UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Semjon Kravtšenko

The Estonian Mobile-ID Implementation on the SIM Card

Bachelor's Thesis (9 ECTS)

Supervisor: Arnis Paršovs, PhD

Tartu 2022

The Estonian Mobile-ID Implementation on the SIM Card

Abstract:
Mobile-ID is an eID solution that can be used for authentication and digital signing. It has been transaction for more than a decade, and it is popular in Estonia. As of May 2022, more than 251 000 Mobile-IDs are in use, and, on average, 10 million operations are completed each month [1]. The solution relies on a special Mobile-ID functionality built into a phone SIM card. There is publicly available documentation about Mobile-ID [2], but it mainly describes the communication between the e-services implementing Mobile-ID support and the Mobile-ID backend. Not much information is publicly available on how Mobile-ID functionality is implemented on the SIM card, how it interacts with the phone and how it communicates with the Mobile-ID backend. In this work, interactions between the SIM card and the phone are documented. Additionally, the communication between the SIM card and the Mobile-ID backend is described, as well as how Mobile-ID service SMS are used to perform Mobile-ID transaction.

Keywords:
Mobile-ID, SMS, SIM card, SIM Application Toolkit, Over-the-Air (OTA)

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

Eesti Mobiil-ID rakendus SIM-kaardil

Lühikokkuvõte:
Mobiil-ID on eID lahendus, mida saab kasutada autentimiseks ja digiallkirjastamiseks. See toimib juba ligi aastakümne ja on Eestis populaarne. 2022. aasta mai seisuga oli Mobiil-ID-l üle 251 000 kasutaja ning iga kuu tehakse keskmiselt 10 miljonit toimingut [1]. Lahendus tugineb telefonis oleva SIM-kaardisse sisseehitatud spetsiaalsele Mobiil-ID funktsionaalsuele. Mobiil-ID kohta on olemas avalikult kättesaadav dokumentatsioon [2], kuid see kirjeldab peamiselt infovahetust Mobiil-ID tuge pakkuva e-teenuse ja Mobiil-ID tagasüsteemi vahel. Avalik teabe sellest, kuidas Mobiil-ID funktsionaalsus on rakedatud SIM-kaardis ning ka sellest, kuidas toimib selle interaktsioon mobiiltelefoniga ja infovahetus Mobiil-ID tagasüsteemiga, on ebapiisav. Käesolevas töös on dokumenteeritud mobiiltelefoni ja SIM-kaardi vahelised interaktsioonid. Lisaks sellele, SIM-kaardi ja Mobiil-ID tagasüsteemi vaheline side ja infovahetus on kirjeldatud ning ka see, kuidas Mobile-ID teenindus SMS-sõnumeid kasutatakse Mobiil-ID toimingute teostamiseks.

Võtmesõnad:
Mobiil-ID, SMS, SIM-kaart, SIM Application Toolkit, Over-the-Air (OTA)

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Estonia is known for its digital solutions that allow its residents to do their banking, access governmental services and even vote, all online. One of these solutions is Mobile-ID. Just like the ID-card, Mobile-ID allows authenticating to various e-services and also to sign files digitally. Some people find that using Mobile-ID is more convenient than using the ID card, since it can be used without additional hardware (a smart card reader). The only hardware required for using Mobile-ID is the user's phone, and a Mobile-ID-enabled SIM card inserted into it.

1.1 Mobile-ID transactions

Mobile-ID authentication works similarly to Mobile-ID digital signing. To perform both of these actions, a so-called Mobile-ID transaction must be performed. To perform a Mobile-ID transaction, an e-service must deliver to the user's phone a value that must be cryptographically signed inside the user's SIM card. The user's SIM card has a specialized Mobile-ID application (applet), as well as the Mobile-ID holder's private keys. After the signature is created, it must be transmitted from the SIM card back to the e-service. Hence, completing a Mobile-ID transaction is a complicated process involving several parties.

To start a Mobile-ID transaction, a Relying Party transmits the necessary information about the transaction to the provider of the Mobile-ID service (also known as the Application Provider) — SK ID Solutions AS. The Application Provider ensures that a special service SMS message arrives to the user's phone. This message contains the hash to be signed, the text that should be displayed to the user, and other information about the transaction. When the phone receives a service SMS message (class 2 SMS message), it does not show it to the user. Rather, the phone relays this incoming Mobile-ID SMS to the SIM card.

The communication between the SIM card and the phone follows an international GSM standard [3]. This standard, SIM Application Toolkit (STK), defines numerous commands that can be exchanged between a

phone and a SIM card. In the case of Mobile-ID, a command is used by the phone to deliver the Mobile-ID service SMS message to the SIM card. This command is issued by the phone to relay the service SMS message during the Mobile-ID transaction. The standard also defines proactive commands that the SIM card may issue to the phone in order to interact with the user (i.e., showing the text prompts or obtaining user input). STK also defines a command that allows SIM card to send class 2 SMS messages to the cellular network.

When the phone submits the service SMS with the information about the Mobile-ID transaction to the SIM card, the applet on the SIM card starts issuing STK commands for interacting with the user. In this interaction, the user should confirm the transaction by entering their Mobile-ID PIN code. After the transaction is confirmed, the SIM card creates a cryptographic signature over the hash received in the service SMS. Then, it creates a new, outgoing service SMS that contains the created cryptographic signature, and issues an STK command to the phone, to send this message to the cellular network. When the signature arrives to the Application Provider, it forwards it to the Relying Party, thus completing the transaction.

1.2 The Mobile-ID functionality provided by SIM card

The Mobile-ID-enabled SIM card provides two user-facing capabilities.

The first of them is confirming Mobile-ID transactions. When a transaction is performed, the SIM card ensures that a message (Mobile-ID prompt) with the transaction information appears on the phone's screen. When this message is displayed, the user may either accept or reject it. If the user accepts this message, they should enter their Mobile-ID PIN code. There are two Mobile-ID PIN codes[1]: PIN1 must be entered to confirm authentication transaction, and PIN2 must be entered to confirm digital signing. If a PIN code is entered incorrectly three consecutive times, the code will become blocked — i.e., it can not be used unless it is unblocked.

[1]Mobile-ID PIN codes (as well as the PUK code) are not the same codes as PIN and PUK codes of the SIM card

Additionally, the Mobile-ID-enabled SIM card provides the capability to use the SIM card menu. The user may unblock Mobile-ID PIN codes by entering their Mobile-ID PUK code in this menu. Additionally, the SIM card menu allows the user to change their Mobile-ID PIN codes and the Mobile-ID PUK code.

The SIM card (also referred to as UICC) is actually a smart card. The communication between the phone (also called the "terminal") and SIM card follows the T0 protocol for smart cards. As will be shown in Chapter 2 of this work, this means that general-purpose smart card readers are capable of enabling communication between a computer and the SIM card. The instructions sent between the SIM card and the phone can be analyzed (see Chapter 4).

1.3 Scope and contributions

This work describes the Mobile-ID implementation on the SIM card. It describes the STK communication that underlies the exchange of Mobile-ID service SMS between the phone and the SIM card, and Mobile-ID related UI activities invoked by the phone. Furthermore, it documents the format and handling of class 2 SMS messages that are used in Mobile-ID transactions. The work describes the parties that are involved in completing a Mobile-ID transaction and what are their roles. As an additional contribution, we discuss the security implications of some design weaknesses of Mobile-ID discovered in this research.

The main contribution of this research are:

1. Firstly, we believe that any technological solution that society relies on must be as transparent as possible. This research contributes to the transparency of the Mobile-ID solution.
2. Secondly, this research is beneficial because the research of a security-critical solution sometimes leads to the discovery of security vulnerabilities and design flaws that affect the security of a solution. Some design flaws were observed and documented in this work.
3. Thirdly, the results of this research may be used to aid in developing alternative Mobile-ID implementations.

1.4 Structure of the thesis

The thesis is structured as follows. Chapter 2 describes the equipment that was used for this research. Chapter 3 gives a high-level overview of the parties that perform Mobile-ID transactions and what their roles are. Chapter 4 describes Mobile-ID communication between the phone and the SIM card. It contains descriptions of commands exchanged between the phone and the SIM card and provides explanations for exchanged commands. It also shows the Mobile-ID functionality provided by the SIM card from the user's perspective. Chapter 5 analyzes the content and the handling of the Mobile-ID service SMS that are used to implement Mobile-ID transactions. Chapter 6 discusses several security implications of the Mobile-ID solution, that were found during this research.

2 Methods and tools used

Several crucial capabilities were needed to conduct this research. Firstly, to analyze and document the commands exchanged between the phone and the Mobile-ID-enabled SIM card, it was important to observe what those commands are. Secondly, to further research the peculiarities of the Mobile-ID functionality on the SIM card, it was essential to have the capabilities to modify the commands that are sent from the phone to the SIM card. Thirdly, to test how the network processes the data that is sent from the SIM card, it was necessary to modify the data that is sent from the phone to the network. The next three sections describe how these capabilities were obtained, while the last section describes the SIM cards and the phones used for this research.

2.1 Passively observing communication between the SIM card and the phone

A capability to observe the commands that are exchanged between the phone and the SIM card was required to analyze and document them. For passively observing these commands, a Simtrace2 [4] board (hardware) was used.

Simtrace2 acts as an intermediary (a machine-in-the-middle) device between the phone and the SIM card. To use the Simtrace2, one must first connect it to the phone and the SIM card. Additionally, the Simtrace2 must be connected to a computer. The Simtrace2 routes all the commands exchanged between the phone and the SIM card trough itself and sends copies of these commands to the computer.

See Figure 1 for an illustration of the physical setup. The connection to the SIM card is achieved by inserting the SIM card into the Simtrace2 slot for the SIM card. This slot is intended for inserting cards of standard SIM form factor. However, SIM cards of other form factors can be used by using an adapter. The Simtrace2 board is connected to the phone with a piece of a flexible printed circuit board (PCB). The board is connected to the computer via USB Type A interface.

Figure 1. Setup involving Simtrace2 for observing the communication between the phone and the SIM card

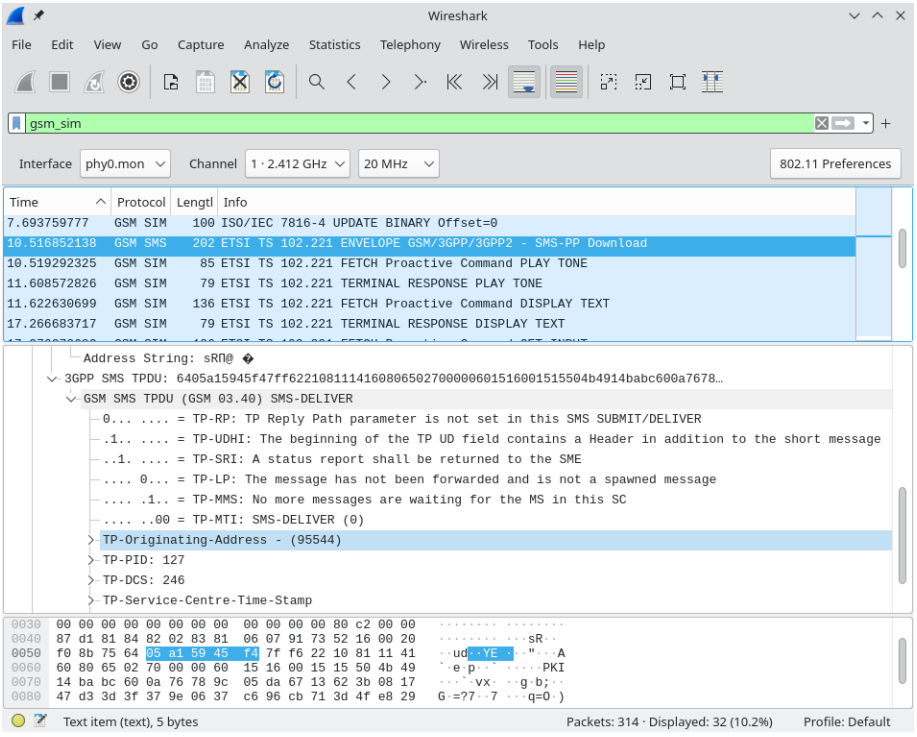


Figure 2. Dissected commands exchanged between the phone and the SIM card shown in Wireshark

For Simtrace2 to be operational, a firmware must be installed on it. Multiple firmwares may be installed on the Simtrace2 board, and they provide different functionality. For passively sniffing the communication, `simtrace-trace-dfu` firmware was used. A computer that is connected to the Simtrace2 can only accept the sniffed commands if it runs specialized software. Both the firmware and the application are a part of the Osmocom SIMtrace2 system [4].

For receiving commands that are sniffed by the Simtrace2 with the `simtrace-trace-dfu` firmware, `simtrace2-sniff` application is used. On the computer, sniffed communication was viewed using a network traffic analysis software Wireshark [5], that allows to partially dissect the commands used in the communication (see Figure 2). The dissections helped to analyze the exchanged commands.

2.2 Supplying service SMS messages to the SIM card

In order to further research, how the SIM card processes the incoming Mobile-ID SMS messages, another capability was required. Namely, the capability to issue STK commands with a modified incoming Mobile-ID service SMS messages to the SIM card[2].

To achieve this capability, a Python 3 script was developed (see Listing 2 in Appendix A). This script communicates with the SIM card, that is inserted into a smart card reader connected to the computer. Inserting SIM card into the smart card reader required using a special adapter. The script has the capability to issue STK commands with service SMS messages to the SIM card. It is also capable of responding to the proactive commands sent by the SIM card. This script implemented a subset of STK commands that may be sent during the Mobile-ID transaction, effectively emulating a phone by sending the commands that would be send by a phone. Therefore, the script allowed us to perform the STK communication with the SIM card.

[2]A capability to modify the commands that are exchanged between the phone and the SIM card would have also allowed us to research, how the SIM card processes the incoming Mobile-ID messages. However, to our knowledge, there are no tools that provide such capability. We decided that it is outside of scope for this research to develop such tool.

Only the commands that are required to perform a Mobile-ID transaction were implemented; the script does not provide the capability to simulate the phone in other interactions, e.g., using the SIM card menu.

2.3 Actively intercepting the service SMS messages

During the course of this research, it was observed that the SIM card reacts differently to the incoming Mobile-ID messages, depending on whether the same message was already supplied to it or not (see Section 5.3).

This is why it was not sufficient to use the script discussed in previous section to supply the service messages that were passively collected while performing a Mobile-ID transaction (as described in Section 2.1).

Another capability was required. Namely, this is the ability to capture incoming Mobile-ID message in a way, so that it does not reach the SIM card — the ability to intercept incoming Mobile-ID messages. We achieved this capability using two different methods[3].

The first discovered method involved using the Simtrace2 board. However, in this method, we used firmware and application different from what we used to passively observe the communication between the phone and the SIM card (Section 2.1). In this case, we used the `simtrace-trace-dfu` firmware and the `simtrace2-cardem-pcsc` application. Both the firmware and the application are developed within the Osmocom SIMtrace 2 project [4]. They are used similarly to what is described in Section 2.1 of this work, but these programs ensure that the phone communicates with the SIM card that is inserted into a smart card reader, rather than the SIM card inserted into the Simtrace2 board slot. An illustration of physical setup is provided in Figure 3. The script mentioned in the previous section can be used with this setup as well.

A modification to the `simtrace2-cardem-pcsc` application was required to ensure that the incoming Mobile-ID service SMS is not relayed from the phone to the SIM card. We modified to the application such that

[3]Primarily, the second method was used to conduct the research. Still, we document both methods, as it may help conduct other research.

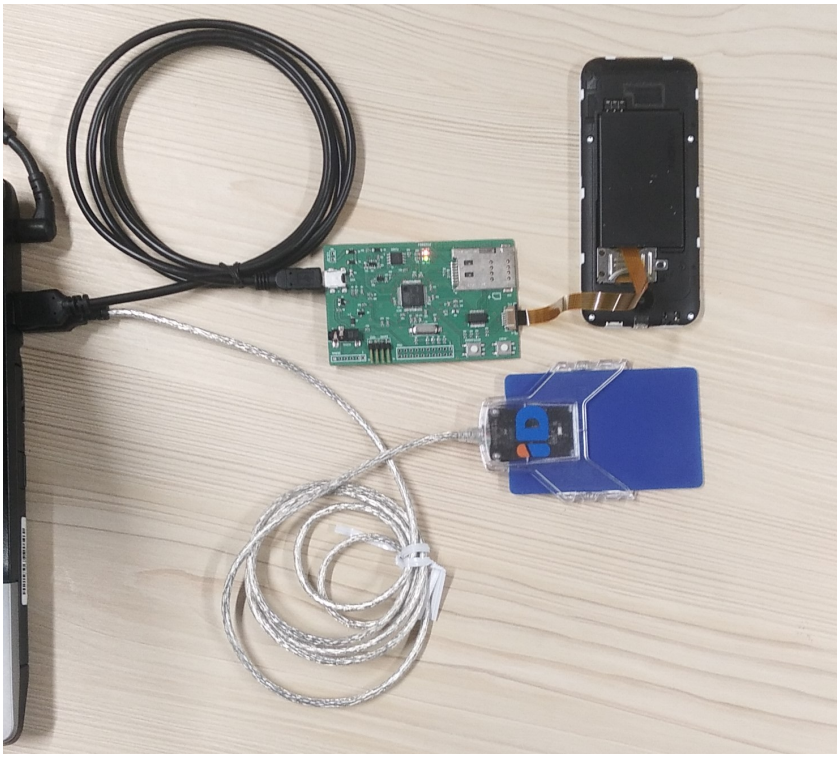


Figure 3. Setup involving Simtrace2 for intercepting service messages

it would not forward the commands sent by the phone to the SIM card, if they were longer than a set threshold, effectively blocking the commands containing incoming Mobile-ID SMS (see Listing 1 in Appendix A). The same modification also displayed the content of captured SMS message in hex format. The modifications were made by modifying the available source code and compiling the modified version.

We decided to use `simtrace-trace-dfu` and `simtrace2-cardem-pcsc` programs for the purpose of capturing incoming Mobile-ID SMS (rather than the programs introduced in the previous section), because developing the modification mentioned above was decided to be significantly easier than making a similar modification to the `simtrace-trace-dfu` firmware. The rationale for this is that `simtrace-trace-dfu` forwards the commands to the computer connected to the Simtrace2 board. This means that these commands can be filtered and displayed in a more high-level application.

The second discovered method for capturing the incoming Mobile-ID SMS involved using a phone with specialized firmware. This phone relays all the cellular communication that it receives and sends it to the connected computer. Moreover, it does not implement SIM Application Toolkit commands, and thus it does not transmit the incoming Mobile-ID messages to the connected SIM card. This phone and the firmware are discussed in the next section.

2.4 Modifying the data sent from the phone to the network

Modifying the data (the outgoing Mobile-ID SMS messages) that is sent from the phone to the cellular network is important to test how the Application Provider processes this data. We decided that changing the firmware of a phone to allow us to modify SMS messages before they are sent requires a tremendous effort that is not in scope of this work. Instead, we achieved a capability to send service SMS messages (to the cellular network) with arbitrary content and formatted as outgoing Mobile-ID SMS messages.

For this, `layer1` and `layer23` component of the OsmocomBB [6] software were used. OsmocomBB is an open source GSM Baseband software implementation: it implements some functionality of a mobile phone, including the capability to send text SMS messages. However, it does not implement the functionality of sending service messages. The `layer1` is a very small firmware that runs on a phone. This firmware can only run on a phone of a supported model; a Motorola C115 phone was used in this research. The phone must be connected to a computer. The `layer23` is software that runs on the computer and communicates with the phone. It implements the higher-level functions of a phone, e.g., sending SMS messages. To enable the functionality that allows sending SMS, the `layer23` must be compiled with a `-enable-transceiver` argument (flag).

For this work, we developed and used a modification for the `layer23` component of the OsmocomBB software that allows sending service messages to the network. A modification was made by changing the source code and building the modified version.

Figure 4. Multipurpose setup. It was used for intercepting and sending Mobile-ID messages using the Motorolla C115 phone. The setup also allows sending arbitrary commands to the SIM card (see Section 2.2).

During the research, it became apparent that many experiments involving sending outgoing Mobile-ID message to the network (for the purpose of completing a Mobile-ID transaction) required the setup to be sufficiently optimized, i.e., it is necessary that the setup allows performing the Mobile-ID transaction before the transaction timeout. By default, `layer23` component of the OsmocomBB software uses the SIM card that is inserted into the phone. However, the Python 3 script that communicated with the SIM card (discussed in the previous section), uses SIM card that is inserted in a smart card reader.

To eliminate the need to relocate the SIM card between the phone SIM card slot and the smart card reader for some of the experiments, the `demo_server` component from the softSIM project [7] was used. It allows using `layer23` application with the SIM card that is inserted in a

smart card reader[4].

The setup for sending outgoing Mobile-ID messages from the Motorolla C115 phone to the network is shown in Figure 4. In this setup, the phone is connected to the laptop with a special cable (UART cable). One end of the cable has a USB Type A form factor, and it is inserted into a computer. The other end has 2.5mm stereo headphone connector form factor and is inserted into the Motorola C115 phone.

To quickly modify the contents of the Mobile-ID SMS messages that are sent from the phone to the cellular network, another Python 3 script was developed (see Listing 3 in Appendix A). It allows to parse (deserialize) the content of the outgoing Mobile-ID SMS, to apply modifications to it, and to combine (serialize) the content back, so it is ready to be sent to the network in an outgoing Mobile-ID message.

2.5 SIM cards and phones used in the research

In Estonia, three cellular network providers issue Mobile-ID-enabled SIM cards: Telia, Elisa and Tele2 [8]. Three SIM cards were used to research how the incoming Mobile-ID messages are processed inside the SIM cards. The SIM card primarily used in this research was a Mobile-ID-enabled SIM card issued by the Elisa Eesti AS (Elisa EE) mobile network carrier in July 2020. Additionally, two other SIM cards were used briefly: these were Mobile-ID-enabled cards issued by Elisa EE and Tele2 EE.

In total, three phones were used to passively observe the communication between a phone and the SIM card: Motorolla C115 and Nokia 225 Dual SIM feature phones, and also Redmi 7A Android phone. All three of these phones were used with their original firmwares.

As mentioned earlier, Motorolla C115 phone was also used to capture data sent from the cellular network to the phone, and to send data to the cellular network. For this purpose, it was used with a `layer1` firmware (see the previous section).

[4]It communicates with the SIM card and exposes it to the `layer23` application via a special interface, called SIM Access Profile.

3 Overview of the Mobile-ID transaction process

This chapter shortly describes the process of Mobile-ID transaction, the participants of this process, and their roles. Before that, the chapter describes the origin of the Mobile-ID transaction scheme.

3.1 Origin of the transaction scheme

The Estonian Mobile-ID was released in 2007 [9]. The scheme for performing Mobile-ID transactions was at least partially developed by the Baltic WPKI Forum [10]. This forum was active between 2006 and 2008 and its work resulted in several agreements. The Application Provider for Mobile-ID, SK ID Solutions AS (at that time called AS Sertifitseerimiskeskus), was one of the participants of this forum [11]. This forum has published a document titled "WPKI mobile transactions: implementation recommendations" [12]. This document includes the term "Mobiil-ID" two times (see Figure 3). It is not clear how close modern-day Estonian Mobile-ID implements this specification, but we believe that this document contains an accurate description of Estonian Mobile-ID at its launch[5].

Some details about the parties performing Mobile-ID transactions can be inferred from another document called "Mobile-ID Short description" [13], by Remarc Systems OÜ, which describes a solution similar in many ways to Estonian Mobile-ID. Some additional information about the role of one of the transaction participants was obtained from a web page titled "OTA (Over The Air)" [14] (published by Thales).

3.2 Description of the transaction process

Mobile-ID transaction is supposed to be triggered when the End User interacts with the Relying Party, signaling that they intend to perform a transaction. Usually, it means clicking a button on an e-service provider's

[5]The document does not mention ECC, that is used in modern-day Mobile-ID. However, it is known that the ECC was not used in the Mobile-ID at its launch

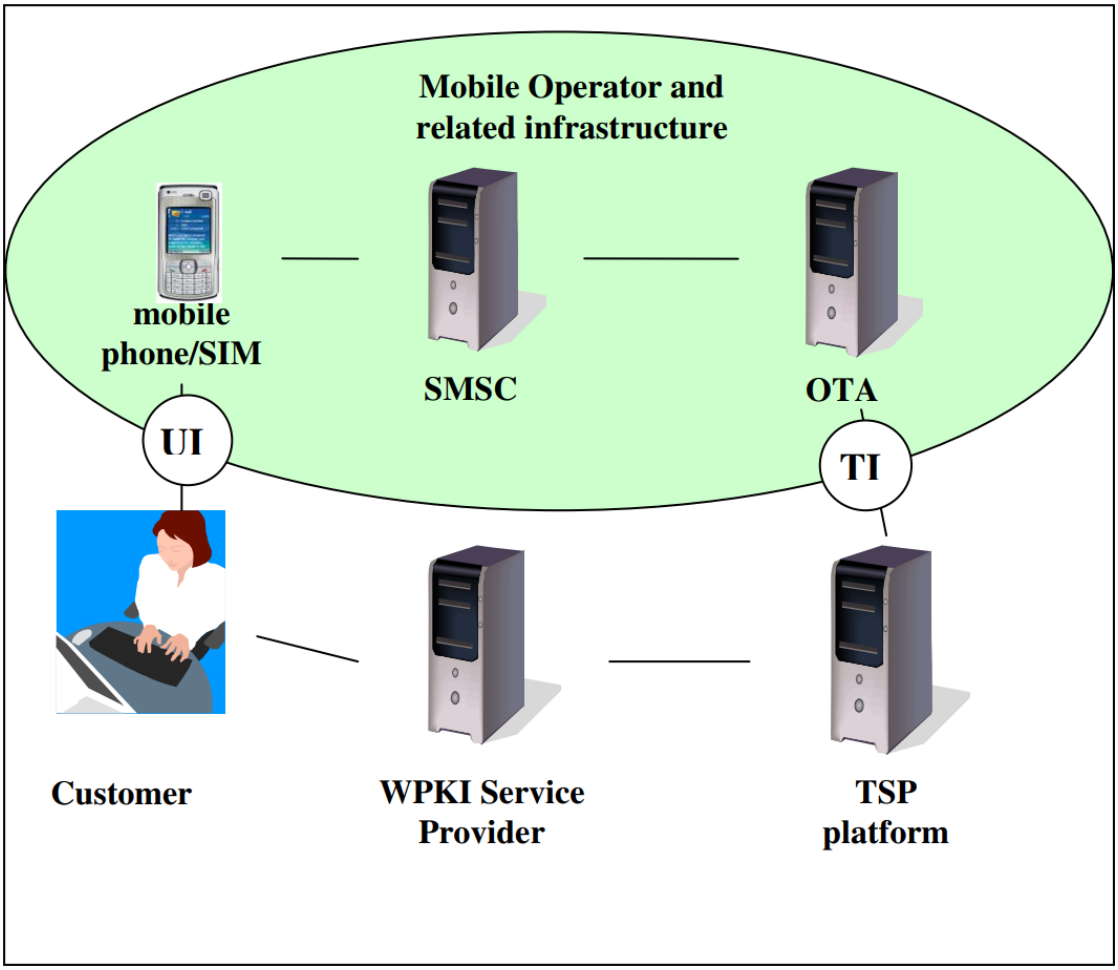


Figure 5. Participants of the Mobile-ID transaction [12, Figure 2]

website, or performing an action in a program maintained by the e-service provider.

To start the transaction, the Relying Party initiates a session with the Application Provider, and issues either an `authentication`, or a `signature` Mobile-ID REST API request[6] [2].

After receiving the API request, the Application Provider's server (Trusted Service Provider platform) forwards all the necessary information about the transaction to an "Over The Air" intermediary server (the OTA

[6]For digital signing, the Relying Party must know the user's certificate before the signature transaction is started. This is because the value that should be signed depends on the user's certificate. To obtain the user's certificate, Relying Party may issue a `certificate` Mobile-ID REST API request.

server). Cellular providers generally use OTA servers to send service SMS messages with the configuration updates to the subscribers' SIM cards, e.g., setting up preferred roaming networks. Each mobile network provider has its own OTA server, therefore the TSP platform should send the information to the "correct" OTA server — i.e., the OTA server of the End User's cellular network provider. Information about the communication channels and the protocol used to send this information has not been documented in publicly accessible sources.

After receiving the necessary information, the OTA server, in turn, creates a Mobile-ID SMS message for the End User's SIM card. This message is a Secured Packet, as defined in 3GPP TS 03 48 [15]. The exact content of this message is not known publicly. However, some information about it can be obtained by studying the related specifications, examining such messages, and using deduction based on the facts known about the message's contents (see Section 5.2 of this work).

After creating the Mobile-ID message, OTA server sends it to the cellular network. After being sent, this message (as any other SMS message) reaches the SMSC of the cellular provider. SMSC is an abbreviation for Short Message Service Center (also called SC) — it is a server whose purpose is to store, forward, convert and deliver SMS messages (Section 3.1 in [16]). After receiving the SMS message, the SMSC converts and sends it to the End User's phone in the same way it converts and sends other SMS messages. The process of sending and receiving SMS messages is shortly discussed in Section 5.1 of this work.

After the phone receives the Mobile-ID SMS message sent by the SMSC, it does not show it to the user, but forwards it to the SIM card (Section 7.1.1 in [3]). The Mobile-ID applet on the SIM card performs some checks to determine the validity of the incoming Mobile-ID SMS, and, depending on the result of the checks, either act upon the message, or discards it. As a rule, the applet acts upon a valid Mobile-ID SMS.

If the SIM card acts upon the message, it will start sending proactive STK commands to the phone, effectively interacting with the End User.

If the user agrees to perform the transaction by entering their Mobile-ID PIN code, the SIM card creates a cryptographic signature using elliptic curve digital signature algorithm (ECDSA). Then, if the signature is created successfully, the SIM card forms a class 2 SMS message with this signature, and issues an instruction to the phone, to send the message to the cellular network. The format of this SMS message is documented in Section 5.2 of this work.

After the SMSC receives this SMS message, it relays it to the OTA server, which, in turn, forwards the signature to the Trusted Service Provider platform.

Finally, if the signature is valid, the TSP platform forwards the signature to the Relying Party[7]. This completes the Mobile-ID transaction.

[7]The Relying Party is expected to confirm the validity of the signature, otherwise, it would needlessly rely on the Application Provider's signature validity check.

4 Communication between the phone and the SIM card

This chapter describes Mobile-ID-related communication between the SIM card and the phone. Where appropriate, it also shows Mobile-ID-related UI elements initiated by the SIM card.

4.1 SIM Application Toolkit

As mentioned earlier, the phone and the SIM card communicate using T0 protocol for smart cards. In this protocol, the terminal (phone) and the smart card (SIM card) exchange APDUs. The phone sends a command APDU (C-APDU) and the SIM card responds with response APDU (R-APDU). APDU stands for Application Protocol Data Unit — in other words, APDU is an instruction data object.

In the communication between a terminal and a smart card, it is the terminal that sends instructions to the smart card, by sending C-APDUs. The smart card executes commands received from the terminal and sends execution results as response APDUs (R-APDUs).

This, however, is not always sufficient. In some scenarios (e.g., using Mobile-ID), it is required that the SIM card may send certain instructions to the terminal. The SIM card may return an instruction to the terminal in the so-called "proactive command", e.g., to display a certain text on the phone's display.

4.1.1 General STK communication procedure

The proactive commands that the SIM card may send to the phone and defined in the SIM Application Toolkit (STK) standard[8] [3]. This standard also defines the communication procedure, that allows the terminal to retrieve the proactive commands and to report the execution results of proactive commands to the SIM card. The same standard defines how the phone should execute the proactive commands.

[8] In the course of this research, we found that the information about STK is scattered across multiple technical specifications. Some references between the standards are not clear, and some information is duplicated in multiple standards. Also, standards change their names during their development.

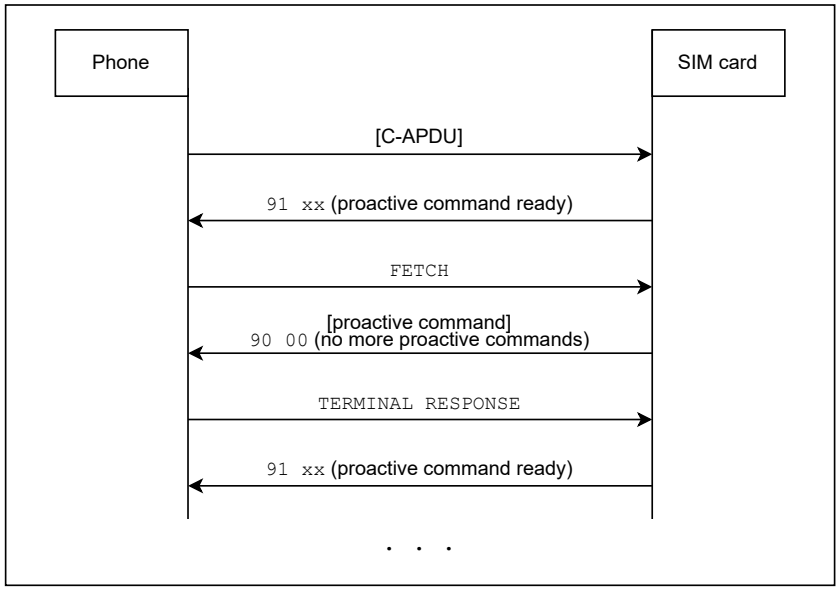


Figure 6. General structure of proactive SIM communication

The general STK communication procedure is presented in Figure 6. In the STK communication process:

1. The SIM card first waits until the terminal sends a C-APDU.
2. After executing the command, the SIM card specifies in the status word included in the R-APDU that a proactive command is available.
3. After the terminal receives a status word that signals the availability of a proactive command, the terminal *may* send a `FETCH` C-APDU to retrieve the proactive command.
4. The SIM card receives a `FETCH` command and returns the proactive command and SW `90 00` in the R-APDU.
5. The terminal receives and performs the proactive command.
6. After the terminal executes the proactive command, it sends a `TERMINAL RESPONSE` (Sections 5.2 and 6.8 in [3]) C-APDU to the SIM card. The `TERMINAL RESPONSE` is used to return the result of the proactive command execution to the SIM card.
7. The SIM card may return a SW indicating that another proactive command is available.

The SIM card can not directly issue commands to the phone, since it

would violate T0 protocol, that is used for the communication. The type of Command APDU is determined by the value of its Instruction Byte. The Instruction Byte values for Command APDU are defined in ETSI TS 102 221 (Table 10.5 in [17]).

The status word for communicating that a proactive command is available is `91 xx`, where `xx` is the length of the available proactive command. The terminal must know the length, as the T0 smart card communication protocol requires the length of the expected response to be specified in C-APDU. The SW for communicating that there is no proactive command available is `90 00`.

According to the STK standard, the SIM card may return to the phone only those proactive commands that are supported by the phone. The phone sends to the SIM card the information about what proactive commands it supports in the `TERMINAL PROFILE` C-APDU. This C-APDU is sent by the phone to the SIM card only once, after the phone boots up. The SIM card never sends proactive commands or SW starting with `91` before it receives the `TERMINAL PROFILE` APDU from the terminal.

Before the SIM card sends an instruction (e.g., to display text), it sometimes requires some additional information to be provided from the terminal beforehand. An `ENVELOPE` C-APDU (Sections 7.1 and 8 in [3]) may be used to transmit to the SIM card the incoming service SMS, the information that the user clicked on a SIM menu entry, etc.

During our research, all the tested phones (with their original firmwares) supported the technical specification defining STK, no interoperability issues were observed.

4.1.2 Proactive commands and data objects used by Mobile-ID

Proactive commands (Section 6.4 in [3]) used to implement Mobile-ID functionality are:

- `SET UP MENU`: specifies the first screen of the SIM card menu. This screen is displayed if the user opens the SIM card menu.
- `SELECT ITEM`: specifies the next screen of the menu, which should be opened after the user selects an item in a menu screen.

- `DISPLAY TEXT`: specifies the text that should be displayed to the user (necessary for displaying Mobile-ID prompt).
- `GET INPUT`: prescribes that the user should be asked for input and that this input should be transmitted back to the SIM card (used to retrieve Mobile-ID PIN codes).
- `PLAY TONE`: prescribes that the phone should generate and play a tone (to evoke the user's attention to the fact that a Mobile-ID prompt is being displayed on the screen).
- `SEND SHORT MESSAGE`: prescribes that the phone should send an SMS message included in this proactive command (used to send a Mobile-ID service message with a signature).

The C-APDUs `ENVELOPE` and `TERMINAL RESPONSE`, as well as the proactive commands, use Tag-Length-Value (TLV) data objects (Section 12 in [3]) in their structure. Each TLV object consists of 1 byte long tag, a length (occupying 1 or 2 bytes), and a value of the given length. Length from 0 to 127 bytes is encoded in one byte. Length from 128 to 255 is encoded in 2 bytes like this: `81 xx` — the `xx` here specifies the encoding of the length as an unsigned integer. Tag values for different TLV data types are given in Section 13 of ETSI TS 101 267 [3]. Every TLV object, according to the ETSI TS 101 267 is either BER-TLV[9], or SIMPLE-TLV. The value of every BER-TLV contains an array of concatenated SIMPLE-TLV objects.

For example, some of the SIMPLE-TLV objects are:

- Command details TLV (Section 12.6 in [3]) has tag value of `81` (or `01`). This object is included in a proactive command sent from the SIM card to the phone, and also in a `TERMINAL RESPONSE` to it. The value of this TLV has length of 3 bytes and contains information about the type of a proactive command and its qualifier.
- Device identities TLV (Section 12.7 in [3]) has tag value of `82` (or `02`), its length is 2 bytes and it contains information about the source and the destination of the command. E.g., the destination of

[9]BER refers to Basic Encoding Rules of ASN.1. BER-TLV use private class ASN.1 tags. This means that BER-TLV can be parsed with an ASN.1 decoder.

a DISPLAY TEXT proactive command is the display of the phone, and the destination of a SEND SHORT MESSAGE proactive command is the network.

- Result TLV (Section 12.12 in [3]) contains information about the result of the execution of a proactive command (for example: executed successfully). For some proactive commands, this object also contains additional information, e.g., the user-entered text, in case of GET INPUT proactive command. The tag value for Result TLV is 83 (or 03).
- Address TLV (Section 12.1 in [3]) contains the encoding of a dialing address, and then the digits of the telephone number (Section 10.5.1 in [18]), e.g., 37258353100. The tag of this TLV is 86 (or 06).
- SMS TPDU TLV (Section 12.13 in [3]) with tag value 8b (or 0b) is a wrapper for an SMS message. TPDU is short for Transfer Protocol Data Unit. The structure of SMS TPDUs are discussed in Section 5.1 of this work.

Examples of BER-TLV objects are:

- SMS-PP download TLV (Section 7.1.2 in [3]) has tag value of d1. The value of this object consists of three SIMPLE-TLV objects: Device identities, Address, and SMS TPDU TLV (the wrapper object). The Address contains the address of SMSC. This TLV can be transmitted to the SIM card with ENVELOPE C-APDU. SMS-PP download TLV is used to submit incoming Mobile-ID SMS to the SIM card.
- Proactive Command TLV. The SIM card sends this TLV as a response to FETCH C-APDU (as described above). The first two SIMPLE-TLV objects encoded in the value of Proactive Command are: Command Details and Device identities. The following encoded SIMPLE-TLV depend on the type of the proactive command. The tag of this BER-TLV is d0.

The STK communication related to the Mobile-ID interactions are discussed in the next section.

4.2 Observed communication

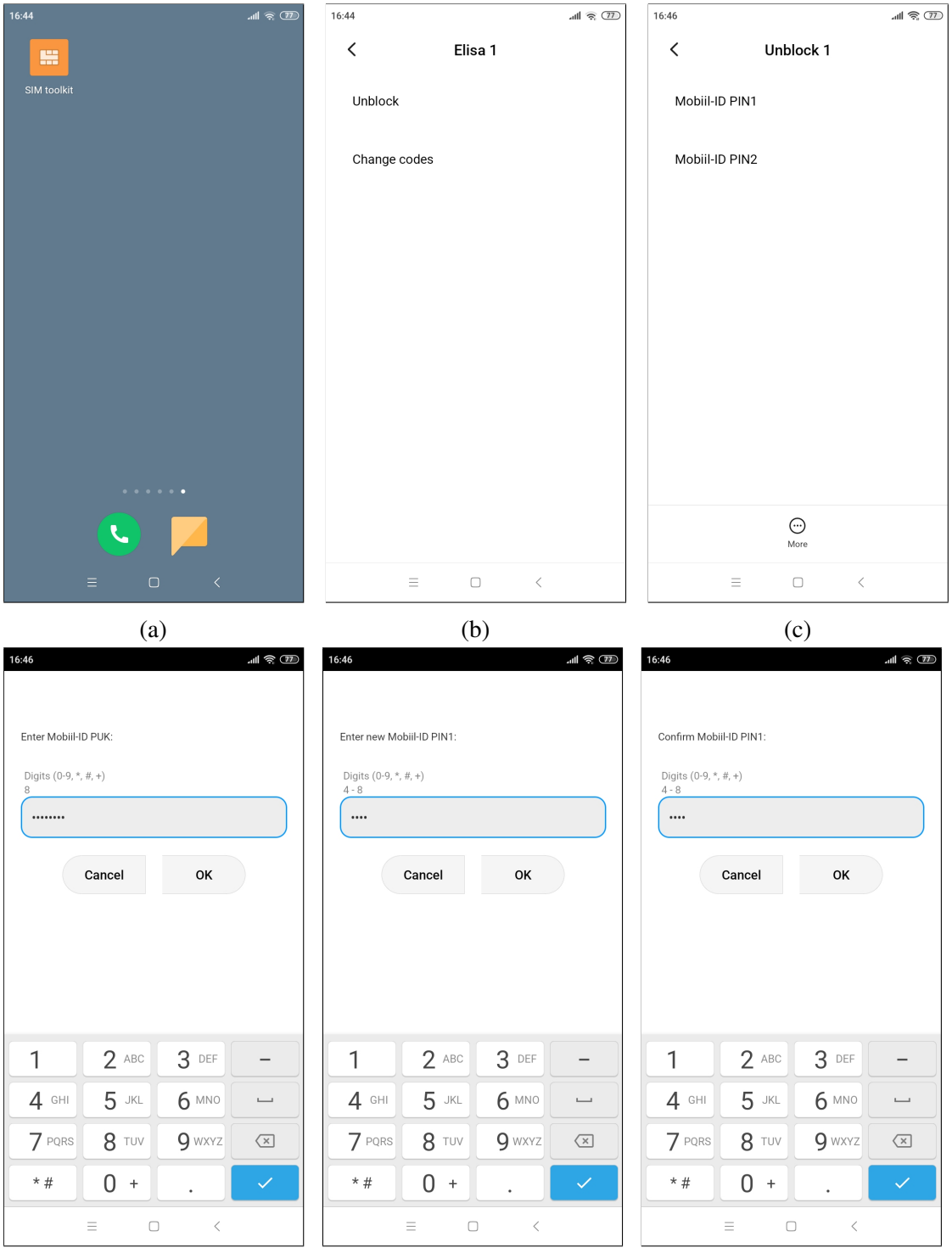
This section describes observed Mobile-ID-related command exchange between the phone and the SIM card. In total, there are three Mobile-ID-specific interactions between the phone and the SIM card, which are described in the three following subsections: setting up the menu, using the SIM card menu functionality and performing a Mobile-ID transaction.

First, a high-level description of interaction is given. Where appropriate, pictures illustrating the interaction from the user's perspective are given. After the general description of an interaction, an example APDU exchange for the interaction is given.

Each block of the APDU exchange starts with a centered comment, that describes the purpose of a C-APDU and the corresponding R-APDU. Directions of APDUs are shown by arrow symbols, placed before the bytes of APDUs being sent: right arrow ($\rightarrow$) signifies that APDU was sent from the phone to the SIM card, left arrow ($\leftarrow$) signifies that APDU was sent by the SIM card to the phone. Parts of the APDUs are annotated in italic.

4.2.1 Setting up menu

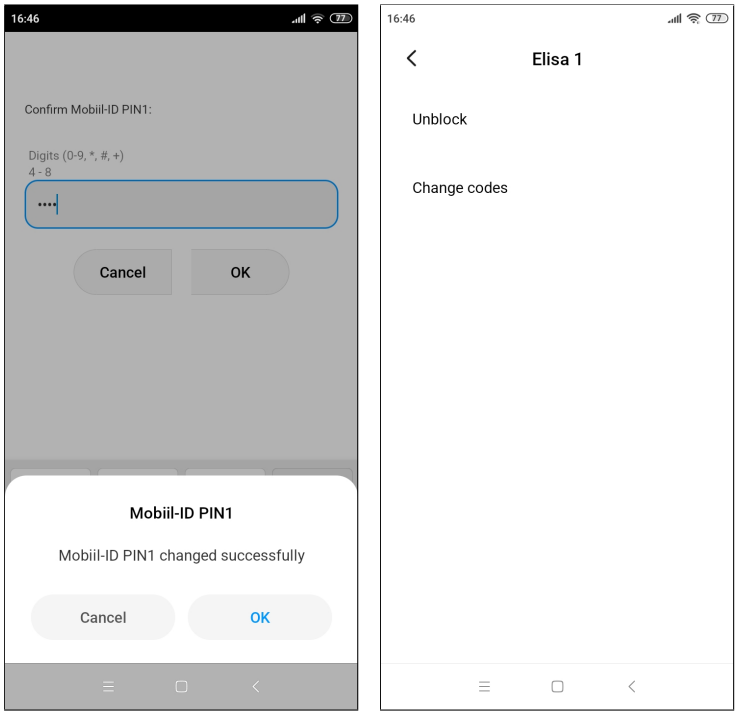
The SIM card menu can be used to change and unblock Mobile-ID codes. An example of UI elements shown to the user when unblocking Mobile-ID PIN1 is provided in Figure 7. The APDU exchange corresponding to unblocking PIN1 code is provided in the next subsection.



(a) (b) (c)

(d) (e) (f)

Figure 7. Unblocking Mobile-ID PIN1 from user's perspective



(g) (h)

Figure 7. Unblocking Mobile-ID PIN1 from user's perspective (continued)

For the SIM card menu to be shown to the user, the SIM card has to set it up each time after the phone boots up. The menu is set by the `SET UP MENU` proactive command. This proactive command specifies the menu title (Alpha identifier), menu items (entries), and, optionally, icons. This interaction does not involve the participation of the user. The `SET UP MENU` proactive command is available to the terminal for fetching after the terminal has sent `TERMINAL RESPONSE` as a response to the `SET UP EVENT LIST` proactive command.

The `SET UP EVENT LIST` proactive command[10] is used to specify to the terminal a list of events about which the SIM card should be notified in case they occur. When the event occurs, information about it is communicated by the phone to the SIM card using an `ENVELOPE` command. Mobile-ID-enabled SIM card issued by the Elisa Eesti AS reports to the

[10]We observed that at least some phones do not fetch proactive commands from a SIM card, before the user enters SIM card PIN.

phone an event list with only one event: “Language selection”. This means that the phone must report the language configured for the user interface every time the language changes.

After the phone boots up, the SIM card requests information about the interface language of the phone by sending a `PROVIDE LOCAL INFORMATION` proactive command. The phone returns its interface language in the `TERMINAL RESPONSE` C-APDU to the proactive command.

Menu entries of Mobile-ID, as well as other text strings used by the Mobile-ID applet, are localized. The Estonian Mobile-ID supports three languages: Estonian, English, and Russian. We have observed that if the language that is reported by the phone is supported by Mobile-ID, the SIM card menu is displayed in this language. If the language reported is not supported by the Mobile-ID, the entries of the SIM card menu are displayed in Estonian. However, the text of the buttons (e.g., the cancel button) is displayed in the language of the user interface of the phone. The language of the Mobile-ID prompt is determined by the Relying Party that initiated the transaction.

Regardless of the language used by the Mobile-ID applet, the applet uses the Estonian spelling “Mobiil-ID” when referring to the codes (for example: “Confirm Mobiil-ID PIN1:”). This spelling is used in all Mobile-ID interactions.

See Figure 8 for the example APDU exchange for setting up the menu. In this an the following figures *Tag and Value* of a TLV is abbreviated as *TL*, *Proactive Command* as *PC*, and *status word* as *SW*. All lengths values are provided in bytes. If *length* is mentioned in a description of TL, it refers to the length of the value of the TLV. If *length* is mentioned in a description of C-APDU header, it refers to the length of the C-APDU Data field.

```
←  91 2b
                                        SW of previous R-APDU,
                                              43 bytes to send
```

Setting up menu

```
→  80 12 00 00 2b
                                        FETCH, expected length: 43
←  d0 29
                                              PC TL, length: 41 bytes
   81 03 01 25 00
                                        Command details: SET UP MENU
   82 02 81 82
                          Device identities: direction is from SIM card to Terminal
   05 05
   45 6c 69 73 61
                                        Alpha Identifier: encodes "Elisa"
   8f 08
   80 55 6e 62 6c 6f 63 6b
                                                        Item TLV:
                                      encodes identifier 80 and "Unblock"
   8f 0d
   81 43 68 61 6e 67 65 20 63 6f 64 65 73
                                                        Item TLV:
                                    encodes identifier 81 and "Change codes"
   90 00
                                                    SW, no data to send
```

Responding that menu was set successfully

```
→  80 14 00 00 0c
                                  TERMINAL RESPONSE starts, length: 12
   81 03
                                        Command details: SET UP MENU
   02 02 82 81
                          Device identities: direction is from Terminal to SIM card
   83 01
                                        Result: Command performed successfully
←  90 00
                                                SW, no more data to send
```

Figure 8. Example interaction for setting up SIM menu (continued)

Note that the Alpha Identifier (menu title[11]) encodes "Elisa" and the items encode Mobile-ID related actions: "Unblock" and "Change codes". As we see, the SIM card applet contains both the Mobile-ID implementation code, and code (or, at least, string values) specific to the provider that has issued the Mobile-ID-capable SIM card. This allows to speculate that

[11] In general, Alpha Identifier specifies text that should be displayed on the screen while the terminal performs a proactive command.

the Mobile-ID-specific functionality is a "module" of the SIM card applet, i.e., it is a part that can be easily integrated into a SIM card applet's code.

4.2.2 Using the SIM card menu functionality

The Mobile-ID applet allows the Mobile-ID PIN1 and PIN2 codes to be unblocked if they were incorrectly entered multiple times. To unblock a Mobile-ID PIN, the user should first open the appropriate SIM card menu entry, pick the unblocked code, and then enter the Mobile-ID PUK code. After that, the user should enter a new PIN code, and then enter it one more time to confirm that it was entered correctly. An example of interaction for unblocking Mobile-ID PIN1 is provided in Figure 7.

Unblocking can also be performed during a Mobile-ID transaction interaction. If the user accepts the transaction prompt, but the corresponding PIN code is blocked, the user will first be asked if they want to unblock the code. If the user unblocks the code, the SIM card creates a signature and transaction is continued normally. Otherwise, the SIM card applet will process this as if the user has canceled the transaction.

If the user in the menu tries to unblock the code that is not blocked, the Mobile-ID applet will send `DISPLAY TEXT` APDU with the text "PIN is not blocked!".

The Mobile-ID applet also allows Mobile-ID PIN1, PIN2 and PUK codes to be changed. For this, the user should first open the appropriate menu entry, then pick the code that should be changed. If that code is blocked, the Mobile-ID applet will send an instruction to the phone to display the text "PIN is blocked!" and then behave the same as if the user tried to unblock the code. If the code is not blocked, the user should first enter the current code, then the new code, and then confirm the new code. The allowed length of PIN1 code is from 4 to 8 digits; for PIN2 code: from 5 to 8 digits; for the PUK code: exactly 8 digits.

If a PIN code is entered incorrectly, Mobile-ID displays the text "Wrong PIN!". A similar text ("Wrong PUK!") is displayed if PUK code is entered incorrectly. If the PIN code is entered incorrectly 3 times, Mobile-ID displays the text "PIN is blocked!" and blocks the code.

Mobile-ID does not allow to set codes that are too simple. If the user enters a new code and confirms it, but the new code is too simple, the "Wrong or predictable PIN, please try again" message will be shown, and the user will need to enter a new code and confirm it one more time. In case the user changes Mobile-ID PUK, "Wrong or predictable PUK, please try again" message is shown. A code is considered too simple if it is a repetition of the same digit or if it is either incremental or decremental from first digit to the last (e.g., 456789 or 5432). This is reflected in the WPKI recommendation document [12] (see requirement KR7 in Section 5.2). We confirmed this by conducting multiple attempts to change Mobile-ID codes to codes that are considered too simple according to these rules.

The procedure of interacting with the SIM card menu and the appearance of the SIM toolkit menu depend on how a particular phone model handles the STK commands. For entering the SIM toolkit menu, no APDUs are sent to the SIM card, as the phone stores the title and items of the SIM toolkit menu that were already communicated by the SIM card in the `SET UP MENU` proactive command (see Section 4.2.1 of this work).

Example APDU exchange for unblocking PIN1 is provided in Figure 9. In this example:

1. The phone communicates to the SIM card the fact that the user clicked "Unblock" menu item in the menu by sending an `ENVELOPE` command.
2. The SIM card sends a `SELECT ITEM` proactive command, to display the next menu screen to the user.
3. In the `TERMINAL RESPONSE` to the `SELECT ITEM` proactive command, the phone specifies that the user clicked "Mobiil-ID PIN1".
4. The SIM card issues a "chain" of multiple `GET INPUT` proactive commands, asking the user to enter their Mobile-ID PUK code, then the new Mobile-ID PIN1 code, then to confirm the new Mobile-ID PIN1 code.
5. After that, SIM card sends `DISPLAY TEXT` to notify the user that the code was unblocked successfully.

Communicating to the SIM card that "Unblock" was pressed in the menu

→ `80 c2 00 00 09`

`ENVELOPE` starts, length: 9

`d3 07`

TL: next 7 bytes are Menu selection

`82 02 01 81`

Device identities: Keypad to UICC

`90 01 80`

Item identifier; encodes `80` is for Unblock (as set up in the previous section)

← `91 36`

SW, 54 bytes to send

Communicating the menu screen for "Unblock", to the phone

→ `80 12 00 00 36`

`FETCH`, expected length: 54

← `d0 34`

PC TL, length: 52

`81 03 01 24 03`

Command details: `SELECT ITEM`; qualifier is `03`: display as list of items

`82 02 81 82`

Device identities: SIM card to Terminal

`85 07 55 6e 62 6c 6f 63 6b`

Alpha identifier: "Unblock"

`8f 0f 01 4d 6f 62 69 69 6c 2d 49 44 20 50 49 4e 31`

Item: "Mobiil-ID PIN1", identifier is `01`

`8f 0f 02 4d 6f 62 69 69 6c 2d 49 44 20 50 49 4e 32`

Item: "Mobiil-ID PIN2", identifier is `02`

`90 00`

SW, no more data to send

Communicating that "Mobiil-ID PIN1" was pressed, to the SIM

→ `80 14 00 00 0f`

`TERMINAL RESPONSE`, length: 15 bytes

`81 03 01 24 03`

Command details, same as in the Proactive Command

`02 02 82 81`

Device identities: Terminal to SIM card

`83 01 00`

Result: Command performed successfully

`10 01 01`

Item identifier: `01` for "Mobiil-ID PIN1"

Continued on the next page...

Figure 9. Example interaction for unblocking Mobile-ID PIN1

```
←  91 26
```
SW, 38 bytes to send

Asking user to enter Mobiil-ID PUK

```
→  80 12 00 00 26
```
FETCH, expected length: 38

```
←  d0 24
```
PC TL, length: 36

```
   81 03 01 23 04
```
Command details: GET INPUT, qualifier encodes: "digits (0 to 9, , #, and)", hidden entry mode

```
   82 02 81 82
```
Device identities: SIM card to Terminal

```
   8d 15
```
Text string TL, length: 21 bytes

```
   04
```
string is encoded in GSM default alphabet

```
   45 6e 74 65 72 20 4d 6f 62 69 69 6c 2d 49 44 20
   50 55 4b 3a
```
encodes "Enter Mobiil-ID PUK:"

```
   91 02 08 08
```
Response length TLV: allowed length is at least 8 and at most 8 symbols

```
   90 00
```
SW, no more data to send

Relaying the entered Mobiil-ID PUK to the SIM card

```
→  80 14 00 00 17
```
TERMINAL RESPONSE, length: 23

```
   81 03 01 23 04
```
Command details, same as in the PC

```
   02 02 82 81
```
Device identities: SIM card to Terminal

```
   83 01 00
```
Result: Command performed successfully

```
   8d 09 04 31 32 33 34 31 32 33 34
```
Text string TLV, encodes "12341234" in GSM default alphabet

```
←  91 2b
```
SW, 43 bytes to send

Asking user to enter new Mobiil-ID PIN1

```
→  80 12 00 00 2b
```
FETCH, expected length: 43

Continued on the next page...

Figure 9. Example interaction for unblocking Mobile-ID PIN1 (continued)

```
    d0 29 81 03 01 23 04 82 02 81 82 8d 1a 04 45 6e
    74 65 72 20 6e 65 77 20 4d 6f 62 69 69 6c 2d 49
←   44 20 50 49 4e 31 3a 91 02 04 08
```

GET INPUT PC
with text "Enter new Mobiil-ID PIN1:"
and allowed length from 4 to 8 symbols

```
    90 00
```

SW, no more data to send

Relaying the new Mobiil-ID PIN1 to the SIM card

```
    80 14 00 00 13 81 03 01 23 04 02 02 82 81
→   83 01 00 8d 05 04 30 30 30 31
```

TERMINAL RESPONSE with
Text string "0001"

```
←   91 29
```

SW, 41 bytes to send

Asking user to confirm the Mobiil-ID PIN1

```
→   80 12 00 00 29
```

FETCH, expected length: 41

```
    d0 27 81 03 01 23 04 82 02 81 82 8d 18 04 43 6f
    6e 66 69 72 6d 20 4d 6f 62 69 69 6c 2d 49 44 20
←   50 49 4e 31 3a 91 02 04 08
```

GET INPUT PC
with text "Confirm Mobiil-ID PIN1:";
and allowed length is from 4 to 8 symbols

```
    90 00
```

SW, no more data to send

Relaying the new Mobiil-ID PIN1 to the SIM card

```
    80 14 00 00 13 81 03 01 23 04 02 02 82 81 83 01
→   00 8d 05 04 30 30 30 31
```

TERMINAL RESPONSE
with Text string "0001"

```
←   91 31
```

SW, 49 bytes to send

Notifying user that PIN1 was changed

```
→   80 12 00 00 31
```

FETCH, expected length: 49

```
←   d0 2f
```

PC TL, length: 47

```
    81 03 01 21 81
```

Command details: DISPLAY TEXT

```
    82 02 81 02
```

Device identities: SIM card to Display

Continued on the next page...

Figure 9. Example interaction for unblocking Mobile-ID PIN1 (continued)

```
    8d 24 04 4d 6f 62 69 69 6c 2d 49 44 20 50 49 4e
    31 20 63 68 61 6e 67 65 64 20 73 75 63 63 65 73
    73 66 75 6c 6c 79
```

Text string "Mobiil-ID PIN1 changed successfully"

```
    90 00
```

SW, no more data to send

Responding to the SIM card that the command was executed successfully

```
    80 14 00 00 0c 81 03 01 21 81 02 02 82 81
→   83 01 00
```

TERMINAL RESPONSE, command performed successfully; Device identities: SIM card to Terminal

```
←   90 00
```

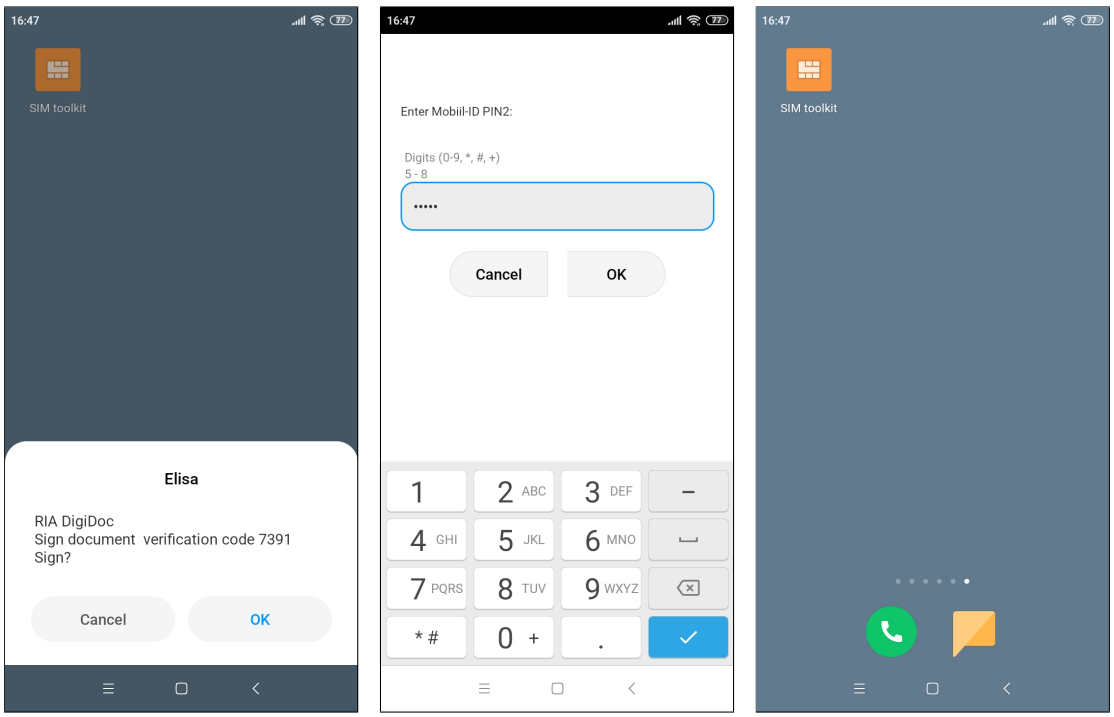
status word, no more data to send

Figure 9. Example interaction for unblocking Mobile-ID PIN1 (continued)

An interesting observation can be made about how the `GET INPUT` command is used in this interaction. Namely, according to the standard, it allows symbols "⋆", "#" and "+" to be used as "digits". We performed an experiment to examine whether this means that the user can change their Mobile-ID codes to include these symbols. In this experiment, we changed Mobile-ID PIN1 to "`+++1`". The Mobile-ID applet accepted this input as a valid new PIN1, and accepted it to confirm a Mobile-ID transaction. The standard prescribes that "+" is not allowed for user input in hidden entry mode (Section 6.4.3 in [3]). Despite the SIM card reporting to the phone that the hidden mode must be used (see Figure 9), the Redmi 7A phone (original firmware) allowed the input of this symbol, and aforementioned PIN1 was accepted by the applet.

4.2.3 Performing a Mobile-ID transaction

This interaction is the most frequent Mobile-ID interaction. This is the interaction where the incoming and outgoing Mobile-ID SMS messages are exchanged — it happens every time a user performs a Mobile-ID authentication or digital signature transaction. An example of this interaction from the user's perspective is provided in Figure 10.



(a) (b) (c)

Figure 10. Conducting Mobile-ID transaction from the user's perspective (continued)

In this interaction, the phone displays a text (the Mobile-ID prompt) with the details of the transaction that may be performed, including the name of the Relying Party, a description of the transaction (provided by the Relying Party), type of the transaction (authentication of signing) and the verification code. After the user agrees to perform the transaction, they are shown a screen with an input field, where they should enter their Mobile-ID PIN, corresponding to the type of the transaction (PIN1 for authentication, PIN2 for digital signing).

An example of an APDU exchange for a successful interaction is provided in Figure 11. In this example:

1. The phone delivers the incoming Mobile-ID service SMS to the SIM card by sending an `ENVELOPE` command.
2. The SIM card sends `PLAY TONE` proactive command to play a beep sound (tone).
3. Then it sends `DISPLAY TEXT` to display the Mobile-ID prompt.

4. After the terminal responds that the user has clicked “OK”, the SIM card sends `GET INPUT` to retrieve the Mobile-ID PIN2.
5. After the terminal responds with the entered code, the SIM card creates an outgoing Mobile-ID service SMS, and sends `SEND SHORT MESSAGE` proactive command with this message to the phone. The phone must send this message over the cellular network to the recipient included in the message, over the cellular network. Data objects that are used to exchange SMS messages (SMS TPDUs) will be discussed in more detail in the next chapter.

Relaying the received class 2 SMS message to the SIM card

→	`80 c2 00 00 87`	
		`ENVELOPE`, length: 135
	`d1 81 84`	
		SMS-PP Download TL, length: 132
	`82 02 83 81`	
		Device identities: from Network to SIM card
	`06 07 91 73 52 16 00 20 f0`	
		Address: 37256100020
	`8b 75`	
		SMS TPDU TL, length: 117 bytes
	`64`	
		`TP-MTI` of this byte encodes that this is a `SMS-DELIVER` TPDU
	`05`	
		`TP-Originating-Address` starts, length: 5 digits
	`a1`	
		National number, Numbering plan: ISDN/telephone
	`59 45 f4`	
		Digits are: 95544
	`7f f6`	
		`TP-PID` and `TP-DCS`
	`22 10 81 11 41 60 80`	
		`TP-Service-Centre-Time-Stamp`
	`65`	
		`TP-User-Data-Length`: 101
	`<TP-User-Data omitted>`	
		see Figure 14 for an example

Continued on the next page...

Figure 11. Example interaction for Mobile-ID transaction

←	`91 12`
	SW, 18 bytes to send
	Playing a notification tone
→	`80 12 00 00 12`
	`FETCH`, expected length: 18
←	`d0 10 81`
	PC TL: length 16
	`03 01 20 00`
	Command details: `PLAY TONE`, qualifier encodes: use of vibrate alert is up to the terminal
	`82 02 81 03`
	Device identities: SIM card to Earpiece
	`0e 01 10`
	Tone TLV: General beep
	`04 02 01 01`
	Duration TLV: 1 second
	`90 00`
	SW, no more data to send
	Notifying the SIM card that the command was performed successfully
→	`80 14 00 00 0c 81 03 01 20 00 02 02 82 81 83 01 00`
	`TERMINAL RESPONSE` from Terminal to SIM card; command performed successfully
←	`91 45`
	SW, 69 bytes to send
	Displaying the information about the transaction
→	`80 12 00 00 45`
	`FETCH`, expected length: 69
←	`d0 43 81`
	PC TL, length: 67
	`03 01 21 81`
	Command details: `DISPLAY TEXT`, qualifier encodes: display with high priority and wait for user to clear the message
	`82 02 81 02`
	Device identities: SIM card to Display

Continued on the next page...

Figure 11. Example interaction for Mobile-ID transaction (continued)

```
    04 52 49 41 20 44 69 67 69 44 6f 63 0a 61 61 61
    61 61 61 61 61 61 61 61 61 20 0d 76 65 72 69 66
    69 63 61 74 69 6f 6e 20 63 6f 64 65 0d 30 30 30
    30 20 0d 53 69 67 6e 3f
```

Text string:
`RIA DigiDoc\n`
`aaaaaaaaaaa \n`
`verification code\n`
`0000 \n`
`Sign?`

```
    90 00
```

SW, no more data to send

Notifying the SIM card that the text was displayed

```
→   80 14 00 00 0c 81 03 01 21 81 02 02 82 81 83 01 00
```

`TERMINAL RESPONSE` *from Terminal to SIM card; command performed successfully*

```
←   91 27
```

SW, 39 bytes to send

Asking the user to enter Mobile-ID PIN2

```
→   80 12 00 00 27
```

`FETCH`*, expected length: 39*

```
←   d0 25
```

PC TL, length: 37

```
    81 03 01 23 04
```

Command details: `GET INPUT`*, qualifier encodes: "digits (0 to 9, , #, and)", hidden entry mode*

```
    82 02 81 82
```

Device identity: SIM card to Terminal

```
    8d 16 04 45 6e 74 65 72 20 4d 6f 62 69 69 6c 2d
    49 44 20 50 49 4e 32 3a
```

Text: "Enter Mobiil-ID PIN2:"

```
    91 02 05 08
```

Response length: allowed length is at least 5 and at most 8 symbols

```
    90 00
```

SW, no more data to send

Relaying the Mobiil-ID PIN2 to the SIM card

```
    80 14 00 00 14 81 03 01 23 04 02 02 82 81
→   83 01 00 8d 06 04 30 30 30 30 31
```

`TERMINAL RESPONSE` *with Text string "00001"*

```
←   91 73
```

SW, 115 bytes to send

Continued on the next page...

Figure 11. Example interaction for Mobile-ID transaction (continued)

Sending the outgoing SMS with a signature

→	80 12 00 00 73	
		FETCH, expected length: 115
←	d0 71	
		PC TL, length: 113
	81 03 01 13 00	
		Command details: SEND SHORT MESSAGE
	82 02 81 83 06 07 91 73 52 16 00 20 f0	
		Address: 37256100020
	0b 5d	
		SMS TPDU TL, length: 93
	01	
		TP-MTI of this byte encodes that this is a SMS-SUBMIT TPDU
	00	
		TP-MR: 0
	05 81 59 45 f4	
		TP-Destination-Address: 95544
	00 f6	
		TP-PID and TP-DCS
	53	
		TP-User-Data-Length: 83
	<TP-User-Data omitted>	
		see Figure 15 for an example
	90 00	
		SW, no more data to send

Responding to the SIM card that SMS was sent

→	81 03 01 13 00 02 02 82 81 83 01 00	
		TERMINAL RESPONSE
←	90 00	
		SW, no more data to send

Figure 11. Example interaction for Mobile-ID transaction (continued)

The Duration TLV in the PLAY TONE command contains a value of 1 second. However, according to the standard (Section 6.6.5 in [3]), this value should be ignored if a single (not a continuous or repeatable) tone is played. The Redmi 7A phone plays a single tone, and the tone duration is less than one second.

The GET INPUT proactive command for PIN entry is the same as the command used for PIN entry in the SIM card menu (the only difference is the text, specifying which code should be entered).

If the user cancels the transaction, the SIM card also sends a SEND SHORT MESSAGE proactive command. However, this message does not include a signature and has a different status code. See Figure 16 for an example of such message.

The behavior of the phone, in case the user does not react to the prompt, depends on the firmware of the phone.

It was observed that Nokia 225 Dual SIM phone sends the TERMINAL RESPONSE command to the SIM card, if the phone does not obtain user response within 60 after the proactive command is received. The Result value of this TERMINAL RESPONSE is 12 ("No response from user"). After the SIM card receives such TERMINAL RESPONSE, it sends the same message as if the user has canceled the prompt (see Figure 16), but with the transaction status containing value 05.

If such SMS is sent to the cellular network during the Mobile-ID transaction, the Application Provider completes the Mobile-ID session with the relying party by returning TIMEOUT status code.

Motorola C115 phone (with original firmware) only sends the TERMINAL RESPONSE after it obtains user response.

This means that the phone will continue to show the prompt, even long after the MID session between the Application Provider and the Relying Party has timed out. The timeout for Mobile-ID transactions, enforced by the Application Provider, is approximately 120 seconds.

Redmi 7A phone closes the prompt if the user does not react to it within 10 seconds. It was observed that if the prompt is closed by the phone, the Application Provider ends the session with the Relying Party with USER_CANCELED result. It is safe to assume[12] that the session ended because the phone sent the TERMINAL RESPONSE with Result value 11 ("Backward move in the proactive UICC session requested by the user"); and that the SIM card issued the SEND SHORT MESSAGE proactive command with an outgoing Mobile-ID message containing the corresponding transaction status (01).

[12]We were not able to confirm this because, unfortunately, we did not have a flexible PCB that would allow us to connect the Redmi 7A phone to the Simtrace2.

5 Mobile-ID transaction SMS messages

In Mobile-ID, SMS service messages are used to exchange the transaction data. Two types of Mobile-ID service SMS messages are used to implement Mobile-ID transactions. An incoming SMS is used to transmit information necessary to produce a signature from the OTA server to the SIM card, and an outgoing SMS is used to transmit a signature from the SIM card to the OTA server. Firstly, this chapter provides general information about SMS messages and how they are transmitted. After that, the chapter describes the formats of the messages specific to Mobile-ID transactions.

5.1 General information about SMS

Sending and receiving SMS messages is described in 3GPP technical specification 23.040 [16]. This standard defines data objects that are used to exchange SMS messages. These data objects are also referred to as TPDUs (short for Transfer Protocol Data Units), or simply messages.

For example, a device connected to the cellular network can send an SMS message by transmitting the `SMS-SUBMIT` TPDU to the network. The network relays this TPDU to the server processing SMS messages (SMSC). After SMSC receives this TPDU, it converts it into a `SMS-DELIVER` TPDU, which is then relayed to the destination device. This device receives this TPDU as an SMS message. If the receiving device is a phone, it may display this message or relay it to the SIM card (depends on the type of the message). Structures of `SMS-SUBMIT` and `SMS-DELIVER` TPDUs are presented in Figures 12 and 13. Names of the fields that are present in both of the TPDUs are outlined in green color.

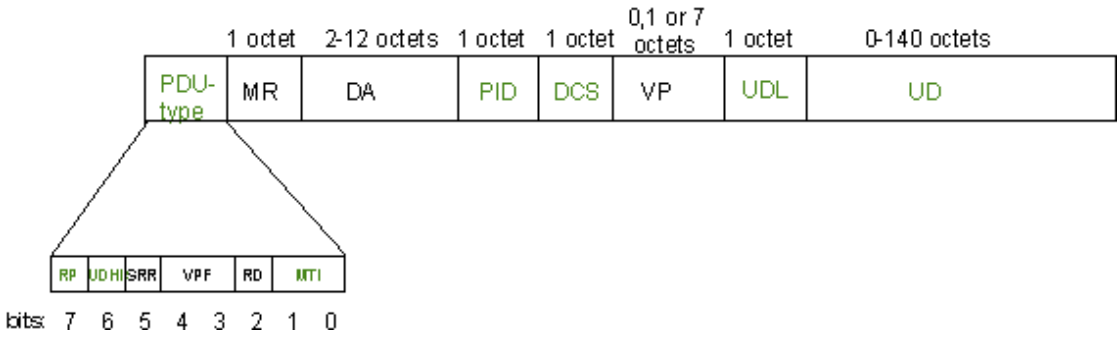


Figure 12. Structure of SMS-SUBMIT TPDU (based on [19]).

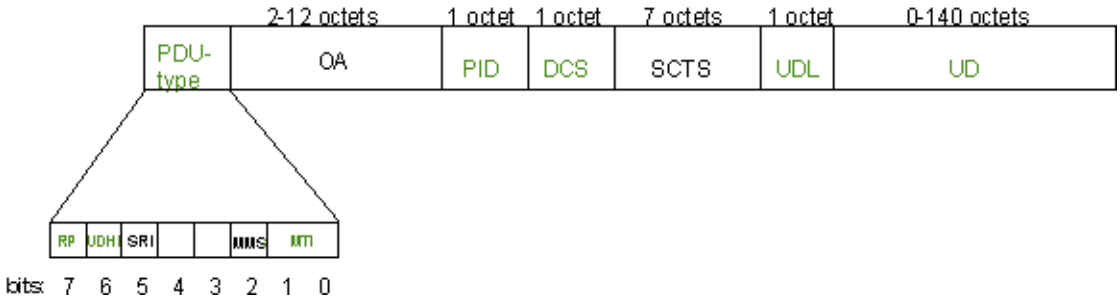


Figure 13. Structure of SMS-DELIVER TPDU (based on [19]).

In the Mobile-ID SMS-SUBMIT TPDUs sent from the SIM card to the OTA server, the length of field VP is 0 (i.e., VP is not present).

The standard provides descriptions of different fields that are present in the SMS TPDUs (Section 9.2.3 in [20]):

- The field TP-MTI encodes the type of the SMS TPDU, e.g., SMS-SUBMIT or SMS-DELIVER.
- The field TP-Originating-Address (TP-OA) that is present in the SMS-DELIVER TPDU contains information about the supposed sender of the SMS. If a sender is a cellular network subscriber (regular user), then the field will contain their phone number.
- The field TP-Destination-Address (TP-DA) that is present in the SMS-SUBMIT TPDU contains information about the destination of the SMS. Usually, it contains the phone number of the cellular network subscriber to whom the message should be delivered.

- The field `TP-PID` (Protocol Identifier) contains information about a higher-level protocol being used (if any), or indicates interworking with certain types of equipment. In a usual user-to-user text SMS, this value is 0. The value of this field may indicate to the phone that the SMS should be forwarded to the SIM card (see below).
- The field `TP-DCS` (Data Coding Scheme) contains an encoding of the contents of the message, and also the class of the message. For example, in a usual user-to-user text SMS in UCS2 encoding, this value is 8. This field should be set correctly so the receiver can recognize the class of the SMS.
- The field `TP-User-Data` contains the content of the SMS message.

Different encodings may be used in the SMS messages. Some of them are: GSM-7 encoding, UCS-2 encoding and binary data encoding [21]. A message that only consists of Latin letters and punctuation marks can be sent as a GSM-7 encoded message, that may contain up to 160 characters[13]. A message containing symbols such as "Õ" or "Ъ" can not be transmitted as a GSM-7 encoded SMS, and thus, the encoding used for a message containing these symbols is UCS-2. UCS-2-encoded message (if not concatenated) can contain up to 70 characters. The Mobile-ID SMS (both incoming and outgoing) use raw data encoding. The content of an SMS message in raw encoding can consist of up to 140 octets.

5.2 Format and processing of the incoming message

Incoming Mobile-ID SMS is a partially encrypted message that is used to deliver the hash to be signed to the SIM card. It is created and sent by the OTA server. The WPKI recommendations document [12] (see definition for OTA in Section 2) implies that this message is a Secured Packet, as defined in the 3GPP technical specification

[13]The concatenation mechanism allows sending longer texts using multiple TPDUs. The use of 2048-bit RSA as a Mobile-ID signature algorithm would require that this mechanism is used. However, in practice, RSA is not used in modern-day Mobile-ID. Thus, the SMS concatenation mechanism is not used in Mobile-ID.

03.48 (Sections 5.1 and 6.2 in [15]).

It is known that the contents of this message include the hash to be signed (as this is the only channel that can be used to deliver the hash value to the SIM card). Also, the contents of this message include: the name of the Relying Party (as displayed in the transaction prompt), display text string (and its format), the UI language that should be used to display the prompt and the type of the transaction (authentication or digital signature). As will be shown in Section 5.3, contents of this message also include a field that acts as an identifier of the transaction. All the values specific to a given transaction are contained inside the Secured Data portion of this Secured Packet — i.e., in the encrypted part of the message. The structure of the incoming Mobile-ID messages is the same regardless of the type of the transaction (authentication or digital signature).

The `TP-Originating-Address` of this SMS always contains the same value — the address of the OTA server. For SIM cards issued by Elisa EE, this value is 95544. For SIM cards issued by Tele2 EE, this value is 95746. The field `TP-DCS` of this SMS always contains the value `0xF6`, which indicates that it is a class 2 SMS that uses the raw data (binary) encoding. The field `TP-PID` of this message contains value `0x7F` (SIM Data download), which specifies to the phone that this message should be transmitted to the SIM card.

If the SMSC does not receive the confirmation from the phone that an incoming Mobile-ID SMS is received, it will retry sending it one more time.

An example of the content (the `TP-User-Data` field) of an incoming Mobile-ID SMS message is shown in Figure 14. This figure also provides descriptions for parts of this SMS. In it, the monospace text is a hex representation of the body of an incoming Mobile-ID SMS; the text on the left (in italic) describes different parts of the message.

The field `Header` contains a constant value and can be further split up into three fields: `UDHL`, `IEIa` and `IEIDLa`. The incoming Mobile-ID service message contains header value `02 70 00`, as prescribed by the technical specification defining Secured Packets.

```
        Header  02 70 00
           CPL  00 60
           CHL  15
           SPI  16 00
           KIc  15
           KID  15
           TAR  50 4b 49
encrypted-part  14 ba bc 60 0a 76 78 9c 05 da 67 13 62 3b 08 17
                47 d3 3d 3f 37 9e 06 37 c6 96 cb 71 3d 4f e8 29
                9d 67 b9 3f e1 dc bc 8a 82 87 6b 94 19 44 c6 f9
                b3 0a 07 98 4b f0 e0 9f ab 13 f3 6a 12 aa c2 f3
                43 3a bf 3d 7e fb 36 9f 1a a5 ed b6 99 a4 5b d4
                ac d4 4e e7 96 b1 05 ed
```

Figure 14. Example of the `TP-User-Data` of an incoming Mobile-ID SMS.

The field `CPL` contains the total length of the fields that follow. In Figure 14, this length is 96 bytes, but it can be different depending on the length of the display text for the prompt.

The value of the field `CHL` (Command Header Length) indicates that the total length of the fields that follow before the Secured Data is 21 bytes.

The field `SPI` (Security Parameter Indicator) encodes that the message includes a counter, and that the message should be discarded if the value of this counter is less or equal to the counter value of the last received Secured Packet. It also indicates that the message has been encrypted and has been protected by a cryptographic checksum. The use of these cryptographic mechanisms is in line with the WPKI forum recommendations document [12] (see TR2 in Section 5.1) and their use was partly confirmed by our experiments (see below). Additionally, the `SPI` field indicates that no Proof of Receipt reply should be sent when this SMS message is received.

The fields `KIc` (Ciphering Key Identifier) and `KID` (Key Identifier) encode that the encryption algorithm for both the Ciphering and the Cryptographic Checksum is “Triple DES in outer-CBC mode using two different keys”, and that the same key with index 1 is used.

Note that the TAR (Toolkit Application Reference) field encodes ASCII-string "PKI".

Fields from CHL to TAR always contain the same value. An experiment was conducted to see if the SIM card discards the message if any of the bits in these fields are changed. It was observed that this is indeed the case: the SIM card rejects the message (by responding with 90 00 status word) if any of the bits of the message are changed. However, if the two least significant bits of the second byte of SPI are changed, the returned status word will instead be 62 00 (a warning "No information given"). If the same SMS message is supplied again, the SIM card will act upon it (by responding with a status word starting with 91). This means that if any of these fields are modified, the SMS message is discarded by the SIM card without increasing the counter.

The fields of the message that follow (including the Secured Data) are encrypted (ciphered). According to the standard that defines Secured Packet [15], these fields are:

- Counter (length: 5 bytes)
- Padding counter (length: 1 byte) — encodes the number of padding octets. The data must be padded to the multiple of 8 octets, to ensure that the data can be encrypted using the block cipher (3DES in this case).
- Redundancy Check, Cryptographic Checksum or Digital Signature. The field SPI indicates that the message uses Cryptographic Checksum.
- Secured Data — the data specific to the Mobile-ID transaction.

Based on the value of CHL and the total length of previous fields, it can be calculated that the length of the Cryptographic Checksum field is 8 bytes. According to the standard [15], the fields that are included in deriving Cryptographic Checksum are: Secured Data (with Padding), all the fields from SPI to PCNTR (inclusive), and (optionally) fields CPL and CHL. The checksum is calculated before the fields (including the checksum) are encrypted.

We confirmed the fact that the message is encrypted, as the value of

this field obviously carries data, and also looks random. The latter can be checked with a simple randomness test: the `encrypted-part` field values for 39 SMS were concatenated, and then the byte frequency distribution of the concatenation was calculated. The frequency distribution was similar to such distribution of a random byte string.

We confirmed the fact that the SIM card verifies the counter value. The counter is important to prevent attacks against the users that rely on the rearrangement of the incoming Mobile-ID SMS. We conducted an experiment where two incoming SMS were intercepted and supplied to the SIM card in reverse order. After receiving the SMS message that was intercepted last, the SIM card acted upon it (by responding with `91 xx` status word) and started sending proactive commands. The SIM card discarded the other SMS (that was intercepted first), by responding with `90 00` status word. Therefore, the SIM card verifies the counter included in the message. The SIM card also refuses to react to the SMS if its counter value is equal to the latest seen value: the same SMS received by the SIM card twice will be processed by the SIM card only when it is received for the first time. It was observed that the counter value is updated even if the phone abruptly ends the communication with the SIM card right after the Secured Packet is submitted to the SIM card — i.e., the phone does not send `FETCH` C-APDU after the SIM card responds with a status word starting with `91`.

If an incoming Mobile-ID SMS is supplied to the SIM card while the user is still interacting with a Mobile-ID prompt of a previous transaction, the SIM card discards the incoming SMS, but the counter value is still updated.

We confirmed that the SIM card verifies the checksum of the message by performing the following experiment. We intercepted an incoming Mobile-ID message. We made several attempts to supply it to the SIM card, but each time one bit in some position of the `encrypted-part` field was flipped. A separate attempt was made for each bit of the same intercepted SMS. The experiment showed that the SIM card discarded the altered SMS with status word `90 00`, no matter which bit was changed.

After that original SMS was supplied, the SIM card responded with a status word starting with `91`. This experiment shows that the SIM card verifies the integrity of the message, and that the SIM card does not update the counter if the message does not pass the integrity check.

Another experiment was conducted to verify that the Mobile-ID applet would reject messages that are encrypted for a different SIM card. In this experiment, two SIM cards, both issued by Elisa EE were used. One incoming Mobile-ID SMS was intercepted for each of the SIM cards. Then each message was supplied to the "wrong" card[14]. Upon receiving these messages, both SIM cards rejected them. This experiment does not show that all correctly formed Secured Packets are rejected if sent to the "wrong" card, but rather that this is likely that different encryption keys are used to encrypt Secured Packets for different SIM cards.

It is known that a Mobile-ID-enabled SIM card has 4 keypairs — two for creating RSA cryptographic signatures and two for ECC signing. Mobile-ID REST API documentation [2] provides an interesting insight. In Section 3.3.5 (field `signature.algorithm`) it states that the Application Provider selects which signature algorithm will be used for the operation. It means that the type of key (ECC or RSA) that should be used to produce a signature, is likely to be contained in the incoming SMS.

An observation can be made by examining the length of the encrypted part of the incoming SMS. Namely, the length of the encrypted part never changes in relation to the language used in the transaction prompt. However, localization strings for each language, that is supported by the Mobile-ID, require a different number of bytes to encode. This means that it is very likely that the localized strings ("verification code", "Enter?" and "Sign?" and their variants for different languages) are stored in the SIM card, and that the incoming SMS only contains a flag denoting the language to be used for the transaction.

One more experiment was performed to test if the SIM card would

[14] Supplying "wrong" SMS to multiple cards (as opposed to sending just one SMS message to a "wrong" card) is important to exclude the possibility that a SIM card successfully decrypts a message, but then rejected because of its counter value.

accept an SMS, where the value of the TP-Originating-Address does not match the address of the corresponding OTA server. This experiment showed that a Mobile-ID-enabled SIM card issued by Elisa EE would act upon such a message, given that the message is not rejected for another reason.

Interestingly, the Elisa EE network does not prevent the network subscribers from sending service SMS messages — i.e., the network allows its subscribers to send arbitrary service SMS messages to other Elisa EE subscribers. This was verified by performing the following experiment. In this experiment, two pairs, each consisting of a phone and an Elisa EE operated SIM card, were used (pairs are referred to as A and B). The incoming Mobile-ID message intended for SIM card A was intercepted. Then, this SMS was transferred to phone B, and sent from phone B to phone A. It was observed that the phone A indeed received the message, forwarded it to the SIM card, and the Mobile-ID applet acted upon it by initiating the displaying of the Mobile-ID transaction prompt After the prompt was confirmed, the SIM card issued a SEND SHORT MESSAGE proactive command to the phone, containing an outgoing Mobile-ID message. The message was correctly formed. Interestingly, the Destination Address of this SMS still contained the correct address of the OTA server of the Elisa EE network (95544) and not the number of the phone B from which the incoming Mobile-ID SMS was received. Unfortunately, we could not verify whether the OTA server would accept this message. Each time we attempted to perform the experiment, the Mobile-ID transaction timeout (120 seconds) happened before we could send the outgoing message to the OTA server, thus leaving us no opportunity to receive feedback on how the message was processed. Still, we believe that, since the SMS message was formatted correctly, it is very likely that it would be accepted by OTA server. This experiment shows that the address of the OTA server is stored in the SIM card.

We performed a number of experiments to learn if the SIM card will reject incoming Mobile-ID messages if their TP-DCS and/or TP-PID values are incorrect. The result of the experiments is that the SIM card

acts upon the messages even if TP-DCS and TP-PID field values are both set to 0, as long as the message is otherwise correctly formatted and contains the correct values.

5.3 Format and processing of the outgoing message

An outgoing Mobile-ID SMS message is created by the SIM card to return the created signature to the Application Provider. The value of TP-Destination-Address of this message is always the same: it matches the TP-Originating-Address of the incoming Mobile-ID message. Therefore, the recipient of this message is the OTA server. The value of the field TP-DCS is 0xF6 (class 2 message with raw data). The value of the field TP-PID is 0. Contrary to the incoming Mobile-ID message, this message does not follow the Secured Packet format and it is not encrypted.

Figure 15 shows and describes the content of this SMS message. Figure 16 shows an example of an SMS that is sent if the user cancels the transaction. The structure of these messages is the same regardless of the type of the transaction (authentication or digital signature), for which these messages are transmitted.

```
        header  f1
          type  00
hex-identifier  65 64 38 62 37 61 36 61
      outer-TL  42 47
        seq-TL  30 45
      int-a-TL  02 21
       int-a-V  00
                97 73 c4 f6 2f 0a 92 2a 1e 76 32 84 2f 1d 1c 3f
                5d 92 e6 bb 9f f8 77 fd 6d 60 80 ca a0 23 7c 6a
      int-b-TL  02 20
       int-b-V  2f 4a 93 02 eb 1d 52 28 15 a7 57 eb d3 a4 6c ae
                73 b3 25 d5 fb 8a d6 ee 37 d5 55 da 83 57 ca 90
```

Figure 15. Example of the TP-User-Data of SMS containing the user's cryptographic signature

```
        header  f1
          type  01
hex-identifier  35 36 34 63 37 61 65 31
```

Figure 16. Example of the `TP-User-Data` of an SMS that is sent if user cancels the process

The value of the field `header` in all the messages is always `f1`. Since it is not overruled that the receiver of this message (the OTA server) may receive messages of other types, it can be speculated that this value signifies to the OTA server that this SMS is a Mobile-ID SMS message.

The field `type` indicates the type of the outgoing SMS (status of the transaction). For a message of a successfully completed Mobile-ID transaction, this value is `00`. In case the user canceled the transaction, the value of this field is `01`. It was confirmed during the research that if an incorrect value is supplied in this field (e.g., `02`), the Application Provider will end the session with the Relying Party by returning an error result code `SIM_ERROR` (Section 3.3.8 in [2]).

The field `hex-identifier` contains 8 bytes that contain ASCII representation of lowercase hexadecimal symbols that encode a 4-byte value. In Figure 15, this value encodes ASCII string `e964ff6d`. This value is different for every outgoing SMS. It seems to be random and unrelated to other fields. An experiment was conducted to verify that the value of this field ties an outgoing SMS to a specific incoming SMS (i.e., serves as a Mobile-ID transaction identifier). Two signature transactions were started for this experiment, both involving the same Mobile-ID-enabled SIM card. For both transactions, the Relying Party supplied the same hash and the message to be displayed in the prompt. Incoming Mobile-ID SMS of both transactions were captured, and then delivered to the SIM card. After performing two interactions with the phone, the SIM card produced two outgoing SMS. A combined outgoing SMS was crafted by taking the outgoing message of the first transaction, and replacing its `hex-identifier` with the `hex-identifier` of the outgoing message of the second transaction. This combined SMS was sent to the

SMSC. It was observed that the Application Provider sent the signature to the Relying Party inside the *second* session, while the first session timed out. This means that the `hex-identifier` field ties outgoing SMS to a specific transaction — i.e., this field allows recognizing the outgoing SMS as a response to a specific incoming SMS.

It is safe to assume that the value of field `hex-identifier` is included in the incoming Mobile-ID SMS in the Secured Data field, as there there is no other possible source where the SIM card may obtain this value (unless the SIM card generates this value).

A simple randomness test was conducted. In it, `hex-identifier` field values for 25 SMS were concatenated and then decoded from hexadecimal symbols to bytes, and then byte frequency distribution was calculated for this byte array. The frequency distribution was similar to such distribution for random bytes. Also, the character frequency of the concatenated hexadecimal values was measured: it was similar to character frequency distribution for random hex strings. It is not known if `hex-identifier` value is generated by the TSP platform or by the OTA server. This value is not generated by the Relying Party, as this value is not included in the MID REST API request. Nor it is generated or by the SMSC, as this server does not implement Mobile-ID-specific logic (moreover, the incoming message is encrypted after it leaves the OTA server).

The fields that follow after the `hex-identifier` encode the value of the produced ECDSA signature (signature created using ECC)[15]. This is signature that the Relying Party receives from the Application Provided when the transaction completes.

The signature is DER-encoded. It consists of an object with an application class tag, containing a `SEQUENCE` of two values: components r and s of the ECDSA signature, encoded as two `INTEGER`s.

We conducted an experiment to determine if the Application Provider checks that this field contains a valid signature of the hash that has been

[15]To the best of our knowledge, RSA as a signature algorithm for Mobile-ID is not used today.

requested to be signed. In this experiment, a Mobile-ID transaction was conducted normally, but one bit inside one of the integers of the outgoing Mobile-ID SMS was flipped. The Application Provider returned to the Relying Party the `SIGNATURE_HASH_MISMATCH` error status. This means that the Application Provider indeed verifies the correctness of the signature before forwarding it to the Relying Party.

We also conducted multiple experiments to determine the format validation checks implemented by the server that processes the DER-encoded signature. In each experiment, we conducted a Mobile-ID transaction normally, but changed the DER-encoded object that is sent from the phone in an outgoing Mobile-ID SMS message. In some of the experiments, where the DER-encoded object was tampered with, the Mobile-ID completed successfully. In these experiments

- The tag (the first byte) in the `outer-TL` field was changed to an invalid value (`00`)
- The length (the second byte) in the `outer-TL` field was changed to an invalid value (`01`)
- An extra DER-encoded object was added inside the `SEQUENCE`, after the two `INTEGER`s. In one experiment the object added was an `INTEGER` with value 0, in another experiment it was a `NULL` object. In both of the experiments, the bytes that encode lengths were adjusted appropriately, so the DER-encoded object would remain valid.

In anorther experiment, we replaced values of both of the integers with `7f`. The transaction concluded: the Application Provider returned to the Relying Party the `SIGNATURE_HASH_MISMATCH` error status. In some of the experiments, the outgoing Mobile-ID SMS messages were discarded. I.e., after the message was sent, the transaction did not complete until the timeout. In these experiments:

- The tag in the field `seq-TL` field was changed to an invalid value (`00`)
- The length in the `outer-TL` field was changed to an invalid value (`00`)

- An extra DER-encoded `NULL` object was added *before* the two `INTEGER`s encoding the signature

The experiments described above allow to speculate that the OTA server performs preliminary validation of the DER-encoded object and, if the validation check fails, does not notify the Application Provider that an outgoing Mobile-ID message for the transaction was received. Otherwise it sends the signature to the Application Provider, that the signature itself is valid.

We conducted an experiment to learn whether the transaction would complete (before the timeout) even if the source address of the Mobile-ID SMS message with the signature is of a different cellular operator subscriber than to whom the incoming MID SMS was sent. We learned that in this case message is discarded — i.e., the transaction is not completed. The message is also discarded if outgoing Mobile-ID SMS message is sent to the wrong OTA address (e.g., a message is sent from an Elisa EE operated SIM card to the address of OTA server of Tele2).

6 Security implications

During the course of this research, multiple security issues of Mobile-ID became evident.

Possibility of a backdoor in the SIM card. As discussed in Chapter 4, in Mobile-ID transactions, it is the Mobile-ID applet that sends commands for the phone to perform, e.g., displaying a prompt. The applet is closed source; there is next to no outside oversight for the process of installing the applets to the SIM cards[16]. This *allows* the cellular network providers to distribute SIM cards with undocumented features, e.g., (1) silently extracting private keys or (2) silently obtaining a signature in the name of the subscriber. Such functionality *may* be introduced either intentionally, or arise as a result of a programming mistake. Introducing or using this functionality, in all likelihood, would not get noticed by any party that might raise the alarm.

Relaxed keypair generation requirements. The WPKI forum recommendations document [12] describes that the private keys must be destroyed from the SIM card manufacturer's equipment after the SIM card personalization process completes (see PR2 in Section 5.3). We believe that imposing this requirement is not sufficient to ensure that the private keys are not being copied. We believe that there should be a requirement that the keypairs must be generated inside the SIM cards, such that they never leave the card

Possibility of an attack by the OTA server. It is not clear how exactly the content of the incoming Mobile-ID SMS is encrypted. It seems that there is no mechanism in place to prevent the OTA server from replacing the hash of the transaction[17]. If there is indeed no hash protection (as

[16]The information on where exactly the SIM personalization process is performed is not documented in publicly accessible sources.

[17]This is hinted in WPKI forum recommendations document [12] (see TI1 in Section 9). It is mentioned that the OTA server must accept the hash (and other information for conducting the transaction) from the TSP platform. However, it is not mentioned that the hash (or the whole request) is encrypted for the SIM card or that the TSP platform includes in the request a cryptographic checksum that should be verified by the SIM card.

described above), then the OTA server can replace the hash of a legitimate transaction with a hash value of a document, they want to forge a signature on. The attack will raise no issue, as the forged document can be generated to have the same control code as the legitimate transaction.

An attack involving a malicious thin SIM card. We found an issue regarding how Mobile-ID PIN and PUK codes are exchanged between the SIM card and the phone. According to the STK standard [3], there is no mutual authentication between the phone. The SIM card and so the Mobile-ID codes are transmitted in clear text. The lack of mutual authentication makes the Mobile-ID vulnerable to attacks based on using a malicious thin SIM card [22]. In this attack, a very thin device is installed between the phone and the SIM card, without the knowledge of the SIM card holder. This device relays the APDUs through itself and monitors them. When the thin SIM card relays Mobile-ID PIN and PUK codes, it saves copies of them. Afterwards, the thin SIM card may interfere with the communication between the phone and the SIM card, to silently confirm transactions. E.g., in may silently confirm all the transactions, or (a less suspicious option) all the transactions occurring at a specific time (when the user is not likely to use Mobile-ID), or all the transactions with a specific display text. Alternatively, the thin SIM card may send saved codes in a service message to an attacker's device.

We discovered that in Mobile-ID, the cellular network providers are trusted more than anticipated. The solution leaves them several opportunities for abuse. Moreover, there is always a risk of compromise, which applies to any type of technology regardless of how secure it is. The security risks described in this chapter, related to the role of cellular network providers in Mobile-ID, are aggravated by the fact that there are multiple cellular network providers that support Mobile-ID.

7 Conclusion

In this work, the communication that facilitates Mobile-ID interactions was researched and documented: including the STK command exchange and formats of the SMS messages used for transactions. Mobile-ID transaction participants and their roles were established. Additionally, several security weaknesses of Mobile-ID solution were described.

This work was conducted despite the lack of public information on how the Mobile-ID transactions are conducted. Due to the lack of information, it was not possible to establish the exact formatting, and the fields included in the Secured Data part of the incoming Mobile-ID SMS. It was also impossible to document the communication between the Trusted Service Provider platform and OTA servers. Thus, these aspects of Mobile-ID were not documented: this is the main limitation of this work. We believe that SK ID Solutions could make a valuable and welcomed contribution to the field by releasing some of the internal documentation on Mobile-ID, or by sharing some insights on the inner working of Mobile-ID with the interested parties.

A higher level of transparency is vital to ensure the security of electronic identity solutions. We believe that the Estonian state should step up to demand a higher level of transparency for eID solutions on which the well being of our e-society relies, despite the additional expenses and effort required.

In future work, a comparative analysis of the implementation peculiarities of Mobile-ID-enabled SIM cards, issued by different cellular providers may be conducted. Also, an analysis of public keys used in the Mobile-ID solution could be performed to potentially learn the algorithms used for generating Mobile-ID private keys.

Acknowledgments This research has been carried out with financial support from the Estonian Ministry of Economic Affairs and Communications.

References

[1] Estonian Information System Authority. RIA has concluded a Mobile-ID contract for five years, May 2022. https://www.ria.ee/en/news/ria-has-concluded-mobile-id-contract-five-years.html.

[2] SK ID Solutions AS. Mobile ID (MID) REST API, January 2021. https://github.com/SK-EID/MID/tree/2e2415b582a73deaf0586eaf887437a073d91aad.

[3] European Telecommunications Standards Institute. Digital cellular telecommunications system (Phase 2+); Specification of the SIM Application Toolkit (SAT) for the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface (3GPP TS 11.14 version 8.18.0 Release 1999), June 2007. https://www.etsi.org/deliver/etsi_ts/101200_101299/101267/08.18.00_60/ts_101267v081800p.pdf.

[4] Osmocom community. Osmocom SIMtrace 2, March 2022. https://osmocom.org/projects/simtrace2.

[5] Wireshark contributors. Wireshark. https://www.wireshark.org.

[6] OsmocomBB community. OsmocomBB wiki, March 2020. https://osmocom.org/projects/baseband/wiki.

[7] OsmocomBB community. OsmocomBB softSIM, January 2019. https://osmocom.org/projects/baseband/wiki/SoftSIM.

[8] Application of mobile-ID. https://www.id.ee/en/article/application-of-mobile-id/.

[9] Cybernetica AS. The History of Digital Identity in Estonia, February 2020. https://cyber.ee/resources/news/the-history-of-digital-identity-in-estonia/.

[10] Baltic WPKI Forum. Baltic WPKI Forum website, February 2014. https://web.archive.org/web/20150828182742/http://wpki.eu/doku/doku.php.

[11] Delfi.ee. Peagi saab digiallkirja kasutada üle Baltikumi, March 2007. https://arileht.delfi.ee/artikkel/51081418/peagi-saab-digiallkirja-kasutada-ule-baltikumi.

[12] Jürgen Niinre and Ramūnas Šablinskas. WPKI mobile transactions: implementation recommendations PUBLIC VERSION 0.3, August 2007. https://cybersec.ee/storage/Baltic_WPKI_standard_draft-0.3.pdf.

[13] Remarc. Mobile-ID USAT applet, January 2017. https://sourceforge.net/projects/mobile-id-usat-applet.

[14] Thales. OTA (Over The Air). https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/ota.

[15] 3GPP. 3GPP TS 03.48 V8.9.0 (2005-06) 3rd Generation Partnership Project; Technical Specification Group Terminals; Security mechanisms for the SIM application toolkit; Stage 2 (Release 1999), June 2005. https://www.3gpp.org/ftp/Specs/archive/03_series/03.48/0348-890.zip.

[16] 3GPP. 3GPP TS 23.040 V17.2.0 (2022-03) 3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Technical realization of the Short Message Service (SMS) (Release 17), March 2022. https://www.3gpp.org/ftp/Specs/archive/23_series/23.040/23040-h20.zip.

[17] European Telecommunications Standards Institute. ETSI TS 102 221 V17.0.0 (2021-10) Smart Cards; UICC-Terminal interface; Physical

and logical characteristics (Release 17), October 2021. https://www.etsi.org/deliver/etsi_ts/102200_102299/102221/17.00.00_60/ts_102221v170000p.pdf.

[18] 3GPP. 3GPP TS 11.11 V8.14.0 (2007-06) 3rd Generation Partnership Project; Technical Specification Group Terminals Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface (Release 1999), June 2007. https://www.3gpp.org/ftp/Specs/archive/11_series/11.11/1111-8e0.zip.

[19] ActiveXperts Software BV. SMS Messaging. https://www.activexperts.com/serial-port-component/tutorials/smstechnical/.

[20] 3GPP. 3rd Generation Partnership Project; Technical Specification Group Terminals; Technical realization of the Short Message Service (SMS) (Release 1998), December 2001. https://www.3gpp.org/ftp/Specs/archive/03_series/03.40/0340-750.zip.

[21] European Telecommunications Standards Institute. Digital cellular telecommunications system (Phase 2+); Alphabets and language-specific information (GSM 03.38) , July 1996. https://www.etsi.org/deliver/etsi_gts/03/0338/05.03.00_60/gsmts_0338v050300p.pdf.

[22] Rowan Phipps, Shrirang Mare, Peter Ney, Jennifer Rose Webster, and Kurtis Heimerl. Thin Sim-Based Mobile Money Attacks, 2018. https://ictd.cs.washington.edu/docs/talks/2018/rowan_compass_2018.pdf.

Appendices

A Code snippets

```
diff --git a/host/src/simtrace2-cardem-pcsc.c b/host/src/simtrace2-cardem-pcsc.c
index 8b08f36..00e7909 100644
--- a/host/src/simtrace2-cardem-pcsc.c
+++ b/host/src/simtrace2-cardem-pcsc.c
@@ -133,6 +133,17 @@ static int process_do_rx_da(struct osmo_st2_cardem_inst *ci,
     uint8_t *buf, int l

         data = (struct cardemu_usb_msg_rx_data *) buf;

+       // In the context of this research, in may be assumed that
+       // if APDU is sufficiently long, it contains Mobile-ID incoming SMS.
+       if (data->data_len > 100) {
+               printf("\nLong APDU captured: \n");
+               printf("%s", osmo_hexdump(data->data, data->data_len));
+               printf("\n\n");
+               fflush(stdin);
+
+               return 0;
+       }
+
         printf("=> DATA: flags=%x, %s: ", data->flags,
                 osmo_hexdump(data->data, data->data_len));
```

Listing 1. Modification for `simtrace2-cardem-pcsc` (only the relevant part)

```
# File: phone_simulator.py

from smartcard.CardType import AnyCardType  # tested with pyscard 2.0.2
from smartcard.CardRequest import CardRequest
from smartcard.CardConnection import CardConnection
import socket

if __name__ == '__main__':
    always_proactive = False
    c = CardRequest(timeout=100, cardType=AnyCardType()).waitforcard().connection
    c.connect(CardConnection.T0_protocol)
    print("[+] Selected reader:", c.getReader())
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # Internet, UDP


def apdu_to_array(apdu):
    if isinstance(apdu, str):
        for c in ' <>:-\n':
            apdu = apdu.replace(c, '')
        apdu = bytes.fromhex(apdu)
    if isinstance(apdu, bytes):
        apdu = [int(x) for x in apdu]
    return apdu


def apdu_to_string(apdu):
    if not isinstance(apdu, list):
```

```
        apdu = apdu_to_array(apdu)
    return ' '.join([bytes([x]).hex() for x in apdu])


# GSM_SIM tap, for Wireshark
# Filter: (gsm_sim or gsm_sms) and not icmp
def gsm_tap(apdu, ret=None):
    apdu = apdu_to_array(apdu)
    if ret is None:
        ret = [0, 0]
    gsm_tap_magic = b'\x02\x02\x04' + b'\x00' * 5
    sock.sendto(gsm_tap_magic + bytes(apdu) + bytes(ret), ("127.0.0.1", 4729))


def cmd(apdu):
    apdu = apdu_to_array(apdu)
    ret, sw0, sw1 = c.transmit(apdu)
    ret += [sw0, sw1]
    gsm_tap(apdu, ret)
    return bytes(ret)


# Executes proactive commands while there are proactive commands
# Execution may be disabled by user input
# ! Please replace the hardcoded values if you decide to expand this code !
def proactive_sim(got):
    global always_proactive
    while True:
        if got[-2] != 0x91:
            break  # SIM does not want to be proactive
        length = got[-1]

        if not always_proactive:
            ap_input = input('proactive?: ')
            if ap_input == '':
                always_proactive = True
            if ap_input in ['yes', 'y', 'ok', '']:
                pass  # read and execute the proactive command
            else:
                return  # do not read+execute the proactive command

        if isinstance(length, bytes):
            length = length[0]
        r = cmd('80 12 00 00 ' + bytes([length]).hex())  # FETCH

        assert r[0] == 0xd0  # proactive command

        ac = -1  # Removing extra length
        for i in range(0, 7):
            if r[i] == 0x81 and r[i + 1] == 0x03:
                ac = i
        assert ac != -1
        r = r[ac:]

        cmd_str = None
        if r[3] == 0x26:  # PROVIDE LOCAL INFORMATION
            # Watch out for out of border
            if r[4] == 0x04:  # Language setting
```

```
                en = '656e'
                cmd_str = f'8014000010810301260402028281830100ad02{en}'
            if r[3] == 0x05 and r[11] == 0x07:  # Language selection
                cmd_str = f'801400000c8103010500020281830100'
            if r[3] == 0x25:  # SET MENU
                cmd_str = f'801400000c8103012500020281830100'
            if r[3] == 0x20:  # PLAY TONE
                print('<RING-RING>')
                cmd_str = f'801400000c8103012000020281830100'
            if r[3] == 0x21 or r[6] == 0x21:  # DISPLAY TEXT
                format = 'ucs2' if r[11] == 0x08 else 'gsm7'
                s = ''
                for i in range(13 if format == 'ucs2' else 12, 300, 2 if format == 'ucs2'
                    else 1):
                    if i < len(r) - 2:
                        cc = chr(r[i])
                        if cc == '\r':
                            cc = '\\r'
                        s += cc
                print(f'===== ===== =====\n{s}\n----- ----- -----')
                cmd_str = f'801400000c8103012181020281830100'
            if r[3] == 0x23:  # GET INPUT
                for i in range(len(r) - 2, -1, -1):
                    if r[i] == 0x04:
                        ac = i
                        break
                len_text = r[ac - 1]
                print(r[ac + 1:len_text + ac].decode())
                response = input().encode()
                format = '04'
                text = '8d' + bytes([len(response) + 1]).hex() + format + response.hex()
                cmd_str = f'8014000014810301230402028281830100{text}'
            if r[3] == 0x13:  # SEND SHORT MESSAGE
                print('Sending SMS...')
                print(''.join([bytes([x]).hex() for x in r]))
                aa = -1
                for a in range(len(r) - 1):
                    if r[a] == 0x0 and r[a + 1] == 0xf6:
                        aa = a + 3
                if aa != -1:
                    print(''.join([bytes([x]).hex() for x in r[aa:-2]]))
                cmd_str = f'801400000c8103011300020281830100'
            if cmd_str is None:
                print("Error! Could not process this proactive command")
                break
            got = cmd(cmd_str)


    def get_iccid():
        got = cmd('00 a4 08 04 04 7f ff 6f 07')
        if got == b'\x91\x19':
            got = cmd('00 c0 00 00 19')
        print(got)


    def get_timestamp():
        import time
        t = time.strftime("%Y;%m;%d;%H;%M;%S").split(';')
```

```
    ts = f'{t[0][2:][::-1]} {t[1][::-1]} {t[2][::-1]} ' \
         f'{t[3][::-1]} {t[4][::-1]} {t[5][::-1]} ' \
         f'80'
    return ts


# Supports multiple possible truncations of captured SMS, will prepend necessary parts
    to formulate the apdu
def envelope_from_capture(apdu, send=True):
    def length(apdu):
        ldata = len(apdu.replace(' ', '')) // 2
        if 0 <= ldata <= 127:
            return f'{bytes([ldata]).hex()}'
        else:
            return f'81 {bytes([ldata]).hex()}'

    print('sms_from_capture, ', end='')
    apdu = apdu_to_string(apdu)

    # Add user data header
    if ' '.join(apdu.split(' ')[3:10]) == '16 00 15 15 50 4b 49':
        apdu = '02 70 00 ' + apdu

    # Transform to SMS-DELIVER value
    if apdu.startswith('02 70 00'):
        originating_address = '05 a1 59 45 f4'
        timestamp = get_timestamp()
        user_data_length = bytes([len(apdu.replace(' ', '')) // 2]).hex()
        tp_pid = '7f'
        tp_dcs = 'f6'
        apdu = f'64 {originating_address} {tp_pid} {tp_dcs} {timestamp} {
            user_data_length} {apdu}'

    # Transform to SMS-DELIVER TLV
    if apdu.startswith('64'):
        apdu = f'8b {length(apdu)} {apdu}'

    # Transform to SMS-PP Download TLV
    if apdu.startswith('8b'):
        identities = '82 02 83 81'
        address = '06 07 91 73 52 16 00 20 f0'  # Originator: SMSC
        apdu = f'{identities} {address} {apdu}'
        apdu = f"d1 {length(apdu)} {apdu}"

    # Add ENVELOPE command
    if apdu.startswith('d1'):
        apdu = f"80 c2 00 00 {bytes([len(apdu.replace(' ', '')) // 2]).hex()} {apdu}"

    # Send
    if send:
        got = cmd(apdu)
        assert got == b'\x90\x00' or got[0] == 0x91 or got == b'b\x00'
        print(got.hex())
        proactive_sim(got)
    else:
        return apdu
```

```
if __name__ == '__main__':
    # TERMINAL PROFILE
    cmd('8010000012ffffffff7f9300dfff00000000050a060080')

    # SELECT APPLICATION
    got = cmd('00a4040c0ca0000000871002f3728f0589')  # Only tested with Elisa SIM cards
    proactive_sim(got)

    apdus = [x for x in open('apdus', 'r').read().split('\n') if len(x) > 1]

    envelope_from_capture(apdus[0])
```

Listing 2. Python 3 script for communicating with SIM card

```
# File: outgoing_modify.py

from phone_simulator import apdu_to_array as to_arr
from phone_simulator import apdu_to_string as to_str


def deserialize_contents(apdu):
    apdu = to_arr(apdu)
    obj = {}
    obj['header'] = to_str(apdu[0:1])
    obj['type'] = to_str(apdu[1:2])
    obj['hex-identifier'] = to_str(apdu[2:10])
    apdu = apdu[10:]
    if obj['type'] != '00':
        assert len(apdu) == 0
        return obj
    else:
        assert len(apdu) > 20

    for ns in ['outer', 'seq']:
        obj[ns + '-T'] = to_str(apdu[0:1])
        assert obj[ns + '-T'] == '42' if ns == 'outer' else obj[ns + '-T'] == '30'
        assert 2 + int(to_str(apdu[1:2]), 16) == len(apdu)
        apdu = apdu[2:]

    for ns in ['int-a', 'int-b']:
        obj[ns + '-T'] = to_str(apdu[0:1])
        assert obj[ns + '-T'] == '02'
        value_l = int(to_str(apdu[1:2]), 16)
        obj[ns + '-V'] = to_str(apdu[2:2+value_l])
        apdu = apdu[2+value_l:]

    assert len(apdu) == 0
    return obj


def serialize_contents(obj):
    apdu = f"{obj['header']} {obj['type']} {obj['hex-identifier']}"
    if obj['type'] != '00':
        return apdu.replace(' ', '')

    def tlv(name):
        assert name + '-T' in obj
        assert name + '-V' in obj
```

```
        nameL = to_str([len(to_arr(obj[name + '-V']))])
        if name + '-L' in obj:
            nameL = obj[name + '-L']
        return f"{obj[name + '-T']} {nameL} {obj[name + '-V']}"

    if not 'seq' + '-V' in obj:
        obj['seq' + '-V'] = f"{tlv('int-a')} {tlv('int-b')}"
    if not 'outer' + '-V' in obj:
        obj['outer' + '-V'] = f"{tlv('seq')}"
    return f"{apdu} {tlv('outer')}".replace(' ', '')


# Some tests to ensure that the script works as intended
def selftest():
    apdu = 'f100 3334636430393832 42483046' \
           '022100 a64fa8883b416cb304a51d13df5003fb8d3edae3e1971694b24bba91cfe88bfd' \
           '022100 947c8bb1e9c76a62057e94a5d4d03e78193402406e1acd23ff1e05d5b962e977'
    assert serialize_contents(deserialize_contents(apdu)) == to_str(apdu).replace(' ',
        '')
    t = deserialize_contents(apdu)
    t['int-a-T'] = 'de'; t['int-a-L'] = 'ad'; t['int-a-V'] = 'be ef'
    assert 'deadbeef' in serialize_contents(t)
    t = deserialize_contents(apdu)
    t['outer-V'] = 'deadbeef'
    assert 'deadbeef' in serialize_contents(t)
selftest()


if __name__ == '__main__':
    apdu = [x for x in open('resp', 'r').read().split('\n') if len(x) > 1]

    obj1 = deserialize_contents(apdu[0])

    # Changes can be put here
    obj1['outer-T'] = '00'

    s1 = serialize_contents(obj1)
    print(s1)
    print(f"smsmid 1 95544 {s1}")
```

Listing 3. Python 3 script for modifying the content of Mobile-ID SMS sent to the network

B Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Semjon Kravtšenko**,
*(*author's name)

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

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

   The Estonian Mobile-ID Implementation on the SIM Card,
   *(*title of thesis)

   supervised by Arnis Paršovs.
   *(*supervisor's name)

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

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

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

Semjon Kravtšenko
2022-05-10