Bachelor Thesis

SIM Toolkit In Practice

August 27, 2012

Author: Sjors Gielen
s.gielen@student.ru.nl
Supervisors: Erik Poll
Fabian van den Broek
{erikpoll,f.vandenbroek} @ cs.ru.nl

Abstract

This paper gives an overview of the SIM Toolkit standard (STK), used in mobile phones in the global system for mobile communication (GSM) to provide value-added services for the user and the operator. The standard is used in practice on many SIM cards today.

I will describe a short background of smart card, SIM card and SIM Toolkit communication, and show how to do a man-in-the-middle on the protocol to read and modify the traffic between the SIM and the phone. I will also give some examples of real-life SIM Toolkit traffic and its results on various phones. Finally, I will describe some possible vulnerabilities of the SIM Toolkit standard and compliant phones, tested in practice.

This bachelor thesis was written within the Honours Programme of the Faculty of Science at the Radboud University in Nijmegen, The Netherlands.

1 Introduction

When the first specifications for the Global System for Mobile Communications (GSM) were finalized in 1990, a SIM card had only two purposes: it stored information on what GSM network to connect to, and it could run a cryptographic algorithm using a secret key to uniquely identify itself to that network [4].

However, as the GSM network grew and mobile phone manufactorers and service providers started to make more money off value added services, the operators that usually only gave out the SIM card to customers needed an extra facility to provide extra services to their customers. This is why in 1996 the SIM Toolkit standard was released, giving operators the possibility to let special applications on the SIM card proactively initiate communication with the users and the network [6]. Since then, the standard went through many iterations and is now called the *Card Application Toolkit*, meant for any type of smart card [8] (but still usually called SIM Toolkit when used in SIM card contexts).

As usual in technical documents specifying communication methods used internationally all over the world, the SIM Toolkit standards consist of thousands of pages of complex documentation with numerous versions over the years, references all over the place and no clear practical summary, and are written by hundreds of different people. This makes it difficult for new developers to start using the protocol, regardless of whether they work on the SIM card or on the phone side.

In this bachelor thesis, I will describe the ways SIM Toolkit can be used in practice, with examples of real-world commands and their results. I will show some methods for analyzing and modifying real SIM Toolkit traffic. After this, I will give some examples of value-added services that can be implemented using SIM Toolkit, and show some of the security risks that SIM Toolkit introduces for end users.

I will start off with an overview of the background of this thesis: SIM cards in section 2, and SIM Toolkit itself in section 3. In section 4.1, I will describe how the existing RebelSIM hardware can quickly be set up to listen in to existing SIM and SIM Toolkit traffic. Using this, existing SIM Toolkit functionality can be examined. In section 4.3 I will show how the SmartLogicTool [10] can support SIM traffic and be used to do an active man-in-the-middle on the protocol, with the ability to change the traffic as it goes over the line. Finally in section 4.4 I will show how the Bladox TurboSIM, a thin SIM-sized chip that can modify the traffic between the phone and the actual SIM card, can be used to deploy new SIM Toolkit services without the need to modify the actual SIM software.

In section 5, I will give a list of some SIM Toolkit commands that the SIM can send proactively to the phone, including several examples. In section 6, I will describe some real-world risks imposed by the SIM Toolkit standard. Finally, I will conclude this thesis in section 7.

Contents

2 SIM card background

In this section, I will describe some of the essentials behind smartcard and SIM cards, their history, and the way they communicate. This section is intended for readers who are not familiar with smart card communication or the specifics of SIM cards and gives an introduction in order to understand the rest of this paper. A complete description of the protocols is outside the scope of this paper but can be found, for example, in the book *Smart card, tokens, security and applications* by Mayes and Markantonakis [12].

2.1 Smart cards

Smart cards, formally called *integrated circuit cards*, are thin, usually credit-card sized cards with a microprocessor embedded within them. Their purpose is mostly to provide some kind of data storage, usually protected by an authentication and access control mechanism. Since their invention in 1977 by Michel Ugon, their use has grown to financial debit and credit cards, car fuel cards, authorization cards for pay television, cryptographical certificate or encryption key storing, physical company access, single sign-on and most notably for this paper: SIM cards.

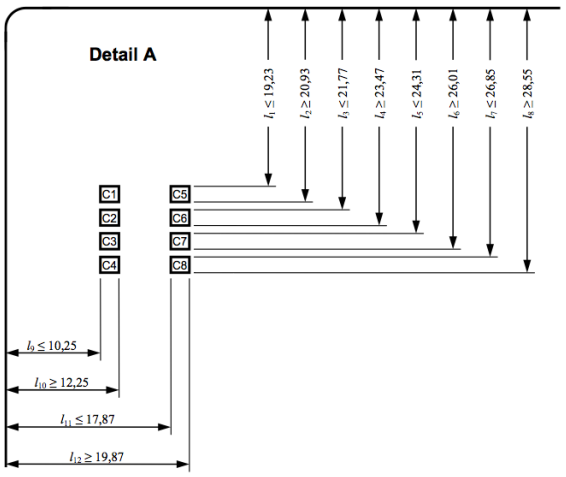


Figure 1: The connection pad design of a smart card. As can be seen, the sizes are strictly defined in order to achieve interoperability. From ISO 7816-2 [11]

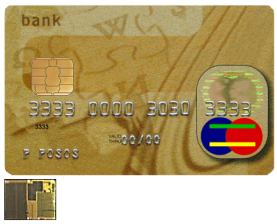


Figure 2: A financial smart card. The distinctive connection pads are seen to the left of the image. From Wikipedia

The physical design and inner workings of smart cards is specified in the ISO7816 standard [11]. In particular, the location of the connection pads is tightly defined, as can be seen in figure 1: all cards adhering to the standard should fit and work in all terminals. In figure 2, a credit card with smart card functionalities can be seen.

2.1.1 Smart card communication

By inserting a smart card in a terminal, the metal strips on the connector are electronically connected to the pins inside the reader. Most of the pins are used for hardware communication; for example, one of the pins is used for applying working voltage, another for hardware resets. The most important pin for communication is the I/O pin. This pin can be set high (H) or low (L), and by reading or changing the state of the pin at specific timings, bytes are transmitted [11]. The timings depend on the state of the clock pin and

Figure 3: The pin-out scheme of a smart card. From Wikipedia/File:SmartCardPinout.svg

CLA	INS	P1	P2	LEN
A0	A4	00	00	02

Table 1: A command APDU describing the GSM SELECT command, indicating 2 bytes of extra payload data. (The payload is not shown here.)

	CLA	INS	P1	P2	LEN	Data
Command APDU	A0	A4	00	00	02	7F 20

	SW1	SW2
Response APDU	90	00

Table 2: A GSM SELECT command APDU including its response, formulated using the APDU model.

the speed of communication that the card and the terminal agreed on, either as a default speed or after a speed enhancement procedure.

When a smart card is inserted into a terminal, the terminal will start applying a voltage to the voltage pin, a clock rate to the clock pin, and issues a hardware reset. The smart card is now powered and has a sense of timing using the clock rate, and will initiate the protocol with the *Answer to Reset* (or ATR). This message, consisting of several bytes, contains the capabilities and other properties of the smart card. For example, it will tell the terminal what kind of smart card this is.

After this, the card is activated and the terminal and card can start exchanging commands. In smart card communication, commands are always initiated by the terminal by sending a *Command APDU* (application protocol data unit). An example command APDU is shown in table 1: the first byte is the *class byte* and is used to identify the global command type, in this case `A0` which means it is a GSM command. The second byte is the *instruction byte* and gives the actual command within the global type, in this case `A4` means this is the `GSM SELECT` command. Its two parameters are `00` (because they are unused), and its length byte parameter is set to `02`, meaning there are two bytes of extra data that will follow.

The smart card now responds with the instruction byte `A4`, after which additional data of exactly `LEN` bytes is sent by either the terminal or the card, depending on the command. Finally, the card responds with two *status word* bytes; the most common value is `90 00` which means the command simply succeeded.

In a higher level, this protocol is often simplified to form the *APDU model*: the repeating of the `INS` byte by the card is simply ignored, and the data is appended to the command or prepended to the response depending on whether the terminal or the card sent the data. Therefore, in a higher level model describing the `GSM SELECT` command, one may actually see a model as in table 2, where one should remember this is not exactly how the bytes were transferred. When the smart card sends some data instead of the terminal, the APDUs are written as in table 3.

	CLA	INS	P1	P2	LEN
Command APDU	A0	B0	00	00	01

	Data	SW1	SW2
Response APDU	03	90	00

Table 3: A GSM READ BINARY command and response APDU. Here, the SIM card sent extra data, instead of the terminal as with GSM SELECT.

2.2 SIM cards

A SIM card is a special type of smart card which is used for connecting to and using the mobile telephony network known as the GSM network (*Global System for Mobile Communication*). Highly standardised, the card contains the unique subscriber number (*IMSI* for *International Mobile Subscriber Identity*), the secret authentication key and other properties used to authenticate to the network and uniquely identify the card's owner.

The first official SIM card specification was written in january 1990 as ETSI Technical Specification 11.11 [1]. This first specification is not available anymore; the oldest available version defines a *Phase 1* SIM card as follows [4, page 2]:

> Recommendation GSM 02.17 states the basic concept of a split of the MS into a removable SIM (Subscriber Identity Module) which contains all network related subscriber information and a ME (Mobile Equipment) which is the remaining part of the MS, and realizes all the functions common to all GSM subscribers.

In other words: in GSM, the phone or *Mobile Station* is split up into a SIM card with subscriber-specific information, and *Mobile Equipment*, the physical hardware of the phone, which is common for every subscriber.

The SIM is subsequently given only two "fundamental purposes" [4, page 5]:

> - storing data (and controlling the access to this data),
>
> - executing algorithms in secure conditions: for its maintask of authentication of the subscriber's identity according to Recommendation GSM 02.09.

Even today, these purposes have stayed largely the same. New functions and standardised data storages have been added in newer specifications such as secure channels and SIM applications [9]. The largest feature added was the *SIM Toolkit,* which is described in the rest of this thesis.

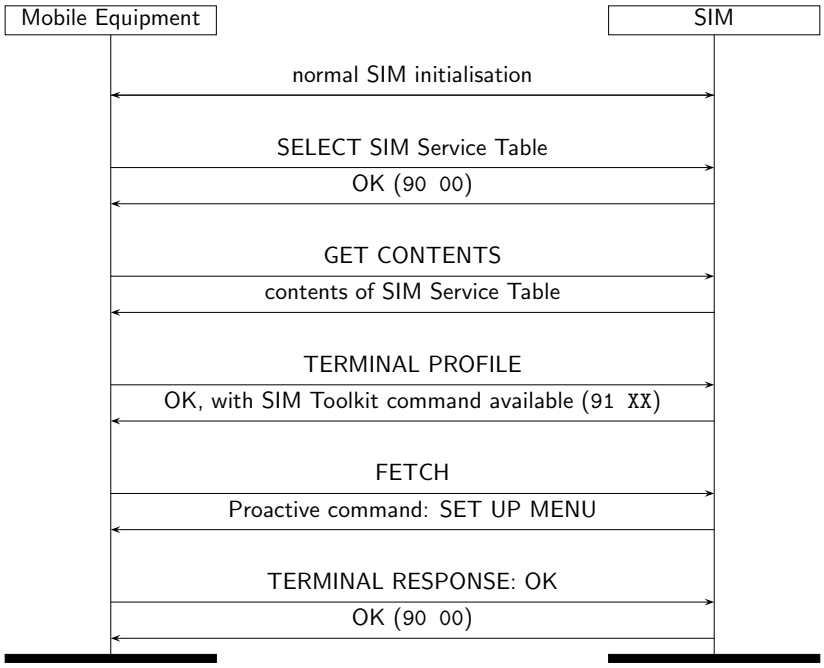


Figure 4: An overview of the SIM Toolkit initialisation. (Recall that `90 00` is the normal smart card "OK" response.)

3 SIM Toolkit

In the previous section, I have described the background of SIM cards and a bit about their communication using APDUs. This communication allows the phone to couple with the SIM card and use the data provided by the SIM card to use the network correctly.

However, the standard SIM application does not allow for much flexibility in value-added services from the network operator in the way Java applications could, early on in mobile phone development. Operators wanted to add more services to the SIM card to keep an innovative edge on other providers. The SIM Toolkit standard was created for this purpose, by allowing the SIM to run its own applications that could communicate with the user and the network. Also, these SIM applications gained the possibility to send *proactive* commands to the mobile equipment. Recall from the background section that normally, a smartcard simply receives a command, executes it, and responds with the results. The SIM Toolkit standard gives them the possibility to reply with commands *to be executed by the terminal*. For example, it could take temporary control to display a message on the screen or send an SMS message [12, chapter 4.7].

The standard in which SIM Toolkit was originally introduced was ETSI TS 11.14 [7]. This standard defines several new APDUs that are used solely for SIM Toolkit: `TERMINAL PROFILE`, `ENVELOPE`, `FETCH` and `TERMINAL RESPONSE`. It also defines a *SIM Service Table*, a file stored on the SIM card that tells the mobile equipment what services the SIM supports. A typical SIM Toolkit initialisation procedure can be seen in figure 4.

3.1 SIM Toolkit initialisation

When the mobile equipment is turned on, it will start to initialise the SIM card to be able to connect to the mobile network. As soon as the normal initialisation of the SIM card is done, the SIM Toolkit stack is initialised. First, the phone queries the *SIM service table*, a regular *Elementary File* (`6F 38`) on the SIM card. This file contains the various services the SIM can provide, and whether they should be used by the mobile equipment. An example of a SIM service table reading is seen below. First the phone selects the file, and then the contents are read.

```
ME: 00 A4 00 04 02
SIM: A4
ME: 6F 38
SIM: 61 1E
ME: 00 B0 00 00 07
SIM: B0 9E EF 1F 1C FF 3E 04 90 00
```

Because this is incomprehensible to human beings, I have created the SIMparser.pl tool to explain the data as it is read. I will explain this tool in section 4.2. It has the following to say about the above conversation: (From now on, I will only give SIMparser.pl tool output.)

```
      COMMAND: 00 A4 00 04 02
               SELECT file [P1=04?]
 COMMAND DATA: 6F 38
               elementary file under dedicated file; SST (SIM service table)
     RESPONSE: 61 1E
               SUCCESS, 30 extra response bytes, use GET RESPONSE

      COMMAND: 00 B0 00 00 07
               READ BINARY
     RESPONSE: 90 00
               SUCCESS
RESPONSE DATA: 9E EF 1F 1C FF 3E 04
               == Contents of file 6F 38 ==
               Service 1: CHV1 disable function (not allocated, activated)
               Service 2: Abbreviated Dialling Numbers (allocated, activated)
               Service 3: Fixed Dialling Numbers (allocated, not activated)
               Service 4: Short Message Storage (not allocated, activated)
               Service 5: Advice of Charge (allocated, activated)
               Service 6: Capability Configuration Parameters (allocated, activated)
               Service 7: PLMN selector (not allocated, activated)
               Service 8: (RFU) (allocated, activated)
               Service 9: MSISDN (allocated, activated)
               Service 10: Extension1 (allocated, activated)
               Service 11: Extension2 (allocated, not activated)
               Service 12: SMS Parameters (not allocated, not activated)
               Service 13: Last Number Dialled (not allocated, not activated)
               Service 14: Cell Broadcast Message Identifier (allocated, activated)
               Service 15: Group Identifier Level 1 (allocated, not activated)
               Service 16: Group Identifier Level 2 (not allocated, not activated)
               Service 17: Service Provider Name (allocated, activated)
               Service 18: Service Dialling Numbers (allocated, activated)
               Service 19: Extension3 (allocated, activated)
               Service 20: (RFU) (allocated, activated)
               Service 21: VCGS Group Identifier List (not allocated, activated)
               Service 22: VBS Group Identifier List (allocated, activated)
               Service 23: enhanced Multi-Level Precedence and Pre-emption
                           Service (allocated, activated)
               Service 24: Automatic Answer for eMLPP (not allocated, not activated)
               Service 25: Data download via SMS-CB (not allocated, not activated)
               Service 26: Data download via SMS-PP (allocated, not activated)
               Service 27: Menu selection (not allocated, not activated)
               Service 28: Call control (not allocated, not activated)
```

The SIM service table shows the services understood by the SIM card. The standard requires a SIM to declare support for some services in its service table for the mobile equipment to be allowed to use them. For example, the ME is not allowed to send data from SMS cell broadcasts to the SIM (service 25), as it is not allocated. Every service is coded with two bits in the contents, so the 7 bytes listed in `RESPONSE DATA` constitute the 28 services seen above [5, 10.2.7, page 56].

Here, a fundamental difference between theory and practice is seen. The SIM card announces its SIM service table is only 7 bytes long, so the Proactive SIM service (service 29 [5, 10.2.7, page 56]) is not listed. This means the service must be regarded as being not allocated, and shall not be used by the ME – yet as we will clearly see, the ME and SIM will both use it extensively. This is also true for the "menu selection" service, which is not allocated or activated in the SIM card; yet, the SIM card will soon set up this "menu selection" service itself, as we will see in a moment.

When the mobile equipment has retrieved the SIM service table, it will send its own list of capabilities. A SIM with no proactive components has no need for this table, as it will only handle simple requests from the ME, never needing to know what the ME supports. Modern-day SIM cards can use this information to change their behaviour: for example, a phone without a display (capability 110) will not be able to display the messages. The following terminal profile table comes from a HTC Desire phone, with only the most interesting parts of the trace shown here. Note that while the SIM Service Table uses two bits for every capability, the Terminal Profile uses only one and therefore can store 8 capabilities in a byte.

```
      COMMAND: 80 10 00 00 14
               TERMINAL PROFILE
 COMMAND DATA: FF FF FF FF 1F 00 00 DF D7 03 0A 00 00 00 00 06 00 00 00 00
               == Mobile Equipment SIM Toolkit Capabilities ==
               Capability 1: Profile download (enabled)
```

This first capability indicates the mobile equipment supporting the TERMINAL PROFILE command. Since it appears inside the data of the TERMINAL PROFILE command, one would assume this capability is always set.

Following are some capabilities for the Proactive SIM set (most lines have been removed for the sake of compactness):

```
               Capability 2: SMS-PP data download (enabled)
               Capability 3: Cell Broadcast data download (enabled)
               Capability 4: Menu selection (enabled)
               Capability 5: SMS-PP data download (enabled)
               Capability 6: Timer expiration (enabled)
               Capability 16: Display Text (enabled)
               Capability 17: Proactive SIM: DISPLAY TEXT (enabled)
               Capability 18: Proactive SIM: GET INKEY (enabled)
               Capability 19: Proactive SIM: GET INPUT (enabled)
               Capability 20: Proactive SIM: MORE TIME (enabled)
               Capability 21: Proactive SIM: PLAY TONE (enabled)
               Capability 29: Proactive SIM: SET UP CALL (enabled)
               Capability 30: Proactive SIM: SET UP MENU (enabled)
               Capability 31: Proactive SIM: PROVIDE LOCAL INFORMATION (MCC, MNC,
                              LAC, Cell ID & IMEI) (enabled)
               Capability 32: Proactive SIM: PROVIDE LOCAL INFORMATION (NMR) (enabled)
               Capability 59: Proactive UICC: PROVIDE LOCAL INFORMATION (date, time
                              and time zone) (enabled)
               Capability 60: GET INKEY (enabled)
               Capability 61: SET UP IDLE MODE TEXT (enabled)
               Capability 62: RUN AT COMMAND (not enabled)
               Capability 63: SETUP CALL (enabled)
               Capability 64: Call Control by NAA (enabled)
               Capability 65: DISPLAY TEXT (enabled)
```

```
                Capability 66: SEND DTMP command (enabled)
                Capability 67: Proactive UICC: PROVIDE LOCAL INFORMATION
                               (NMR) (enabled)
                Capability 68: Proactive UICC: PROVIDE LOCAL INFORMATION
                               (language) (not enabled)
                Capability 69: Proactive UICC: PROVIDE LOCAL INFORMATION
                               (Timing Advance) (enabled)
                Capability 70: Proactive UICC: LANGUAGE NOTIFICATION (not enabled)
                Capability 71: Proactive UICC: LAUNCH BROWSER (enabled)
                Capability 72: Proactive UICC: PROVIDE LOCAL INFORMATION
                               (Access Technology) (enabled)
```

In the last entries of the trace above, the mobile equipment tells the SIM card what parts of the Proactive SIM/UICC standard it understands. After this, it tells the SIM some of its physical properties:

```
                Capability 97: CSD (not enabled)
                Capability 98: GPRS (not enabled)
                Capability 99: Bluetooth (not enabled)
                Capability 100: IrDA (not enabled)
                Capability 101: RS232 (not enabled)
                Capability 105: (Screen height) (not enabled)
                Capability 110: No display capability (not enabled)
                Capability 111: No keypad capability (not enabled)
                Capability 112: Screen Sizing Parameters (not enabled)
                Capability 113: (Screen width) (not enabled)
                Capability 120: Variable size fonts (not enabled)
                Capability 121: Display can be resized (not enabled)
                Capability 122: Text Wrapping supported (enabled)
                Capability 123: Text Scrolling supported (enabled)
                Capability 124: Text Attributes supported (not enabled)
     RESPONSE: 91 28
                SUCCESS (38 bytes of extra SIM Toolkit information in memory)
```

Now that the SIM card has received the full capability set from the mobile equipment, it has enough information to send its first proactive command. It notifies the phone about this using the `91 XX` response to the `TERMINAL PROFILE` command above, also containing the length of the SIM Toolkit message. The mobile equipment can now retrieve this message using the `FETCH` APDU, whose response will consist of TLV (Tag-Length-Value) objects [8, annex C, page 188].

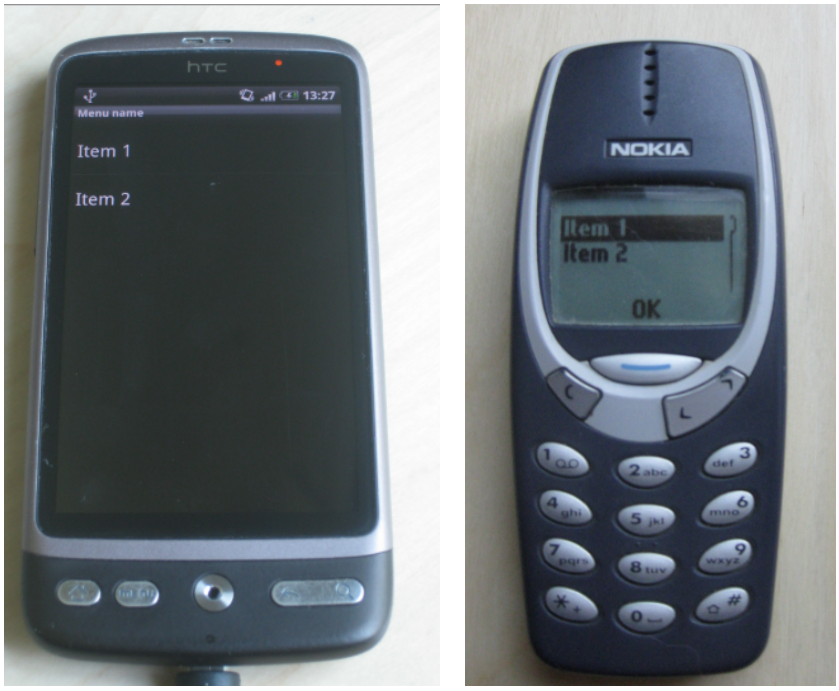
3.2 SIM Toolkit menu installation

In this case, the mobile equipment advertised the capability for SIM Toolkit menus, so most observed SIM cards will now run a `SET UP MENU` command:

```
      COMMAND: 80 12 00 00 28
               FETCH
     RESPONSE: 90 00
               SUCCESS
RESPONSE DATA: D0 26 81 03 01 25 00 82 02 81 82 85 09 4D 65 6E 75 20 6E 61 6D 65 8F 07
               80 49 74 65 6D 20 31 8F 07 81 49 74 65 6D 20 32
               == Proactive SIM command from SIM Toolkit to phone ==
               Command details tag (comprehension required)
                 Command sequence number: 1
                 Type of command: 25 (SET UP MENU)
                 Command Qualifier: 00 (unknown)
```

```
Device identity tag (comprehension required)
  Source: SIM
  Destination: ME
Alpha identifier tag (comprehension required)
  Alpha identifier: 4D 65 6E 75 20 6E 61 6D 65
  As string: "Menu name"
Item tag / eCAT client identity (comprehension required)
  Item '80': "Item 1"
Item tag / eCAT client identity (comprehension required)
  Item '81': "Item 2"
```

As can be seen above, the SIM sets up a SIM Toolkit menu called "Menu name", with two items, each with an identifier and a label. This corresponds to the menu shown in figure 5.



(a) HTC Desire (b) Nokia 3310

Figure 5: The example menu given above.

As soon as the phone has set up the SIM Toolkit menu, it sends a success response to the SIM. It stores the menu for when the user requests it, but does not yet display anything. (For a menu that is immediately displayed, SELECT ITEM is used, as described in section 5.1.3.)

```
      COMMAND: 80 14 00 00 0C
               TERMINAL RESPONSE
 COMMAND DATA: 81 03 01 25 00 02 02 82 81 03 01 00
               == Response from phone to proactive SIM command from SIM Toolkit ==
               Command details tag (comprehension required)
                 Command sequence number: 1
                 Type of command: 25 (SET UP MENU)
                 Command Qualifier: 00 (unknown)
               Device identity tag (comprehension not required)
                 Source: ME
                 Destination: SIM
               Result tag (comprehension not required)
```

```
                Result: 00 (Command performed succesfully)
                Status: Succesfully performed
                Additional information:
    RESPONSE: 90 00
              SUCCESS
```

The SIM responds with `90 00`, which means there are no more proactive commands to send. The SIM Toolkit stack of the phone will now simply wait for incoming commands again, either from the user or from the SIM.

3.3 Menu selection

The incoming command from the user comes when the user actually picks one of the menu items in the SIM Toolkit menu. When this happens, the mobile equipment sends an `ENVELOPE` APDU to the SIM with the menu selection tag:

```
     COMMAND: 80 C2 00 00 09
              ENVELOPE (SMS Point-to-point or cell broadcast data download)
COMMAND DATA: D3 07 02 02 01 81 10 01 80
              == Command from ME / network to SIM ==
              Type: D3 (Menu Selection)
              Device identity tag (comprehension not required)
                Source: Keypad
                Destination: SIM
              Item identifier tag / Encapsulated envelope type (comprehension not required)
                Item identifier: 80
    RESPONSE: 91 25
              SUCCESS (37 bytes of extra SIM Toolkit information in memory)
```

Item identifier `80` was chosen, which corresponds to the first entry in the list above. The SIM processes the choice, and usually will have new SIM Toolkit commands to respond or ask follow-up questions. In this example, the SIM responds with an input request, which pops up on the screen:

```
     COMMAND: 80 12 00 00 25
              FETCH
    RESPONSE: 90 00
              SUCCESS
RESPONSE DATA: D0 23 81 03 01 23 01 82 02 81 82 8D 14 F4 47 69 76 65 20 6D 65 20 73 6F
               6D 65 20 69 6E 70 75 74 21 91 02 01 0A
               == Proactive SIM command from SIM Toolkit to phone ==
               Command details tag (comprehension required)
                 Command sequence number: 1
                 Type of command: 23 (GET INPUT)
                 Command Qualifier: 01 (Characters from SMS default alphabet; ME
                 may echo user input on display; User input to be in unpacked format)
               Device identity tag (comprehension required)
                 Source: SIM
                 Destination: ME
               Text string tag (comprehension required)
                 Encoding: F4 / 8-bit default SMS
                 String: "Give me some input!"
               Response length tag (comprehension required)
                 Length between 1 and 10 characters
```

A dialog as shown in figure 6 appears on the screen, containing the text "Give me some input!" and an input field that follows the rules as laid down by the SIM card, allowing between 1 and 10 characters.

(a) HTC Desire (b) Nokia 3310

Figure 6: The GET INPUT screen after the above command.

Figure 7: The RebelSIM (to the right), connected to a HTC Desire phone.

4 Man-in-the-middle tools

For reading along with the GSM traffic, eavesdropping tools are necessary, consisting of a hardware part that reads in on the traffic, and a software part that can interpret or modify it. In this section, I will describe some tools that are useful for eavesdropping the communication between the mobile equipment and the phone. Some of these tools are also able to modify the traffic as it flows, making many experiments possible.

All of these tools are ultimately useful, in part due to their limitations. The RebelSIM tool only supports eavesdropping, it cannot change the traffic. However, it does work on all phones. The SmartLogicTool can change the traffic, but turns out only to work on old phones. Finally, the Bladox TurboSIM card can change traffic for all phones, but it cannot display what it changed.

4.1 RebelSIM

The tool I started out with is the Rebel Simcard Scanner (figure 7), a $25 tool that relays traffic between the phone and the SIM card and allows eavesdropping over a serial connection[1]. It has several limitations:

- One cannot see the direction of the traffic; it must be derived using understanding of the protocol. (See also section 4.2 about SIMparser.pl below.)
- As the tool is directly connected to the analog line between SIM and phone, changing the traffic as it is sent is impossible.
- For the same reason, you'll have to know the baud rate of the communication. The baud rate is dependent on the clock frequency of the terminal (mobile equipment) and differs wildly. It can be determined using an oscilloscope or the SmartLogicTool, described below.
- If the baud rate is not set exactly to that of the communication, or the phone's clock is not very reliable, at some point the bit phase timing of the serial reader will be off and no communication or only garbage will be read. The communication between the SIM and the phone will still continue unharmed, but it will be invisible to the eavesdropper.

[1] `http://bb.osmocom.org/trac/wiki/RebelSIM_Scanner`

4.2 SIMparser.pl

The SIMparser.pl tool[2] is a Perl script that I wrote to interpret the output of the HTerm application[3]. When using HTerm together with the RebelSIM hardware, the application delivers time-stamped output in the following format:

```
16:08:31.359:
00A400
16:08:31.359:
0402A46F
16:08:31.359:
B7612100
```

Above this output, I added a personal header describing what the log was about and when I traced it. The SIMparser.pl tool would then analyse the log, and parse and describe the protocol. Since the RebelSIM Scanner tool does not show you where the bytes came from, an understanding of the protocol is necessary to figure out who sent what. For example, it depends on the specific smart card command sent by the mobile equipment whether additional data will be sent by the terminal or by the SIM card. The SIMparser.pl tool knows these protocol ambiguities and is able to provide a correct parsing of the above example:

```
  0.000:       COMMAND: 00 A4 00 04 02
                        SELECT file [P1=04?]
          COMMAND DATA: 6F B7
                        elementary file under dedicated file; ECC (Emergency Cell Codes)
  0.000:      RESPONSE: 61 21
                        SUCCESS, 33 extra response bytes, use GET RESPONSE
```

Here, the command as given above is split up between the command parts and the response parts, using the Command and Response APDUs as explained in section 2.1.1. Also, the timing of the commands and responses is seen to the left. The tool recognises that the mobile equipment is asking for the ECC file that contains telephone numbers for the emergency services.

4.3 SmartLogic Tool

The SmartLogic Tool [10] was developed by the Digital Security group at the Radboud University in Nijmegen, to be a general smartcard research tool that can easily be set up to do a man-in-the-middle attack. It uses a client-server architecture over IP, where the server directly interfaces with a single smart card, and multiple clients interfacing with terminals can connect to the server. This architecture allows relaying over the Internet, which has even been shown to work reliably on Dutch Chipknip money transfers [10] even though the Chipknip system is intended for simple point-of-sale transfers.

For most purposes, it has succeeded in its goal. It is possible to write Java code that is imported into the SmartLogic Server to modify the traffic; in section A.1 an example of this code can be seen. Using this tool, it is relatively easy to research the effect of changes in the protocol.

However, this tool is not without its problems. As it was originally intended and primarily tested as a tool for researching smart cards in financial contexts, it seems to be incapable to handle the high speeds of traffic normal in SIM communication. As the communication speeds get higher, the chances of bytes being misread increases, causing communication problems. On some older phones (notably the Nokia 3310) this does not pose a problem as the phone simply tries to send its commands again until it receives a valid response. However, on most smart phones this makes research impossible, as the phone simply decides the SIM is corrupt and does not re-attempt the communication.

The tool is still very useful for determining the frequency of the mobile equipment clock. This frequency can be converted to a baud rate, which is needed to correctly eavesdrop the analog communications by

[2] https://github.com/sgielen/simparser
[3] http://www.der-hammer.info/terminal/

the RebelSIM. In my experience the clock frequencies reported by the SmartLogic Tool are a bit too high; for example, for the Nokia 3310 it reported a clock frequency of 3.25 MHz, leading to a standard baud rate of 8736 symbols per second. Eavesdropping the communication worked best when using a baud rate of 8600 symbols per second.

4.3.1 Configuring

In my experience, the SmartLogic Tool is most easily configured on an Ubuntu machine, where most of the command-line tools needed are readily available in the repositories. The code of the SmartLogic Tool is available from Google Code[4]. The client first initialises the SmartLogic Tool hardware by flashing its firmware, and then listening for incoming commands from the phone. Some changes to the server code are necessary to switch it to SIM card mode. After this, the client and the server can be put together using the `-s` flag to the client, and eavesdropping can begin. Also, Java code changes are possible in the server to modify the communication.

The error `Received 7 bytes of data, expected 9 bytes` can be fixed by removing the file `SmartLogic.ihx` and running `make`.

4.4 Bladox TurboSIM

The TurboSIM manufactured by Bladox[5] is a very thin SIM card sized chip with an ARM microcontroller. It has SIM contacts on both sides, and is typically placed in a mobile phone between the SIM and the terminal contacts, after which it can relay and modify the traffic. Applications written in C can be uploaded to it using the Bladox programmer. Out-of-the-box, it replaces the SIM Toolkit menu with its own, leaving the original menu accessable as a sub-menu option. After that, applications can add entries to the SIM Toolkit menu, provide substitute contents for files on the SIM and activate or deactivate SIM services. The Bladox TurboSIM is very useful to change any SIM Toolkit traffic, but due to its small size it has no way of displaying any communication.

The following example application adds an entry to the SIM Toolkit menu, that simply displays a message when selected:

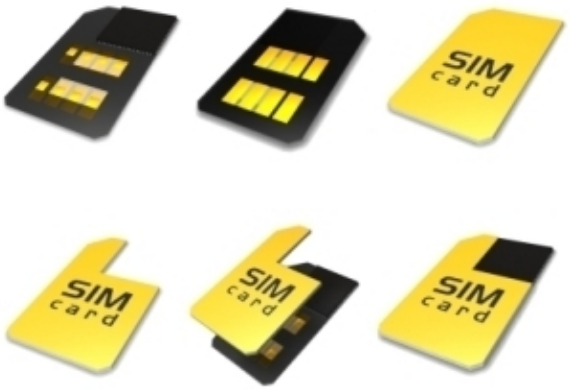


Figure 8: SIM card cutting

[4] `http://code.google.com/p/smartlogictool/`
[5] `http://bladox.cz/?lang=en`

```
#include <config.h>
#include <turbo/turbo.h>

u8 PROGMEM t_Foo_en[] = "Hello World!";

void action_menu (void *data)
{
  display_text_raw (t_Foo_en, Q_DISPLAY_TEXT_USER_CLEAR);
}

void turbo_handler (u8 action, void *data)
{
  switch (action)
  {
    case ACTION_INSERT_MENU:
      insert_menu (t_Foo_en);
      break;
    case ACTION_MENU_SELECTION:
      stk_thread (action_menu, data);
      break;
    default:
      break;
  }
}
```

4.5 Active man-in-the-middle method

To find out phone capabilities for any SIM Toolkit proactive commands, or exploit any security issues we find, we will need to send proactive commands to the phone using the SIM communication channel. But, since behaviour might be different when not connected to a network, and existing SIMs do not disclose their private keys which are required for contacting the network, we will also need to have a correct SIM card in the communication channel.

Considering this, we need a way to use a working SIM in the phone, and still be able to inject SIM Toolkit traffic. As it turns out, this can be done using the SmartLogicTool described in section 4.3. We want to let the normal GSM traffic get through as always, but replace all SIM Toolkit traffic with our own. (The Bladox overlay as described in section 4.4 already does this on its own.)

I will show that injecting SIM Toolkit is relatively easy, by describing the APDUs that need to be monitored or changed by a man-in-the-middle in order to modify the traffic. There are 6 APDUs, sent from mobile equipment to the SIM that specifically need to be modified or monitored:

- `TERMINAL PROFILE` in order to detect terminal capabilities and insert the first SIM Toolkit command
- `FETCH` in order to send our own SIM Toolkit commands
- `TERMINAL RESPONSE` in order to handle command results ourselves
- `ENVELOPE` in order to capture events and act upon them
- `SELECT/READ BINARY` in order to modify the SIM service table

The first time the SIM sends SIM Toolkit traffic is when the phone sends the TERMINAL PROFILE APDU (section 3.1). The SIM card may respond to this APDU with SIM Toolkit traffic, by changing the normal success status word `90 00` to `91 XX`, where `XX` indicates the number of bytes of proactive SIM commands to send [8, section 6.3, page 34]. By intercepting this response, and changing it to send the number of bytes of SIM Toolkit traffic we want to send, the first step of SIM Toolkit traffic injection is set.

Next, we need to intercept any FETCH APDU, and prevent it from being sent to the SIM card which would activate the SIM Toolkit mechanism in the SIM. Instead of sending it to the SIM, the man-in-the-middle responds with its own SIM Toolkit proactive command in memory. The phone does not know the SIM Toolkit traffic did not actually come from the SIM card, and executes the command as always.

Three steps still remain. First, the phone will at some point process the proactive command, and send a TERMINAL RESPONSE APDU to the SIM. Our man-in-the-middle needs to intercept this response, and handle it itself. This will allow subsequent messages to be sent from our own alternative SIM Toolkit stack. Second, messages from the phone, the user or the network to the SIM card may come in using the ENVELOPE APDU, which also needs to be sent to the alternative SIM Toolkit implementation. Also, some interesting procedures are triggered by services in the SIM Service Table (section 3.1), so a man-in-the-middle that needs to change the service table must watch SELECT commands until the service table is selected, then replace the contents with its own.

Lastly, the alternative implementation must change any `90 00` status word to `91 XX` when it has a proactive command to send, and any `91 XX` from the SIM must first be changed to `90 00` or `91 YY` (where `YY` is the correct size of the waiting command) to prevent de-synchronisation between the phone and the alternative SIM stack.

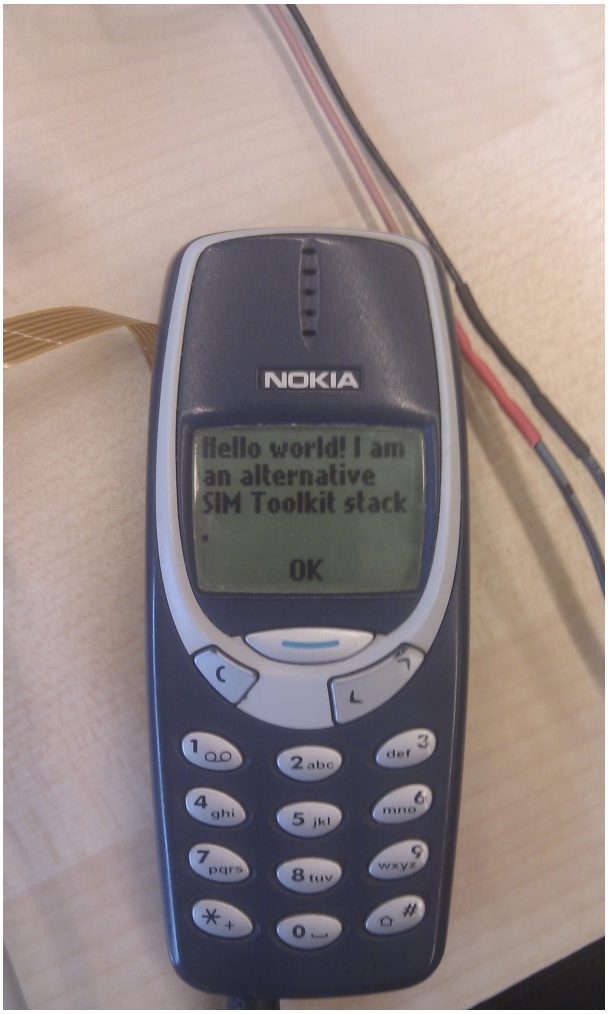


Figure 9: A DISPLAY TEXT command executed by a Nokia 3310 phone.

5 SIM Toolkit capabilities

In this section, I will examine some real-world capabilities of the SIM Toolkit standard. For sending any commands in this section, I used the approach described in section 4.5. The commands described are proactive SIM commands, displayed in their raw byte form, tested in practice using the SmartLogic Tool described in 4.3.

5.1 User communication

First, we focus on SIM Toolkit proactive commands to let the SIM communicate with the user via the mobile equipment.

5.1.1 Display text

The most simple SIM Toolkit command to display is the `DISPLAY TEXT` command, which is intended for displaying a message on the phone display; see figure 9 for an example. In raw byte form, it looks like this:

```
D0 3F            ; This is a proactive command of length 0x3f
  81 03 01 21 81 ; The command is DISPLAY TEXT, high priority, wait for user
```

```
  82 02 81 02    ; It was sent from the SIM to the display
  8D 34 04       ; Encoding is 8-bit default SMS (ASCII), message:
    48 65 6C 6C 6F 20 77 6F 72 6C 64 21 20 49 20 61 6D 20
                 ; "Hello world! I am "
    61 6E 20 61 6C 74 65 72 6E 61 74 69 76 65 20 53 49 4D
                 ; "an alternative SIM"
    20 54 6F 6F 6C 6B 69 74 20 73 74 61 63 6B 2E
                 ; " Toolkit stack."
```

5.1.2 Set up menu

`SET UP MENU` is typically sent at the start of every SIM Toolkit session as described in section 3.2. It installs a SIM Toolkit menu in the phone that the user can access himself, and contains a list of items and their identifiers to report back later. An example can be seen in section 3.2.

When a SET UP MENU is sent, the terminal immediately responds with a TERMINAL PROFILE, usually indicating acceptance of the menu. This does not mean a selection has been made: selections will come in through an ENVELOPE with the menu selection tag set. If such an ENVELOPE comes in and the SIM card wants a new sub-menu to be displayed, this is done through the SELECT ITEM proactive command in response to the ENVELOPE (see section 5.1.3). Such a submenu is typically displayed in the same way as the main menu.

5.1.3 Select item

A `SELECT ITEM` proactive command causes the mobile equipment to present a menu with pre-defined options to the user. Once the user makes a choice, it is sent back to the SIM in a `TERMINAL RESPONSE`. The difference with the `SET UP MENU` command is that the latter does not immediately show on the screen, but is cached in phone memory until the user explicitly requests it. Usually, the two menus do look the same on the mobile equipment screen.

```
D0 1B            ; This is a proactive command of length 0x1b
  81 03 01 24 00 ; The command is SELECT ITEM
  82 02 81 02    ; It was sent from the SIM to the display
  8F 07 AB       ; Item with unique identifier 0xab
    49 74 65 6D 20 31 ; "Item 1"
  8F 07 AC       ; Item with unique identifier 0xac
    49 74 65 6D 20 32 ; "Item 2"
```

5.2 Network communication

In this section, I will describe some SIM Toolkit proactive commands to get information about or communicate with the network. This can be useful for SIM Toolkit applications to call-back to the operator to execute user-requested commands.

5.2.1 Provide local information

A SIM Toolkit command that provides active network information is `PROVIDE LOCAL INFORMATION`, using which the SIM card can gather information like the unique identifier of the phone, the date and time and battery state [2, page 28, 6.4.15]. Also, the network cell ID and timing advancement of the GSM protocol can be requested, using which the phone location can be derived. Two examples of the `PROVIDE LOCAL INFORMATION` command follow:

```
D0 09            ; This is a proactive command of length 0x09
  81 03 03 26 00 ; The command is PROVIDE LOCAL INFORMATION (Network info)
  82 02 81 82    ; It was sent from the SIM to the mobile equipment
```

```
D0 09            ; This is a proactive command of length 0x09
  81 03 04 26 01 ; The command is PROVIDE LOCAL INFORMATION (IMEI number)
  82 02 81 82    ; It was sent from the SIM to the mobile equipment
```

The terminal responds with `TERMINAL RESPONSE` with the requested information. The coding of the MCC/MNC is as in 3GPP TS 24.008 [3], in this case the mobile country code is 204, the network code is 08, which in this case means the network is KPN in the Netherlands[6].

```
81 03 03 26 00   ; Response to PROVIDE LOCAL INFORMATION (Network info) request
02 02 82 81      ; Sent from the mobile equipment to the SIM
03 01 00         ; Result: 0 (success)
13 07            ; Requested information:
   02 F4 80      ; Mobile Country/Network Code
   11 7F         ; Local Area Code
   32 22         ; Cell ID

81 03 04 26 01   ; Response to PROVIDE LOCAL INFORMATION (IMEI number) request
02 02 82 81      ; Sent from the mobile equipment to the SIM
03 01 00         ; Result: 0 (success)
14 08            ; Requested information: IMEI number
   3A 05 01 84 50 44 01 02
```

5.2.2 Send short message

To send SMS messages, there is a `SEND SHORT MESSAGE` command. It contains a `SMS TPDU`, the format used in the GSM protocol to describe the destination, parameters and contents of the message. Also, an alpha identifier is optional and will be displayed on the phone if set. If an alpha identifier is given and has a length of zero, the phone may not give an indication to the user that a message is being sent. This works as described on the HTC Desire, but the Nokia 3310 asks the user for permission whatever the value of the alpha identifier.

An example of a `SEND SHORT MESSAGE` with an alpha identifier follows:

```
D0 2D            ; This is a proactive command with length 0x2d
  81 03 05 13    ; Message type is 0x13 (SEND SHORT MESSAGE)
           00    ; Packing is not required
  82 02 81 82    ; Command from UICC to terminal
  85 0F          ; Alpha identifier: "Visible message"
     56 69 73 69 62 6C 65 20 6D 65 73 73 61 67 65
  8B 11          ; Beginning of the SMS TPDU, length 0x11
     31          ; 0b0011_0001
                 ;          ^^ TP-MTI: SMS-SUBMIT (send an SMS message)
                 ;         ^   TP-RD: Don't accept duplicates
                 ;       ^ ^   TP-VPF: validity period starts counting now
                 ;      ^      TP-SRR: don't send a status report
                 ;     ^       TP-UDHI: no headers in the user data
                 ;    ^        TP-RP: no Reply-Path in this SUBMIT
     00          ; TP-MR: message sequence number, will be set by ME
     0b          ; TP-Destination-Address, number of semi-octets in number
     91          ; TP-Destination-Address, type = international telephone nr
     13 16 32    ; TP-Destination-Address: 31 61 23   (so the number used
     54 76 F8    ; TP-Destination-Address: 45 67 8     is +31 6 12345678)
     00          ; TP-Protocol-Identifier: standard MS to SMSC transfer
     00          ; TP-Data-Coding-Scheme: standard 7-bit encoding
```

[6] http://en.wikipedia.org/wiki/Mobile_Network_Code#N

```
     AA          ; TP-Validity-Period: 4 days
     03          ; TP-User-Data-Length: 3 septets
     C8 77 1A    ; "H o i"
```

The message will be sent to the SMS center, which will accept or deny the message. The mobile equipment will interpret the SMS center response and send it to the SIM in a `TERMINAL RESPONSE`; an example of a successful and an error response are given here:

```
81 03 05 13 00 ; This is a response to your SEND SHORT MESSAGE (message 0x05)
02 02 82 81    ; It is being sent from terminal to UICC
03 01 00       ; Result: Command was executed succesfully

81 03 05 13 00
02 02 82 81
03 03 35       ; Result: Failure: SMS RP-ERROR
         15    ; reason code 21 "Short message transfer rejected"
```

5.2.3 Set up call

Using the SET UP CALL proactive command, the SIM can request the terminal to call a specific number. The SIM Toolkit standard allows the phone to confirm setting up the call with the user [8, section 6.4.13]. Unfortunately, I cannot give a trustworthy listing of actual bytes that were sent, because of an equipment failure before this thesis was printed. However, the following code for the Bladox TurboSIM was tested to succesfully ask the terminal to set up a call to the number in `t_ms`:

```
#include <config.h>
#include <turbo/turbo.h>
#include "utils.h"

u8 PROGMEM t_menuName_en[] = "Set up call";
u8 PROGMEM t_ms[] = "+31612345678"; // <-- Phone number here

void action_menu (void *data)
{
  u8 *address = str2msisdn(t_ms, MSISDN_ADN, MEM_R);
  set_up_call(address, MSISDN_ADN, "");
  free(address);
}

void turbo_handler (u8 action, void *data)
{
  switch (action) {
    case ACTION_INSERT_MENU:
      insert_menu (t_menuName_en); break;
    case ACTION_MENU_SELECTION:
      stk_thread (action_menu, data); break;
    default: break;
  }
}
```

5.2.4 Call control

The SIM Service Table described in section 3.1 has a service called *Call Control* (#28). When this service is enabled, the terminal is required to ask the SIM card for confirmation for every outgoing call [8, section 7.3.1.1] using the `ENVELOPE` command [8, section 7.3.1.6]. The SIM card can respond with status code

`90 00` to allow the call, `93 00` to deny the call, or `9F XX` with 0xXX bytes of further response data. If response data is sent, it itself also contains a byte to control whether the ME is allowed to set up the call with the given parameters, whether it is not allowed, or whether the parameters will change. For example, the following code for the Bladox TurboSIM will disallow any call set-up requests by the mobile equipment:

```
#include <config.h>
#include <turbo/turbo.h>
#include "utils.h"

u8 PROGMEM t_ms[] = "+31612345678";
u8 PROGMEM ef_sst[] = { 0x6f, 0x38 };

void fake_sim_file(File_apdu_data *fa)
{
  // When the sim service table contents are requested, we return the full
  // service table of the SIM with two bits forcibly set to 1.
  if(fa->ins == ME_CMD_READ_BINARY) {
    u8 *servicetable = sim_sst();
    // first byte is the length
    if(servicetable[0] >= 7) {
      // the service table is long enough to contain service 28 Call Control,
      // enable it
      servicetable[7] = servicetable[7] & 0x03;
    }
    memcpy(fa->data, servicetable + 1, servicetable[0]);
    fa->data[fa->p3] = 0x90;
    fa->data[fa->p3 + 1] = 0x00;
  } else if(fa->prev_ins == ME_CMD_ENVELOPE && fa->ins == ME_CMD_GET_RESPONSE) {
    u8 *r = buf_B();
    r[0] = 0x01; // deny call
    r[1] = 0x07; // length
    r[2] = 'B';
    r[3] = 'a';
    r[4] = 'd';
    r[5] = ' ';
    r[6] = 'n';
    r[7] = 'u';
    r[8] = 'm';
    r[9] = 0x90;
    r[10] = 0x00;
  }
}

void turbo_handler (u8 action, void *data)
{
  switch (action)
  {
    case ACTION_APP_INIT:
      reg_file(ef_sst, 1);
      reg_action(ACTION_CALL_CONTROL);
      break;
    case ACTION_FILE_APDU:
      fake_sim_file(data);
      break;
    case ACTION_CALL_CONTROL:
```

```
      retval(0x9f09);
      break;
    default:
      break;
  }
}
```

5.2.5 Events

Events are a way for the SIM card to get immediate notice of a changing environment, without it having to ask every so-many seconds. A SIM Toolkit application uses the `SET UP EVENT LIST` proactive command to notify the mobile equipment of what events it is interested in, and when the action is triggered, an `ENVELOPE` is sent to the SIM with the action parameters. For example, a `LOCATION CHANGED ENVELOPE` will be sent containing the new location. Due to an equipment failure the actual protocol bytes cannot be given here, but the following code for the Bladox TurboSIM card registers the `LOCATION CHANGED` event and every time the event is triggered, the application displays a message using `DISPLAY TEXT`:

```
#include <config.h>
#include <turbo/turbo.h>
#include "utils.h"

u8 PROGMEM t_loc_changed_en[] = "Location changed";

void event_location (void *data)
{
    display_text_raw(t_loc_changed_en, Q_DISPLAY_TEXT_HIGH_PRIORITY);
}

void turbo_handler (u8 action, void *data)
{
  switch (action)
  {
    case ACTION_APP_INIT:
      // If ACTION_EVENT_* is registered during ACTION_APP_INIT,
      // the event list to the ME is automatically updated. Otherwise,
      // set_up_event_list() is needed.
      reg_action(ACTION_EVENT_LOCATION_STATUS);
      break;
    case ACTION_EVENT_LOCATION_STATUS:
      stk_thread (event_location, data);
      break;
    default:
      break;
  }
}
```

6 Security implications of SIM Toolkit

In this section, I will describe several attacks that the SIM Toolkit standard makes possible. As the traffic between the mobile equipment and the SIM card is unencrypted, including the SIM Toolkit messages, it is easy to read and modify the traffic once a man-in-the-middle is in place. This man-in-the-middle is easy to create using the thin Bladox TurboSIM card overlay as in section 4.4. Therefore, with hardware like this and firmware/software as described in 4.5, the attacks in this section become viable. The TurboSIM has to be placed between the contacts of the mobile equipment and the SIM card, and will change the traffic as described in the subsections below.

6.1 Denial of service

Using SIM Toolkit, messages can be displayed that fill the phone screen until the user explicitly dismisses them. By doing this in a loop, messages can block the screen forever causing a denial of service.

In section 5.1.1, an example of the `DISPLAY TEXT` command is shown that has the "high priority" and "wait for user" bits set. This means the message must fill the screen and the user must explicitly dismiss them. When the user dismisses the message, either by pressing the "back" button, the "OK" button or some other facility on the phone, the phone sends a `TERMINAL RESPONSE` command back to the SIM card. An alternative SIM toolkit stack could re-send the same `DISPLAY TEXT` command in response to the `TERMINAL RESPONSE`, and cause an infinite loop and a denial of service. (Example code for the SmartLogicTool to this effect can be seen in section A.1.)

6.2 Disclose private information

Using SIM Toolkit, the connected network cell can be requested, which can be translated into a geographical location using freely available tools. Combined with the ability to send text messages, this could allow an attacker to track the location of a victim.

The `PROVIDE LOCAL INFORMATION` command described in section 5.2.1 will cause the mobile equipment to respond with the connected network and cell ID, which when combined make up an unique identifier for a certain cell in the network. When these two numbers are sent to an attacker using the `SEND SHORT MESSAGE` (section 5.2.2) command, an attacker can see the location of a victim with an accuracy of a few kilometers, or less in populated areas. When combined with the "location changed" event described in section 5.2.5, an attacker can increase this accuracy and closely follow the victim.

6.3 Man-in-the-middle on calls

Using the "call control" service provided by SIM Toolkit (section 5.2.4), a SIM card can control all outgoing calls made by the mobile equipment, and block or re-route the call. This makes it possible to eavesdrop on a victim's calls, by re-routing all interesting calls to an eavesdropping telephone number that also transparently forwards the call.

6.4 Using up credit

A SIM Toolkit application can use up credit or cause a high bill by sending SMS messages in a loop. This can be done using the `SEND SHORT MESSAGE` command described in section 5.2.2. This can be hidden from the user, causing him to only notice the used up credit much later.

6.5 Forward authentication codes

Depending on the phone, a SIM Toolkit application may be able to receive and check incoming SMS messages, and forward them silently to an eavesdropping number. When a victim uses SMS messages to authenticate himself, for example to do a bank transaction, these can be caught and forwarded, facilitating identity fraud.

7 Conclusions

In this thesis, I gave a thorough explanation with examples of how SIM Toolkit works and what applications on the SIM card can do with it. In my words, the goal of the SIM Toolkit was to allow the SIM to gain value-added services, by allowing applications on the SIM to communicate with the user and the network. Using the proactive commands standardised in the SIM Toolkit standard and described in this thesis, this goal is succesfully fulfilled.

However, in this thesis, I have also shown that the possibilities of the SIM Toolkit standard can be turned into risks when it is possible to gain control over the communication between the mobile equipment and the SIM (see section 6): from mere april's fools jokes like constantly hogging the phone screen with a message, making it unable to use, to serious privacy issues like following a victim's location or eavesdropping on his outgoing calls as they are made.

Because the GSM standards of which the SIM Toolkit standard are a part are complex and with thousands of pages can be difficult to browse, I hope this thesis will be useful for people starting to develop using the SIM Toolkit. Using the tools described and the examples provided, one can quickly start developing and testing. Especially the SIMParser tool (section 4.2) I created that automatically explains SIM card traffic eavesdropped by the RebelSIM hardware (section 4.1) will hopefully be helpful to quickly familiarize oneself with the protocol, and see the result of various experiments on the SIM card and the phone.

Looking back on the past year, there are a number of observations I can make. First of all, I did not expect to have a large section with multiple tools to use to eavesdrop and change the communication: at that point, the SmartLogic Tool developed by the Digital Security department of the Radboud University Nijmegen looked very promising and was described to be able to do everything I needed for this thesis. However, it turned out it began to fail on higher communication speeds used by mobile phones. Together with the original author of the device, I searched why this could be, but we did not find a solution after spending too long searching and I had to combine it with other tools to continue with the thesis.

I also spent a lot of time learning about parts of the SIM protocol, and this has been very useful for writing the SIMparser tool, but still took me very long even though it was only background knowledge for this thesis. Even worse, after two months I found out there had been a newer version of this protocol I had never seen, which was difficult at that point to write into the SIMparser. I did eventually succeed here, but it would have been better to ask someone who knew about this in advance.

Reading a lot of SIM Toolkit standards has also been even more difficult than I had imagined before. I knew they were large, but sometimes I nearly gave up after browsing through them for half an hour without finding the technical data I was looking for. However, eventually I could succesfully eavesdrop, interpret and modify existing SIM Toolkit traffic and figured out how the protocol worked in practice.

References

[1] `http://www.3gpp.org/ftp/Specs/html-info/1111.htm`.

[2] 3GPP. *Universal Subscriber Identity Module (USIM) Application Toolkit (USAT)*, 2011. 31.111, version 10.4.0, see `http://www.3gpp.org/ftp/Specs/html-info/31111.htm`.

[3] 3GPP. *Mobile radio interface Layer 3 specification; Core network protocols; Stage 3*, 2012. 24.008, version 11.2.1, see `http://www.3gpp.org/ftp/Specs/html-info/24008.htm`.

[4] European Telecommunications Standards Institute, France. *Specifications of the SIM-ME interface*, 1994. The first version of the GSM TS still publicly available on the Internet: `http://www.etsi.org/deliver/etsi_gts/11/1111/03.16.00_60/gsmts_1111sv031600p.pdf`.

[5] European Telecommunications Standards Institute, France. *Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface*, 1995. GSM 11.11.

[6] European Telecommunications Standards Institute, France. *Digital cellular communications system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile*

Equipment (SIM - ME) interface, 1996. The first version of the GSM STK TS still publicly available on the Internet.

[7] European Telecommunications Standards Institute, France. *Digital cellular telecommunications system (Phase 2+); Specification of the SIM application toolkit for the Subscriber Identity Module - Mobile Equipment (SIM - ME) interface*, 1998. GSM 11.14.

[8] European Telecommunications Standards Institute. *Smart Cards: Card Application Tookit (CAT)*, 2011. The standard for the card application toolkit: `http://www.etsi.org/deliver/etsi_ts/102200_102299/102223/10.05.00_60/ts_102223v100500p.pdf`.

[9] European Telecommunications Standards Institute. *Smart Cards; UICC-Terminal Interface; Physical and logical characteristics*, 2011. The standard for SIM-ME communication: `http://www.etsi.org/deliver/etsi_ts/102200_102299/102221/10.00.00_60/ts_102221v100000p.pdf`.

[10] Joeri de Ruiter Gerhard de Koning Gans. *The SmartLogic Tool: Analysing and Testing Smart Card Protocols*. PhD thesis, Institute for Computing and Information Sciences, Radboud University Nijmegen, 2012. `http://gerhard.dekoninggans.nl/documents/publications/smartlogic.tool.pdf`.

[11] International Organization for Standardization (ISO), ISO copyright office, Geneva. *Identification cards – Integrated circuit cards*, 1998-2007. Reference number ISO/IEC 7816.

[12] Konstantinos Markantonakis Keith Mayes. *Smart Cards, Tokens, Security and Applications*. Springer Science + Business Media, LLC., 2008. ISBN: 987-0-387-72197-2.

A Code

A.1 SmartLogicTool man-in-the-middle code

```
public interface SimToolkitStack {
       public boolean proactiveCommandsAvailable();
       public byte    proactiveCommandLength();
       public byte[] fetch();
       public void envelope(byte[] envelope);
       public void terminalResponse(byte[] response);
       public void terminalProfile(byte[] profile);
       public byte[] simServiceTable(byte[] realSST);
};


public class SimToolkitStackDefault implements SimToolkitStack {
        // TS 102.223 v10.05.00
        public static final byte STK_CMD_REFRESH = 0x01;
        public static final byte STK_CMD_MORE_TIME = 0x02;
        public static final byte STK_CMD_POLL_INTERVAL = 0x03;
        public static final byte STK_CMD_POLLING_OFF = 0x04;
        public static final byte STK_CMD_SETUP_EVENT_LIST = 0x05;
        public static final byte STK_CMD_SET_UP_CALL = 0x10;
        public static final byte STK_CMD_SEND_SS = 0x11;
        public static final byte STK_CMD_SEND_USSD = 0x12;
        public static final byte STK_CMD_SEND_SHORT_MESSAGE = 0x13;
        public static final byte STK_CMD_SEND_DTMF = 0x14;
        public static final byte STK_CMD_LAUNCH_BROWSER = 0x15;
        public static final byte STK_CMD_GEOGRAPHICAL_LOCATION_REQUEST = 0x16;
        public static final byte STK_CMD_PLAY_TONE = 0x20;
        public static final byte STK_CMD_DISPLAY_TEXT = 0x21;
        public static final byte STK_CMD_GET_INKEY = 0x22;
        public static final byte STK_CMD_GET_INPUT = 0x23;
        public static final byte STK_CMD_SELECT_ITEM = 0x24;
        public static final byte STK_CMD_SET_UP_MENU = 0x25;
        public static final byte STK_CMD_PROVIDE_LOCAL_INFO = 0x26;
        public static final byte STK_CMD_TIMER_MANAGEMENT = 0x27;
        public static final byte STK_CMD_SETUP_IDLE_MODE_TEXT = 0x28;
        public static final byte STK_CMD_CARD_APDU = 0x30;
        public static final byte STK_CMD_POWER_ON_CARD = 0x31;
        public static final byte STK_CMD_POWER_OFF_CARD = 0x32;
        public static final byte STK_CMD_GET_READER_STATUS = 0x33;
        public static final byte STK_CMD_RUN_AT_COMMAND = 0x34;
        public static final byte STK_CMD_LANG_NOTIFICATION = 0x35;
        public static final byte STK_CMD_OPEN_CHANNEL = 0x40;
        public static final byte STK_CMD_CLOSE_CHANNEL = 0x41;
        public static final byte STK_CMD_RECEIVE_DATA = 0x42;
        public static final byte STK_CMD_SEND_DATA = 0x43;
        public static final byte STK_CMD_GET_CHANNEL_STATUS = 0x44;
        public static final byte STK_CMD_SERVICE_SEARCH = 0x45;
        public static final byte STK_CMD_GET_SERVICE_INFORMATION = 0x46;
        public static final byte STK_CMD_DECLARE_SERVICE = 0x47;
        public static final byte STK_CMD_SET_FRAMES = 0x50;
        public static final byte STK_CMD_GET_FRAMES_STATUS = 0x51;
        public static final byte STK_CMD_RETRIEVE_MM = 0x60;
        public static final byte STK_CMD_SUBMIT_MM = 0x61;
```

```
public static final byte STK_CMD_DISPLAY_MM = 0x62;
public static final byte STK_CMD_ACTIVATE = 0x70;
public static final byte STK_CMD_CONTACTLESS_STATE_CHANGED = 0x71;
public static final byte STK_CMD_COMMAND_CONTAINER = 0x72;
public static final byte STK_CMD_ENCAPSULATED_SESSION_CONTROL = 0x73;
public static final byte STK_CMD_END_OF_PROACTIVE_SESSION = (byte)0x81;

// ETSI TS 101.220 v11.0.0 (pg 14)
public static final byte STK_TAG_PROACTIVE = (byte)0xd0;
public static final byte STK_TAG_COMPREHENSION_REQUIRED = (byte)0x80;
public static final byte STK_TAG_COMMAND_DETAILS = (byte)0x01;
public static final byte STK_TAG_DEVICE_IDENTITIES = (byte)0x02;
public static final byte STK_TAG_RESULT = (byte)0x03;
public static final byte STK_TAG_DURATION = (byte)0x04;
public static final byte STK_TAG_ALPHA_IDENTIFIER = (byte)0x05;
public static final byte STK_TAG_ADDRESS = (byte)0x06;
public static final byte STK_TAG_CAPABILITY_CONFIGURATION_PARAMETERS = (byte)0x07;
public static final byte STK_TAG_CALLED_PARTY_SUBADDRESS = (byte)0x08;
public static final byte STK_TAG_SS_STRING = (byte)0x09;
public static final byte STK_TAG_SMS_TPDU = (byte)0x0b;
public static final byte STK_TAG_CELL_BROADCAST_PAGE = (byte)0x0c;
public static final byte STK_TAG_TEXT_STRING = (byte)0x0d;
public static final byte STK_TAG_TONE = (byte)0x0e;
public static final byte STK_TAG_ECAT_CLIENT_PROFILE = (byte)0x0e;
public static final byte STK_TAG_ITEM = (byte)0x0f;
public static final byte STK_TAG_ECAT_CLIENT_IDENTITY = (byte)0x0f;
public static final byte STK_TAG_ITEM_IDENTIFIER = (byte)0x10;
public static final byte STK_TAG_ENCAPSULATED_ENVELOPE = (byte)0x10;
public static final byte STK_TAG_RESPONSE_LENGTH = (byte)0x11;
public static final byte STK_TAG_FILE_LIST = (byte)0x12;
public static final byte STK_TAG_LOCATION_INFORMATION = (byte)0x13;
public static final byte STK_TAG_IMEI = (byte)0x14;
// and many more

// TS 102.223 v10.05.00 page 139 (section 8.7)
public static final byte STK_DEVICE_KEYPAD = (byte)0x01;
public static final byte STK_DEVICE_DISPLAY = (byte)0x02;
public static final byte STK_DEVICE_EARPIECE = (byte)0x03;
public static final byte STK_DEVICE_ADDT_CARD_READER = (byte)0x10;
        // + card reader id (0 to 7)
public static final byte STK_DEVICE_CHANNEL_IDENTIFIER = (byte)0x20;
        // + channel id (1 to 7)
public static final byte STK_DEVICE_ECAT_CLIENT_IDENTIFIER = (byte)0x30;
        // + client id (1 to 15)
public static final byte STK_DEVICE_UICC = (byte)0x81;
public static final byte STK_DEVICE_TERMINAL = (byte)0x82;
public static final byte STK_DEVICE_NETWORK = (byte)0x83;

public String deviceToString(byte device) {
        switch(device) {
        case STK_DEVICE_KEYPAD:                return "Keypad";
        case STK_DEVICE_DISPLAY:        return "Display";
        case STK_DEVICE_EARPIECE:        return "Earpiece";
        case STK_DEVICE_ADDT_CARD_READER:
        case STK_DEVICE_ADDT_CARD_READER+1:
        case STK_DEVICE_ADDT_CARD_READER+2:
```

```
        case STK_DEVICE_ADDT_CARD_READER+3:
        case STK_DEVICE_ADDT_CARD_READER+4:
        case STK_DEVICE_ADDT_CARD_READER+5:
        case STK_DEVICE_ADDT_CARD_READER+6:
        case STK_DEVICE_ADDT_CARD_READER+7:
                return "Additional card reader " + (device - STK_DEVICE_ADDT_CARD_READER);
        case STK_DEVICE_CHANNEL_IDENTIFIER:
        case STK_DEVICE_CHANNEL_IDENTIFIER+1:
        case STK_DEVICE_CHANNEL_IDENTIFIER+2:
        case STK_DEVICE_CHANNEL_IDENTIFIER+3:
        case STK_DEVICE_CHANNEL_IDENTIFIER+4:
        case STK_DEVICE_CHANNEL_IDENTIFIER+5:
        case STK_DEVICE_CHANNEL_IDENTIFIER+6:
        case STK_DEVICE_CHANNEL_IDENTIFIER+7:
                return "Channel identifier " + (device - STK_DEVICE_CHANNEL_IDENTIFIER);
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+1:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+2:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+3:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+4:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+5:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+6:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+7:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+8:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+9:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+10:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+11:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+12:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+13:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+14:
        case STK_DEVICE_ECAT_CLIENT_IDENTIFIER+15:
                return "eCat client identifier " + (device
                        - STK_DEVICE_ECAT_CLIENT_IDENTIFIER);
        case STK_DEVICE_UICC:                return "UICC (SIM)";
        case STK_DEVICE_TERMINAL:        return "Terminal (ME)";
        case STK_DEVICE_NETWORK:        return "Network";
        default:                         return "Unknown device " + device;
        }
}

protected byte[] nextMessage = {};
protected byte sequence = 1;

protected void buildNextMessage(byte type, byte qualifier) {
        if(nextMessage.length != 0) {
                System.out.println("Warning: buildNextMessage() called "
                                   +"while nextMessage already present.");
        }
        byte[] next = {STK_TAG_PROACTIVE, 0x00, /* length will be set later */
                       STK_TAG_COMMAND_DETAILS ^ STK_TAG_COMPREHENSION_REQUIRED,
                         0x03, sequence++, type, qualifier};
        nextMessage = next;
        nextMessage[1] = (byte)(nextMessage.length - 2);
}
protected void addDeviceIdentities(byte source, byte destination) {
        byte[] params = {source, destination};
```

```
        addTagToMessage(STK_TAG_DEVICE_IDENTITIES, true, params);
}
protected void addTextTag(String text) {
        byte[] string = text.getBytes();
        byte[] params = new byte[string.length + 1];
        params[0] = 0x04;
        System.arraycopy(string, 0, params, 1, string.length);
        addTagToMessage(STK_TAG_TEXT_STRING, true, params);
}
protected void addAlphaIdentifier(String text) {
        byte[] string = text.getBytes();
        addTagToMessage(STK_TAG_ALPHA_IDENTIFIER, true, string);
}
protected void addItemTag(byte tag, String text) {
        byte[] string = text.getBytes();
        byte[] params = new byte[string.length + 1];
        params[0] = tag;
        System.arraycopy(string, 0, params, 1, string.length);
        addTagToMessage(STK_TAG_ITEM, true, params);
}
protected void addTagToMessage(byte tag, boolean comprehension_required,
        byte[] parameters)
{
        if(nextMessage.length == 0) {
                System.out.println("Warning: addTagToMessage() called while "
                                +"nextMessage not present.");
                return;
        }
        if(comprehension_required)
                tag ^= STK_TAG_COMPREHENSION_REQUIRED;
        byte[] prevMessage = nextMessage;
        nextMessage = new byte[prevMessage.length + parameters.length + 2];
        System.arraycopy(prevMessage, 0, nextMessage, 0, prevMessage.length);
        int pos = prevMessage.length;
        nextMessage[pos] = tag;
        nextMessage[pos+1] = (byte)parameters.length;
        System.arraycopy(parameters, 0, nextMessage, pos + 2, parameters.length);
        nextMessage[1] = (byte)(nextMessage.length - 2);
}

public boolean proactiveCommandsAvailable() {
        return proactiveCommandLength() != 0;
}
public byte    proactiveCommandLength() {
        return (byte)nextMessage.length;
}

public byte[] fetch() {
        byte[] next = nextMessage;
        byte[] empty = {};
        nextMessage = empty;
        return next;
}
public void envelope(byte[] envelope) {}
public void terminalResponse(byte[] response) {}
public void terminalProfile(byte[] profile) {}
```

```
        public byte[] simServiceTable(byte[] realSST) {
                return realSST;
        }
};


/**
 * SimToolkitStackMessage - Constantly display an on-screen message. On some
 * phones, this has the effect of a denial of service.
 */
public class SimToolkitStackMessage extends SimToolkitStackDefault {
        private String text;
        SimToolkitStackMessage(String t) {
                this.text = t;
        }

        public void terminalProfile(byte[] profile) {
                if(profile.length < 3) {
                        System.out.println("Warning: terminalProfile is too short "
                                        +"for Proactive SIM. Trying anyway.");
                } else {
                        byte proactiveSupport = profile[2];
                        System.out.println("Proactive support byte: " + proactiveSupport);
                        if((proactiveSupport >>> 7) == 1) {
                                System.out.println("DISPLAY TEXT is supported in terminalProfile.");
                        } else {
                                System.out.println("DISPLAY TEXT is NOT supported "
                                                +"in terminalProfile. Trying anyway.");
                        }
                }
                buildNextMessage(STK_CMD_DISPLAY_TEXT, (byte)0x81);
                addDeviceIdentities(STK_DEVICE_UICC, STK_DEVICE_DISPLAY);
                addTextTag(this.text);
        }

        /**
         * Don't remove the next message on fetch()
         */
        public byte[] fetch() {
                return nextMessage;
        }
};
```