu
    response, sw1, sw2 = cardservice.connection.transmit(apdu)
else:
    #ReadTransparentFileSection, reads the personal link data
    other()
    print 'Read Transparent File - Part' + str(transFileCounter)
    response, sw1, sw2 = cardservice.connection.transmit(apdu)
    if i==5:
        binstring = response
        realstring = HexListToBinString( response )
    else:
        binstring += response
        realstring += HexListToBinString( response )
if sw1 == 107:
    print '----->Reached the end of the File!'
elif sw1== 144:
    print '----->Command successfully executed!'
else:
    print '----->That should not have happened, seems like the
    script is broken! Remember that this script is designed for E-
    Card Generation 4!'
i+=1
PersonalLinkByte=open('./PersonalLinkByte', 'w+')
PersonalLinkString=open('./PersonalLinkString', 'w+')
print '_____'
print '_____'
print >> PersonalLinkByte, binstring
print >> PersonalLinkString, realstring
print '_____'
print 'Personal Link successfully read!'
print 'Bytes saved to file PersonalLinkByte in the current
directory'
print 'Bytes saved to file PersonalLinkString in the current
directory'
print ''
except CardRequestTimeoutException:
    print 'time-out: no card inserted during last 20s'

```

Listing 6.8: Read personal link data - python script

```
import binascii
from smartcard.CardType import AnyCardType
from smartcard.CardRequest import CardRequest
from smartcard.CardConnectionObserver import
    ConsoleCardConnectionObserver
from smartcard.Exceptions import CardRequestTimeoutException
from smartcard.util import HexListToBinString, BinStringToHexList
# request any card type
cardtype = AnyCardType()
# define the apdus used in this script
selectMF = [0x00, 0xA4, 0x00, 0x0C]
selectAID_AID_DF_SS = [0x00, 0xA4, 0x04, 0x00, 0x08, 0xD0, 0x40, 0
    x00, 0x00, 0x17, 0x00, 0x12, 0x01, 0x00]
selectExecMSE1 = [0x00, 0x22, 0x41, 0xB6, 0x08, 0x84, 0x03, 0x80,
    0x02, 0x00, 0x80, 0x01, 0x04]
selectExecMSE2 = [0x00, 0x22, 0x41, 0xAA, 0x03, 0x80, 0x01, 0x40]
# define global vars
global execCreateSignature
execCreateSignature = []
global verifyPINAPDU
verifyPINAPDU = []
#build create certificate APDU from given hash input
def buildCreateSignatureApu(string):
    global execCreateSignature
    #initialise CLA, INS, P1, P2 and Lc
    frame = bytearray()
    frame.append(0x00)
    frame.append(0x2A)
    frame.append(0x9E)
    frame.append(0x9A)
    frame.append(len(string))

    #append data bytes
    b = bytearray()
    b.extend(string)
    b.append(0x00)
    execCreateSignature = frame + b
#build verify PIN from given PIN input
def createVerifyPINAPDU(string):
    global verifyPINAPDU
    #initialise 2 bytearrays with pin data
    b2 = bytearray(7)
    b3 = bytearray(7)
    counter =0
    j =0
```

```

for x in string:
    if counter % 2 ==0:
        y=0
    else :
        y=1
    tmp1 =j
    if y!= 0:
        b2[tmp1] = b2[tmp1] | int(x)
    else:
        b2[tmp1] = b2[tmp1] | int(x)<<4

    tmp2 = j
    if y!= 0:
        b3[tmp2] = b3[tmp2] | 15
    else:
        b3[tmp2] = b3[tmp2] | 240

    j += counter%2
    counter += 1
#append verify APDU Frame (CLA, INS, P1, P2, LC and FF as
placeholder for PIN data
locapud = bytearray()
locapud.append(0x00)
locapud.append(0x20)
locapud.append(0x00)
locapud.append(0x81)
locapud.append(0x08)
locapud.append(0x20)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
locapud.append(0xFF)
#Insert PIN arrays into APDU frame
i=0
while i<7:
    tmp15_14 = 6+i
    locapud[tmp15_14] = (locapud[tmp15_14] & (b3[i] ^ 0xFFFFFFFF))
    tmp31_30 = 6+i
    locapud[tmp31_30] = (locapud[tmp31_30] | b2[i])
    i+=1
#Insert PIN length into APDU frame
i = 0xFF & 6
j = ((1<<4)-1)
k = 16 - 4 - 4 %8
param = 4/8+5
i = i<<k

```

```

j = j<<k
tmp57_55 = param
locapud[tmp57_55] = ((locapud[tmp57_55] & 0xFF & (j ^ 0xFFFFFFFF
) >> 8))
tmp76_74 = param
locapud[tmp76_74] = ((locapud[tmp76_74] | 0xFF & i >> 8))
tmp95_94 = (param + 1)
locapud[tmp95_94] = ((locapud[tmp95_94] & 0xFF & (j ^ 0xFFFFFFFF
)))
tmp113_112 = (param + 1)
locapud[tmp113_112] = ((locapud[tmp113_112] | 0xFF & i))
verifyPINAPDU = locapud

selectAPDU = {1 : selectMF,
              2:selectAID_AID_DF_SS,
              3:list(verifyPINAPDU),
              4:selectExecMSE1,
              5:selectExecMSE2,
              6:execCreateSignature,
              }
CommandsString = {1 : "    select master file ",
                  2:"    select certificate ",
                  3:"        verify pin        ",
                  4:"        execute MSE        ",
                  5:"        execute MSE        ",
                  6:"    create signature hash",
                  }
}

try:
    global apdu
    binstring = None
    realstring = None
    hashvar = None
    pinvar= None
    # request card insertion
    print 'insert a card, if not timeout within 20s'
    cardrequest = CardRequest(timeout=20, cardType=cardtype)
    cardservice = cardrequest.waitforcard()
    # add card insertion observer & connect
    observer = ConsoleCardConnectionObserver()
    cardservice.connection.addObserver(observer)
    cardservice.connection.connect()
    #perform user dialogue for input data
    print " _____"
    print "|    Welcome to the pyscard signature Hash creator!    |"
    print "|-----|
    print "|    card entered, proceed with parameter input:    |"
    print "|-----|

```



```

hashvar = raw_input("|*****      Enter the hash value to sign
:      *****|\n|      (max 256 chars)
      |\n")
print "|-----|"
print "|*****      You entered the following Hash:      *****|\n|
n|      |"
print "      "+hashvar+ "\n|
      |"

print "|-----|"
pinvar = raw_input("|Enter the signaturePIN of the inserted
citizen card:|\n|      (max 6 numbers, 10 retries)
      |\n")
print "|-----|"
print "|*****      You entered the following PIN:      *****|\n|
n|      |"
print "      "+pinvar + "\n|
      |"

print "|
n|-----|\
print "|-----|\
n|*****|"
print "|*****      Your request is being processed...      *****|"
print "|*****|"
n|-----|"
print "|-----|\
n|_____|"

#build both APDUs from input
buildCreateSignatureApdu(hashvar)
createVerifyPINAPDU(pinvar)
i=1
#process APDU communication
while i<7:
    print "
_____ "
    print "
|*****|"
    print "|****Processing your"+ CommandsString[i] +"
request***|"
    print "
|*****|"
    print "
|-----|"
    if i==3:
        #verify PIN
        response, sw1, sw2 = cardservice.connection.
transmit(list(verifyPINAPDU))
        if (sw1 == 105 and sw2 == 131):
            print "
|-----|"

```



```

        print " | _____ | "
except CardRequestTimeoutException:
    print 'time-out: no card inserted during last 20s'

```

Listing 6.9: Create signature hash - python script

```

PROCESS SIGN REQUEST
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.online.applet.
    AppletBKUWorker run
INFORMATION: Received 1 requests: class at.gv.egiz.stal.service.
    types.SignRequestType
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.smcstal.AbstractSMCCSTAL
    handleRequest
FEIN: Got request list containing 1 STAL requests.
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.smcstal.AbstractSMCCSTAL
    handleRequest
INFORMATION: Processing: class at.gv.egiz.stal.SignRequest.
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.smcstal.AbstractSMCCSTAL
    getResponse
INFORMATION: Retry #1 of 1.
***
***
***          Welcome to the IAIK JCE-ME Library
***
***
***          ***
*** This version of IAIK JCE-ME is licensed for educational and
    research ***
*** use and evaluation only. Commercial use of this software is
    pro- ***
*** hibited. For details please see http://jcewww.iaik.at/sales/
    licences/.***
*** This message does not appear in the registered commercial
    version. ***
***
***
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.smcstal.
    SignRequestHandler handleRequest
FEIN: Found signature method: http://www.w3.org/2001/04/xmldsig-
    more#ecdsa-sha256.

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.ExclSignatureCardProxy
    invoke
AM FEINSTEN: Invoking method createSignature() with exclusive
    access.

```

```

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 4 bytes, nc=0, ne=0
-> 00:a4:00:0c
Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 2 bytes, SW=9000 [14ms]
<- 90:00

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 14 bytes, nc=8, ne=256
-> 00:a4:04:00:08:d0:40:00:00:17:00:12:01:00
Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 24 bytes, SW=9000 [24ms]
<- 6f:14:84:08:d0:40:00:00:17:00:12:01:a5
    :08:53:02:02:00:54:02:01:02:90:00

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 4 bytes, nc=0, ne=0
-> 00:20:00:81
Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 2 bytes, SW=63ca [31ms]
<- 63:ca
Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.reader.PinpadCardReader
    verify
WARNUNG: Falling back to default pin-entry.

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.reader.DefaultCardReader
    verify
FEIN: VERIFY

Maer 19, 2015 5:57:42 PM at.gv.egiz.smcc.reader.DefaultCardReader
    dropExclusive
FEIN: Dropping exclusive card access

Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.BKUGUIImpl
    showSignaturePINDialog
FEIN: Scheduling signature-pin dialog.
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.pin.gui.SignPINProvider
    providePIN
AM FEINSTEN: [Thread-16] wait for action.

Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.BKUGUIImpl$6 run
FEIN: [AWT-EventQueue-2] Show signature-pin dialog.
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.HelpListener
    setHelpTopic

```

```

AM FEINSTEN: Setting help topic: help.signpin.
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Resizing ...
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Running required button panel renderer ...
Maer 19, 2015 5:57:42 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Resize done.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.HelpListener
    getHelpURL
FEIN: Return help url: https://www.prime-sign.com/mocca/help/de/help.signpin.html.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.online.applet.BKUApplet
    getFocusFromBrowser
FEIN: Obtained focus from browser.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Try setting focus to current component ...
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Component to obtain focus: PINField.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.online.applet.BKUApplet
    getFocusFromBrowser
FEIN: Obtained focus from browser.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Try setting focus to current component ...
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Component to obtain focus: PINField.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.online.applet.BKUApplet
    getFocusFromBrowser
FEIN: Obtained focus from browser.
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Try setting focus to current component ...
Maer 19, 2015 5:57:44 PM at.gv.egiz.bku.gui.BKUGUIImpl
    getFocusFromBrowser
FEIN: Component to obtain focus: PINField.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.pin.gui.
    AbstractPINProvider actionPerformed
FEIN: [AWT-EventQueue-2] action performed - sign
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.pin.gui.SignPINProvider
    providePIN
AM FEINSTEN: [Thread-16] received action sign.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl
    showMessageDialog
FEIN: Scheduling message dialog.

```

```

Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 13 bytes, nc=8, ne=0
-> 00:20:00:81:08:XX:XX:XX:XX:XX:XX:XX
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl$7 run
FEIN: [AWT-EventQueue-2] Show message dialog.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl$7 run
FEIN: ButtonKey: null.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.HelpListener
    setHelpTopic
AM FEINSTEN: Setting help topic: wait.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl$7 run
FEIN: No okListener configured.
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Resizing ...
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Running required button panel renderer ...
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.gui.BKUGUIImpl resize
FEIN: Resize done.
Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 2 bytes, SW=9000 [47ms]
<- 90:00

Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.reader.DefaultCardReader
    regainExclusive
FEIN: Trying to regain exclusive card access
Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 13 bytes, nc=8, ne=0
-> 00:22:41:b6:08:84:03:80:02:00:80:01:04
Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 2 bytes, SW=9000 [16ms]
<- 90:00

Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 8 bytes, nc=3, ne=0
-> 00:22:41:aa:03:80:01:40
Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 2 bytes, SW=9000 [19ms]
<- 90:00

Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: CommmandAPDU: 38 bytes, nc=32, ne=256
-> 00:2a:9e:9a:20:37:97:fd:10:90:08:8f:95:cb:d5:d8:27:e5:da:6c:5d:
    f8:7a:be:ae:12:0b:a5:74:e4:65:46:6c:0d:2d:6b:f0:00
Maer 19, 2015 5:57:52 PM at.gv.egiz.smcc.LogCardChannel transmit
AM FEINSTEN: ResponseAPDU: 66 bytes, SW=9000 [217ms]
<- 26:1c:a1:32:28:89:91:99:7c:16:5c:84:5d:0b:63:e4:0f:73:5c:e8:6a
    :3f:94:e8:8f:e9:9d:78:54:23:8b:db:dd:bc:cd:09:cb:76:0e:fb:da
    :33:f9:cf:3f:87:7a:03:38:e1:b3:4a:30:a0:ba:00:4d:21:9a
    :61:87:35:23:4d:90:00

```

```
SEND SIGN RESPONSE
Maer 19, 2015 5:57:52 PM at.gv.egiz.bku.online.applet.
    AppletBKUWorker run
INFORMATION: Sending 1 responses: class at.gv.egiz.stal.service.
    types.SignResponseType
```

Listing 6.10: Sign request and response log