UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS
POSTGRADUATE PROGRAMME DIGITAL SYSTEMS SECURITY

Design and Implementation of an Android Application for extraction of Security Related Data from SIM/USIM

POSTGRADUATE THESIS

Dimitrios G. Raptodimos

Supervisor: Dr. Christos Xenakis
Assistant Professor

Piraeus, May 2013

UNIVERSITY OF PIRAEUS
DEPARTMENT OF DIGITAL SYSTEMS
POSTGRADUATE PROGRAMME DIGITAL SYSTEMS SECURITY

Design and Implementation of an Android Application for extraction of Security Related Data from SIM/USIM

POSTGRADUATE THESIS

Dimitrios G. Raptodimos

Supervisor: Dr. Christos Xenakis
Assistant Professor

Approved by the three-member committee on 23rd May 2013

.....	.....	.....
Dr. Christos Xenakis	Dr. Sokratis Katsikas	Konstantinos Lambrinoudakis
Assistant Professor	Professor	Associate Professor

Piraeus, May 2013

……
Dimitrios G.Raptodimos
Degree in Electrical and Computer Engineering NTUA
Msc in Digital Systems Security University Of Piraeus

The present thesis was co-financed by the project "SSF Scholarships " from funds of the OP "Education and Lifelong Learning", of the European Social Fund (ESF) of NSRF 2007-20013

Η ολοκλήρωση της παρούσας διπλωματικής εργασίας συγχρηματοδοτήθηκε μέσω του Έργου «Υποτροφίες ΙΚΥ» από πόρους του ΕΠ «Εκπαίδευση και Δια Βίου Μάθηση», του Ευρωπαϊκού Κοινωνικού Ταμείου (ΕΚΤ) του ΕΣΠΑ, 2007-2013».

Abstract

In the contemporary society the rapid evolution of communication technologies, raise security and privacy issues for the users of 2G and 3G networks. Many attackers are able to crack or eavesdrop a network and corrupt the communication channel.

The object of the present thesis is to prove that it is feasible to collect security related data using just an application to achieve this. If we analyze these data, then security vulnerabilities of 2G and 3G networks will come to light. Security related data were extracted directly from SIM/USIM module, meaning that we acquired network attributes from the source. The extraction took place by establishing communication with the modem and sending AT Commands to it. Data will be stored and uploaded to a server for further handling.

The application was developed in Android environment, using Android SDK and other development tools such as Eclipse IDE.

***Keywords*:** RIL, SIM, USIM, GSM, UMTS, 3G, 2G, AT Commands, Android, GSM Modem

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor, Assistant Professor Dr. Christos Xenakis, for the chance that he gave me to be part of such a modern and interesting subject, for the continuous support of my study and research, for his patience, motivation and enthusiasm. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my MSc.

Furthermore, I would like to thank my research associate George Paparis for the sleepless nights we were working together, and all my classmates for all the fun we have had in the last two years.

I would like to address special thanks to the State Scholarships Foundation for funding my Master of Science studies.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life.

Dimitrios Raptodimos
May 2013

Table of Contents

Index of Figures

Index of Tables

Index of Screenshots

1

Android Operating System

1.1 Introduction

Android™ is a new mobile operating system and software stack for mobile device. It was initially developed by Android Inc., a firm later purchased by Google, and lately by the Open Handset Alliance. It is totally open source, allowing anyone to develop for it without having to pay fees. That means it is constantly new and evolving, with thousands of new apps available on the Android™ Market, with more to follow.

1.2 Android Architecture

The following diagram shows the major components of the Android operating system. Each section is described in more detail below:

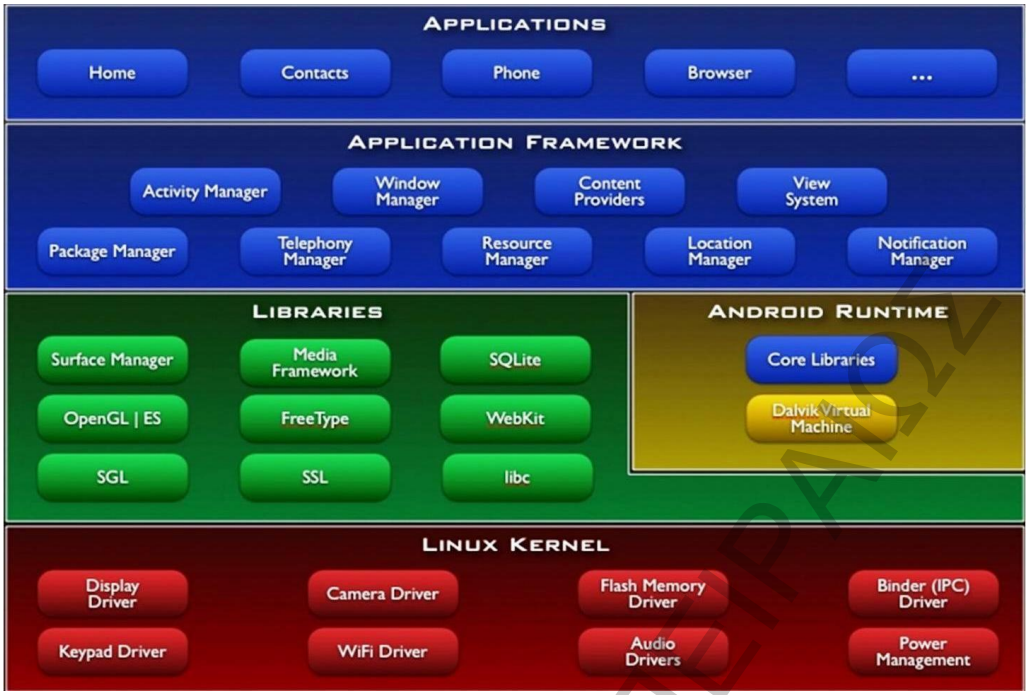


Figure 1. Android Operating System Architecture

1.2.1 Applications

Android includes a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

1.2.2 Application Framework

By providing an open development platform, Android offers developers the ability to build extremely rich and innovative applications. Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, etc.

Developers have full access to the same framework APIs used by the core applications. The application architecture is designed to simplify the reuse of components. Any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user.

1.2.3 Libraries

Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework. Some of the core libraries are listed below:

- System C library - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- Media Libraries - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- Surface Manager - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications
- LibWebCore - a modern web browser engine which powers both the Android browser and an embeddable web view
- SGL - the underlying 2D graphics engine
- 3D libraries - an implementation based on OpenGL ES 1.0 APIs. The libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer
- FreeType - bitmap and vector font rendering
- SQLite - a powerful and lightweight relational database engine available to all applications

1.2.4 Android Runtime

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs

classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management.

1.2.5 Linux Kernel

Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

1.3 Android telephony architecture

All the telephony related applications like Dialer, Call tracker, SMS, MMS, GPRS, Antenna signal indicator and etc., will come into this section. All these applications will be started during the android boot up. These applications will be tied up with the Android telephony framework services. The telephony framework provides APIs to access the Phone.

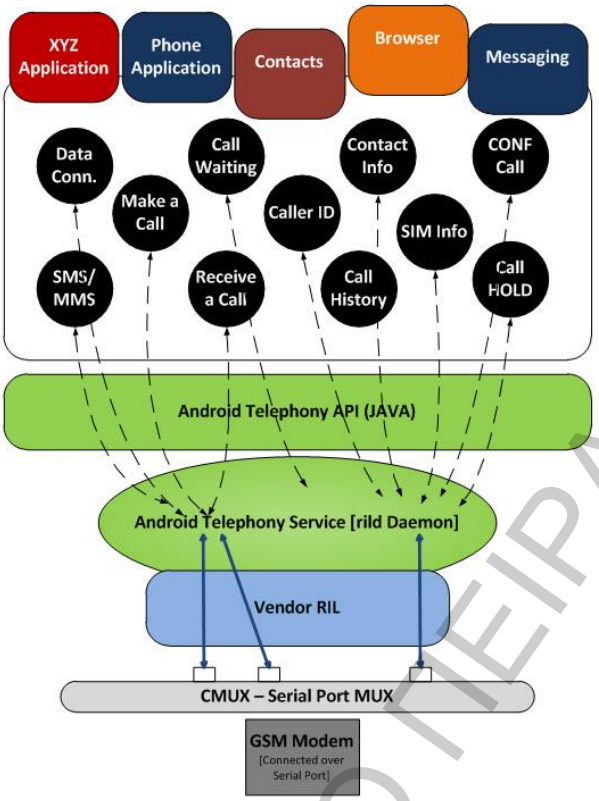


Figure 2. Android RIL Architecture

Telephony framework will be initialized and started during the system start up. All the queries from the application through API will be directed to the Radio interface Layer of Android by these services. The service will keep tracking of all the unsolicited commands from the modem. Unsolicited commands are the commands initiated from the modem.

The bridge between Android phone framework services and the hardware is Radio Interface Layer. In other words, it is the protocol stack for Telephone. The RIL consist of two primary components: RIL Daemon and Vendor RIL.

RIL Daemon (RILD) will be initialized during the Android system start up. It will read the system property to find which library has to be used for Vendor RIL, provide the appropriate input for vendor RIL and finally calls RIL_Init function of Vendor RIL to map all the Vendor RIL functions to the upper layer. Each vendor RIL has RIL_Init function.

Vendor RIL is a library specific to each modem. In other words, we can call it as a driver to function the modem. The RIL daemon will call the RIL_Init function with the device location (eg: /dev/ttyS0). It will initiate the modem and return the RIL_RadioFunctions structure which contains the handles of radio functions.

```
type structure {
intRIL_version;
RIL_RequestFunconRequest;
RIL_RadioStateRequestonStateRequest;
RIL_Supports supports;
RIL_CancelonCancel;
RIL_GetVersiongetVersion;
} RIL_RadioFunctions;
```

These function handles are

- RIL_version : Version of Android RIL
- onRequest : Call to Vendor RIL to make a RIL_REQUEST. It must be completed with a call to RIL_onRequestComplete().It will always be called from the same thread, so returning here implies that the radio is ready to process another command (whether or not the previous command has completed)
- supports: Return current radio state.RADIO_STATE_UNAVAILABLE should be the initial state
- getVersion: Version of Vendor RIL

1.4 Forms of communications in Android RIL

There are two forms of communications in Android RIL: Solicited Commands and Unsolicited Commands.

1.4.1 Solicited Commands

These are commands initiated from the upper layer. Like, Dialing/Send SMS are the solicited commands from the upper layer to the RIL. OnRequest is the function for sending the solicited commands from the upper layer. The following diagram describes the solicited call in Android:

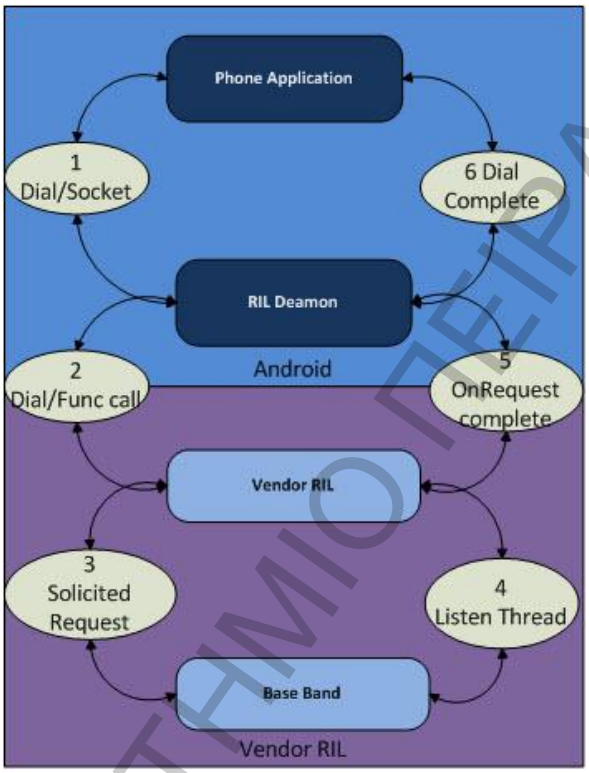


Figure 3. Solicited call in Android

Each onRequest call should end with RIL_onRequestComplete. It is to send the response for the previous onRequest and to intimate we are ready for the next command. Refer ril.h for all the solicited commands

1.4.2 Unsolicited commands

These are the commands initiated from the modem to the upper layer. Like, Receive Call /Receive SMS are the commands. The Vendor RIL has to continuously monitor the device for unsolicited command from the modem. The following diagram describes the unsolicited call in Android:

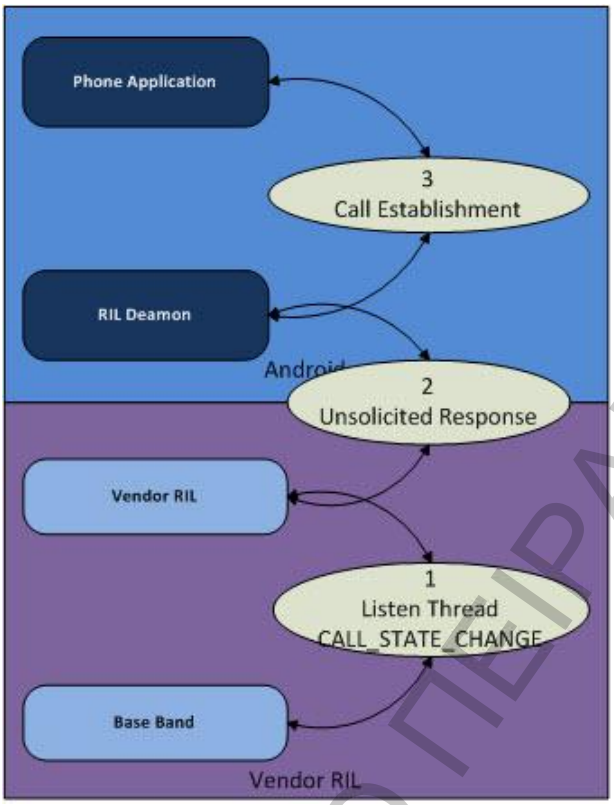


Figure 4. Unsolicited call in Android

2

SIM and USIM Modules

2.1 Introduction

A subscriber identity module or subscriber identification module (SIM) is an integrated circuit that securely stores the international mobile subscriber identity (IMSI) and the related key used to identify and authenticate subscribers on mobile telephony devices (such as mobile phones and computers).

A SIM circuit is embedded into a removable plastic card. This plastic card is called "SIM card" and can be transferred between different mobile devices. SIM cards were first made the same size as a credit card (85.60 mm × 53.98 mm × 0.76 mm). The development of physically smaller mobile devices prompted the development of a smaller SIM card, the mini-SIM card. Mini-SIM cards have the same thickness as full-size cards, but their length and width are reduced to 25 mm × 15 mm.

A SIM card contains its unique serial number (ICCID), international mobile subscriber identity (IMSI), security authentication and ciphering information, temporary information related to the local network, a list of the services the user has access to and two passwords: a personal identification number (PIN) for ordinary use and a personal unblocking code (PUK) for PIN unlocking.

The SIM was initially specified by the European Telecommunications Standards Institute in the specification with the number TS 11.11. This specification describes the physical and logical behaviour of the SIM. With the development of UMTS the

specification work was partially transferred to 3GPP. 3GPP is now responsible for the further development of applications like SIM (TS 51.011) and USIM (TS 31.102) and ETSI for the further development of the physical card UICC [1].

2.1.1 Data

SIM cards store network-specific information used to authenticate and identify subscribers on the network. The most important of these are the ICCID, IMSI, Authentication Key (Ki), Local Area Identity (LAI) and Operator-Specific Emergency Number. The SIM also stores other carrier-specific data such as the SMSC (Short Message Service Center) number, Service Provider Name (SPN), Service Dialing Numbers (SDN), Advice-Of-Charge parameters and Value Added Service (VAS) applications (Refer to GSM 11.11 [10]).

SIM cards can come in various data capacities, from 32 KB to at least 128 KB. All allow a maximum of 250 contacts to be stored on the SIM, but while the 32 KB has room for 33 Mobile Network Codes (MNCs) or "network identifiers", the 64 KB version has room for 80 MNCs. This is used by network operators to store information on preferred networks, mostly used when the SIM is not in its home market but is roaming. The network operator that issued the SIM card can use this to have a SIM card connect to a preferred network in order to make use of the best price and/or quality network instead of having to pay the network operator that the SIM card 'saw' first. This does not mean that a SIM card can only connect to a maximum of 33 or 80 networks, but this means that the SIM card issuer can only specify up to that number of preferred networks, if a SIM is outside these preferred networks it will use the first or best available network[1].

a. International mobile subscriber identity (IMSI)

SIM cards are identified on their individual operator networks by a unique International Mobile Subscriber Identity (IMSI). Mobile network operators connect mobile phone calls and communicate with their market SIM cards using their IMSIs.

The format is: The first three digits represent the Mobile Country Code (MCC).The next two or three digits represent the Mobile Network Code (MNC). Three-digit MNC codes are allowed by E.212 but are mainly used in the United States and Canada.

The next digits represent the Mobile Subscriber Identification Number (MSIN). Normally there will be 10 digits but would be fewer in the case of a 3-digit MNC or if national regulations indicate that the total length of the IMSI should be less than 15 digits[1].

b. Authentication key (Ki)

The Ki is a 128-bit value used in authenticating the SIMs on the mobile network. Each SIM holds a unique Ki assigned to it by the operator during the personalization process. The Ki is also stored in a database (termed authentication center or AuC) on the carrier's network.

The SIM card is designed not to allow the Ki to be obtained using the smart-card interface. Instead, the SIM card provides a function, Run GSM Algorithm, that allows the phone to pass data to the SIM card to be signed with the Ki. This, by design, makes usage of the SIM card mandatory unless the Ki can be extracted from the SIM card, or the carrier is willing to reveal the Ki [1].

c. Location area identity

The SIM stores network state information, which is received from the Location Area Identity (LAI). Operator networks are divided into Location Areas, each having a unique LAI number. When the device changes locations, it stores the new LAI to the SIM and sends it back to the operator network with its new location. If the device is power cycled, it will take data off the SIM, and search for the prior LAI[1].

d. SMS messages and contacts

Most SIM cards will orthogonally store a number of SMS messages and phone book contacts. The contacts are stored in simple "name and number" pairs: entries containing multiple phone numbers and additional phone numbers will usually not be stored on the SIM card. When a user tries to copy such entries to a SIM the handset's software will break them up into multiple entries, discarding any information that is not a phone number. The number of contacts and messages stored depends on the SIM; early models would store as few as five messages and 20 contacts while modern SIM cards can usually store over 250 contacts [1].

2.1.2 SIM/USIM Application structure

The figure below shows the general structural relationships which may exist between files. The files are organized in a hierarchical structure and are of one of the types as defined in 3GPP TS 31.101 [16]. These files may be either administrative or application specific. The operating system handles the access to the data stored in different files.

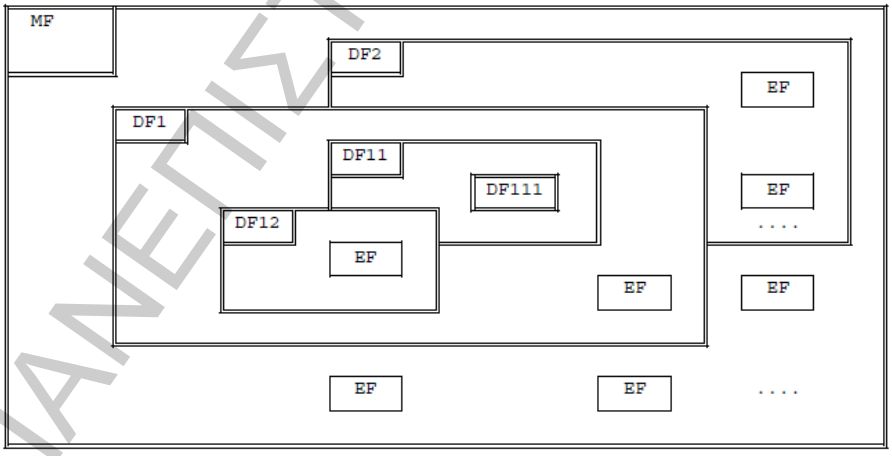


Figure 5. Organization of memory

Files are composed of a header, which is internally managed by the SIM, and optionally a body part. The information of the header is related to the structure and

attributes of the file and may be obtained by using the commands GET RESPONSE or STATUS. This information is fixed during the administrative phase. The body part contains the data of the file [9].

2.2 SIM Card Features

2.2.1 Security features

There are many security aspects concerning GSM. This clause gives information related to security features supported by the SIM to enable the following:

- authentication of the subscriber identity to the network
- data confidentiality over the radio interface
- file access conditions

2.2.2 Authentication and cipher key generation procedure

This clause describes the authentication mechanism and cipher key generation which are invoked by the network. The network sends a Random Number (RAND) to the Mobile Station (MS). The Mobile Equipment (ME) passes the RAND to the SIM in the command RUN GSM ALGORITHM. The SIM returns the values SRES and Kc to the ME which is derived using the algorithms and processes given below. The ME sends SRES to the network. The network compares this value with the value of SRES which it calculates for itself. The comparison of these SRES values provides the authentication. The value Kc is used by the ME in any future enciphered communications with the network until the next invocation of this mechanism [9].

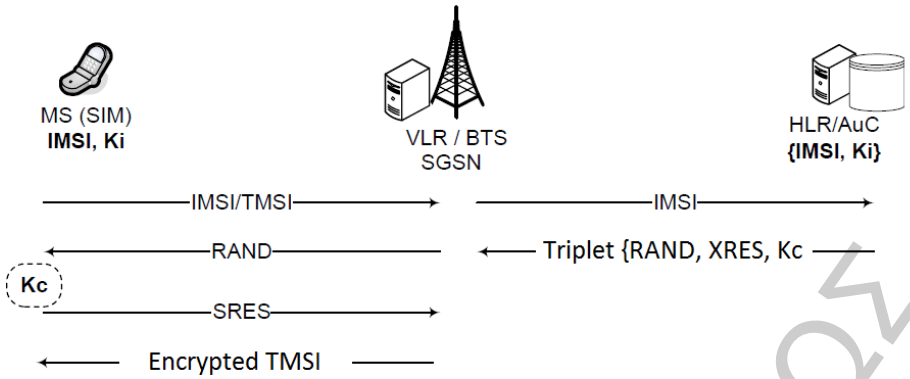


Figure 6. GSM Authentication

A subscriber authentication key Ki is used in this procedure. This key Ki has a length of 128 bits and is stored within the SIM for use in the algorithms described below.

2.2.3 Algorithms and processes

The names and parameters of the algorithms supported by the SIM are the following:

- Algorithm A3 to authenticate the MS to the network
- Algorithm A8 to generate the encryption key.

These algorithms may exist either discretely or combined (into A38) within the SIM. In either case the output on the SIM/ME interface is 12 bytes (96 bits). The inputs to both A3 and A8, or A38, are Ki (128 bits) internally derived in the SIM, and RAND (128 bits) across the SIM/ME interface. The output is SRES (32 bits)/Kc (64 bits) [9].

2.2.4 File access conditions

Every file has its own specific access condition for each command. The relevant access condition of the last selected file shall be fulfilled before the requested action can take place.
For each file:

- the access conditions for the commands READ and SEEK are identical
- the access conditions for the commands SELECT and STATUS are ALWAYS

No file access conditions are currently assigned by GSM to the MF and the DFs. The access condition levels are defined in the following table:

Level	Access Condition
0	ALWAYS
1	CHV1
2	CHV2
3	Reserved for GSM Future Use
4 to 14	ADM
15	NEVer

Table 1. Access condition level coding

The meaning of the file access conditions is as follows:

- **ALWAYS**: The action can be performed without any restriction
- **CHV1:** (card holder verification 1): The action shall only be possible if one of the following three conditions is fulfilled:
    - a correct CHV1 value has already been presented to the SIM during the current session
    - the CHV1 enabled/disabled indicator is set to "disabled"
    - UNBLOCK CHV1 has been successfully performed during the current session
- **CHV2**: The action shall only be possible if one of the following two conditions is fulfilled:
    - a correct CHV2 value has already been presented to the SIM during the current session
    - UNBLOCK CHV2 has been successfully performed during the current session
- **ADM**: Allocation of these levels and the respective requirements for their fulfillment are the responsibility of the appropriate administrative authority. The definition of access condition ADM does not preclude the administrative authority from using ALW, CHV1, CHV2 and NEV if required.
- **NEVER**: The action cannot be performed over the SIM/ME interface. The SIM may perform the action internally.

Condition levels are not hierarchical. For instance, correct presentation of CHV2 does not allow actions to be performed which require presentation of CHV1. A condition

level which has been satisfied remains valid until the end of the GSM session as long as the corresponding secret code remains unblocked, i.e. after three consecutive wrong attempts, not necessarily in the same card session, the access rights previously granted by this secret code are lost immediately. A satisfied CHV condition level applies to both DF_{GSM} and $DF_{TELECOM}$.

The ME shall determine whether CHV2 is available by using the response to the STATUS command. If CHV2 is "not initialized" then CHV2 commands, e.g. VERIFY CHV2, shall not be executable [9].

2.2.5 File Structure of SIM

Elementary Files (EFs) of SIM/USIM modules contain data items. A data item is a part of an EF which represents a complete logical entity. The 3GPP application specification defines the access conditions, data items and coding for each file.

EFs or data items having an unassigned value, or which are cleared by the terminal, shall have their bytes set to 'FF'. After the administrative phase all data items shall have a defined value or have their bytes set to 'FF', unless specified otherwise in other 3GPP specifications. For example, for a deleted LAI in the EF_{LOCI} file defined in TS 31.102 [8], the last byte takes the value 'FE'. If a data item is modified by the allocation of a value specified in another 3GPP TS, then this value shall be used and the data item is not unassigned.

EFs are mandatory (M), optional (O), or conditional (C). A conditional file is mandatory if required by a supported feature, as defined by the 3GPP application. The file size of an optional EF may be zero. All implemented EFs with a file size greater than zero shall contain all mandatory data items. Optional data items may either be filled with 'F', or, if located at the end of an EF, need not exist.

When the coding is according to ITU-T Recommendation T.50 [5], bit 8 of every byte shall be set to 0.

DF_{GSM} shall be selected using the identifier '7F20'. If selection by this means fails, then DCS 1800 MEs shall, and optionally GSM MEs may then select DF_{GSM} with '7F21'. The selection of the GSM application using the identifier '7F21', if selection by means of the identifier '7F20' fails, is to ensure backwards compatibility with those Phase 1

SIMs which only support the DCS 1800 application using the Phase 1 directory $DF_{DCS1800}$ coded '7F21'. To ensure backwards compatibility with those Phase 1 DCS 1800 MEs which have no means to select DF_{GSM} two options have been specified. These options are given in GSM 09.91 [30][9].

In the follow figure the structure of the SIM modules is shown:

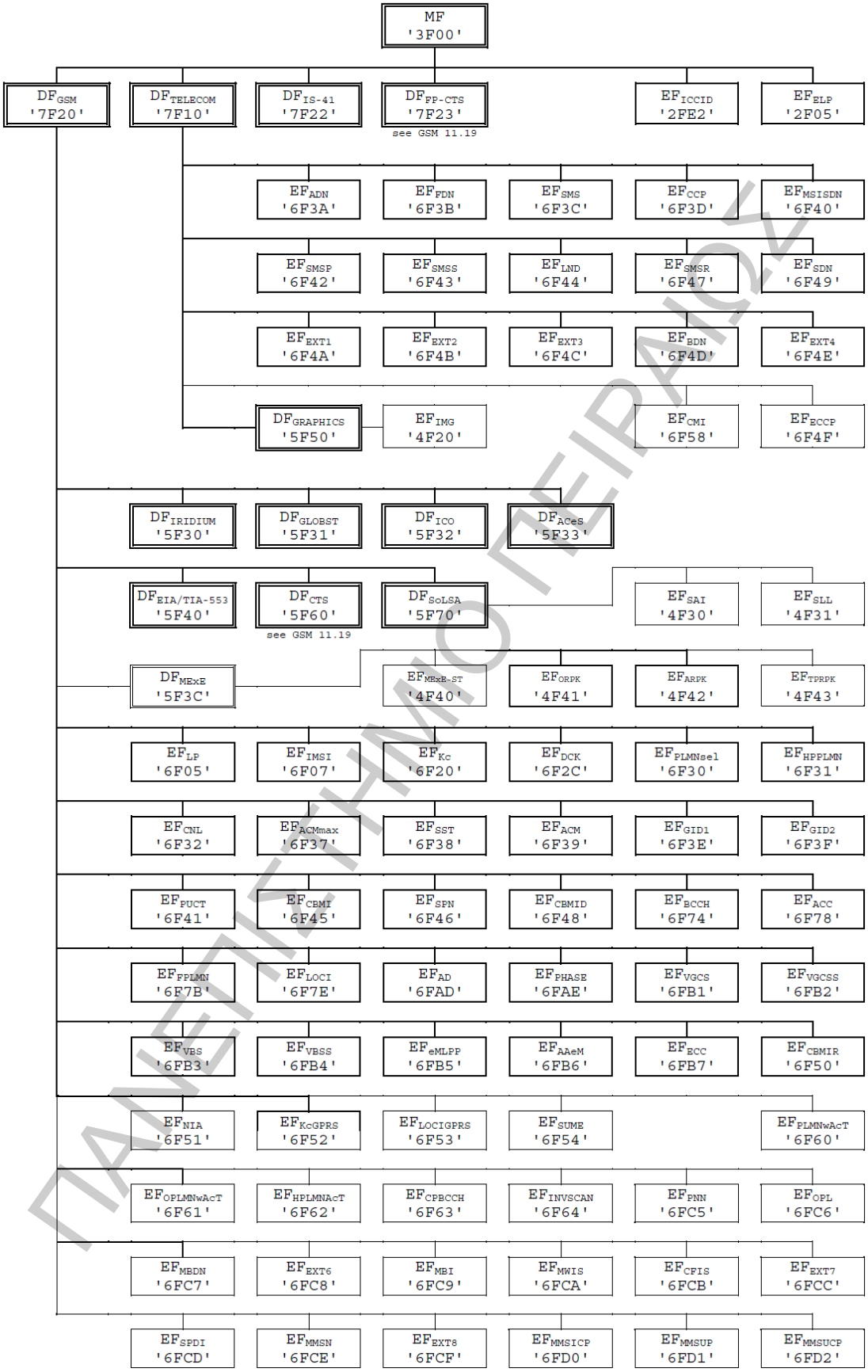


Figure 7. SIM file structure

2.3 USIM Features

2.3.1 Security features

The security aspects of 3G related to security features supported by the USIM to enable the following:

- authentication of the USIM to the network
- authentication of the network to the USIM
- authentication of the user to the USIM
- data confidentiality over the radio interface
- file access conditions
- conversion functions to derive GSM parameters

2.3.2 Authentication and key agreement procedure

This clause gives an overview of the authentication mechanism and cipher and integrity key generation which are invoked by the network.

The mechanism achieves mutual authentication by the user and the network showing knowledge of a secret key K which is shared between and available only to the USIM and the AuC (Authentication Center) in the user's Home Environment (HE). In addition, the USIM and the HE keep track of counters SQN_{MS} and SQN_{HE} respectively to support network authentication. SQN_{HE} is a counter in the HLR/AuC, individual for each user and SQN_{MS} denotes the highest sequence number the USIM has ever accepted [8].

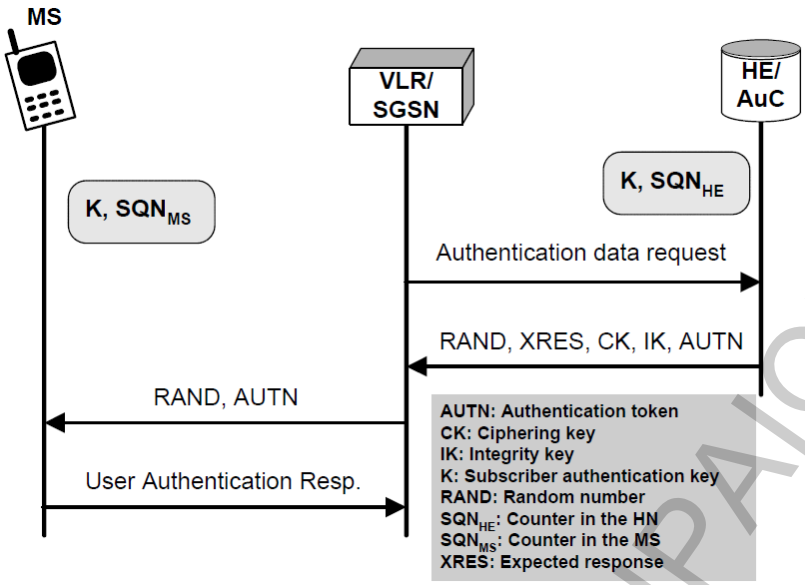


Figure 8. UMTS Authentication

When the SN/VLR initiates an authentication and key agreement, it selects the next authentication vector and sends the parameters RAND and AUTN (authentication token) to the user. Each authentication token consists of the following components: a sequence number SQN, an Authentication Management Field (AMF) and a message authentication code MAC over the RAND, SQN and AMF.

The USIM checks whether AUTN can be accepted and, if so, produces a response RES which is sent back to the SN/VLR. The SN/VLR compares the received RES with XRES. If they match the SN/VLR considers the authentication and key agreement exchange to be successfully completed. The USIM also computes CK and IK. The established keys CK and IK will be used by the ME to perform ciphering and integrity functions.

A permanent secret key K is used in this procedure. This key K has a length of 128 bits and is stored within the USIM for use in the algorithms described below. Also more than one secret key K can be stored in the USIM. The active key to be used by the algorithms is signalled within the AMF field in the AUTN[8].

2.3.3 Cryptographic Functions

The names and parameters of the cryptographic functions supported by the USIM are the following:

- f1: a message authentication function for network authentication used to compute XMAC
- f1*: a message authentication function for support to re-synchronisation with the property that no valuable information can be inferred from the function values of f1* about those of f1, ..., f5, f5* and vice versa
- f2: a message authentication function for user authentication used to compute SRES
- f3: a key generating function to compute the cipher key CK
- f4: a key generating function to compute the integrity key IK
- f5: a key generating function to compute the anonymity key AK (optional)
- f5*: a key generating function to compute AK in re-synchronisation procedures with the property that no valuable information can be inferred from the function values of f5* about those of f1, f1*, f2,..., f5 and vice versa.

These cryptographic functions may exist either discretely or combined within the USIM[8].

2.3.5 Files of USIM

This clause contains a figure depicting the file structure of the UICC and the ADF_{USIM}. ADF_{USIM} shall be selected using the AID and information in EF_{DIR} [8].

Figure 9. File Structure of USIM

3

AT Commands

3.1 General

AT commands is a set of commands that have been initially developed by Hayes Microcomputer Products for the Hayes Smartmodem 300 baud modem in 1981. This was a unique product at the time, because this modem was no longer simply a "dumb" device blindly converting serial data to and from audio tones, but contained some intelligence. It was possible to send commands to the modem to configure it, to execute certain operations (such as dialling a number, quieting the speaker, hanging up, etc.), and to read the current status of the connection. The commands were standardised at some point in time, however, as it is typical with standards, there are several standards. Plus, of course, there are still vendor-specific extensions and implementations in different modems vary slightly. As a result, the command sets of modern modems are not fully compatible with each other. The original Hayes commands, however, should still work, and still form the core of almost all consumer modem command sets.

AT commands are instructions used to control a modem. "AT" is the abbreviation of Attention and every command line starts with "AT" in order to draw the attention of the modem. This is the reason why the Hayes command set is also called AT command set. AT commands are consisted of a series of short text strings. These commands combined together can produce complete commands for operations such as dialing, hanging up, and changing the parameters of the connection. The majority of modems support the Hayes command set. However, given the continuously increasing amount of firmware and baseband devices, many vendor-

specific extensions of AT commands have been developed, giving to each modem supplementary features. In the following clauses we describe the key features of AT commands regarding GSM/GPRS modems [2].

3.2 GSM/GPRS modems AT Command set

3.2.1 Introduction

Besides the common one, GSM/GPRS modems and mobile phones support an AT command set that is specific to the GSM technology and can be categorized as follows:

- Call Control: Commands for initiating and controlling calls
- Data Call Control: Commands for controlling the data transfer and QoS (Quality of Service)
- Network Service: Commands for Supplementary services, Mobile Equipment (ME), operator selection, locking and registration
- SMS Control: Commands for sending, notifying, setting SMS services.
- Mobile Equipment Control & Status: Commands for ME power, keypad, display, phonebook, RTC's

The ETSI GSM 07.07 (3GPP TS 27.007)[13] specifies AT style commands for controlling a GSM phone or modem, along with their syntax and possible modem responses.

3.2.2 Key Tasks Performed by AT Commands

Regarding the categorization described in the previous clause, some of the key tasks that can be done using AT commands with a GSM/GPRS modem or mobile phone are described below:

- Get basic information about the mobile phone or GSM/GPRS modem. For example, name of the manufacturer (AT+CGMI), model number (AT+CGMM),

IMEI number (International Mobile Equipment Identity) (AT+CGSN), and the software version (AT+CGMR).

- Get basic information about the subscriber. For example, MSISDN (AT+CNUM) and IMSI number (International Mobile Subscriber Identity) (AT+CIMI).
- Get the current status of the mobile phone or GSM/GPRS modem. For example, mobile phone activity status (AT+CPAS), mobile network registration status (AT+CREG), radio signal strength (AT+CSQ), battery charge level, and battery charging status (AT+CBC).
- Establish a data connection or voice connection to a remote modem (ATD, ATA, etc.).
- Send (AT+CMGS, AT+CMSS), read (AT+CMGR, AT+CMGL), write (AT+CMGW), or delete (AT+CMGD) SMS messages and obtain notifications of newly received SMS messages (AT+CNMI).
- Read (AT+CPBR), write (AT+CPBW), or search (AT+CPBF) phonebook entries.
- Perform security-related tasks, such as opening or closing facility locks (AT+CLCK), checking whether a facility is locked (AT+CLCK), and changing passwords (AT+CPWD). (Facility lock examples: SIM lock [a password must be given to the SIM card every time the mobile phone is switched on] and PH-SIM lock [a certain SIM card is associated with the mobile phone; to use other SIM cards with the mobile phone, a password must be entered.])
- Control the presentation of result codes / error messages of AT commands. For example, you can control whether to enable certain error messages (AT+CMEE), and whether error messages should be displayed in numeric format or verbose format (AT+CMEE=1 or AT+CMEE=2).
- Get or change the configurations of the mobile phone or GSM/GPRS modem. For example, change the GSM network (AT+COPS), bearer service type (AT+CBST), radio link protocol parameters (AT+CRLP), SMS center address (AT+CSCA), and storage of SMS messages (AT+CPMS).

- Save and restore configurations of the mobile phone or GSM/GPRS modem. For example, save (AT+CSAS) and restore (AT+CRES) settings related to SMS messaging such as the SMS center address.

Note that mobile phone manufacturers usually do not implement all AT commands, command parameters, and parameter values in their mobile phones. Also, the behavior of the implemented AT commands may be different from that defined in the standard. In general, GSM/GPRS modems designed for wireless applications have better support of AT commands than ordinary mobile phones [20].

3.2.3 Main Types and General Syntax of AT Commands

Depending on their type, AT Commands have the corresponding syntax. We can recognize two main types of AT commands, basic commands and extended commands [20]. More specifically:

- Basic commands are AT commands that do not start with "+" such as D (Dial), A (Answer), H (Hook control), and O (Return to online data state).
- Extended commands are AT commands that start with "+". All GSM AT commands are extended commands. For example, +CMGS (Send SMS message), +CMGL (List SMS messages), and +CMGR (Read SMS messages) are extended commands.

The general syntax of extended AT commands is pretty simple. Some of the syntax rules are provided below:

Rule 1: All command lines must start with "AT" and end with a carriage return character, either this character originates from the pressing of the Enter key on the keyboard, either by providing it directly.

```
AT+CGSN<CR>
```

Rule 2: A command line can contain more than one AT command. Only the first AT command should be prefixed with "AT". AT commands in the same command-line string should be separated with semicolons.

```
AT+CGSN; +CGMI<CR>
```

An error will occur if both AT commands are prefixed with "AT".

Rule 3: A string is enclosed between double quotes.

```
AT+CMGL="ALL"<CR>
```

The above command is used to read all SMS messages from a message storage in SMS text mode.

Rule 4: Information responses and result codes (including both final result codes and unsolicited result codes) always start and end with a carriage return character and a linefeed character. A typical example of a mobile device response to +CGMI command, is similar to this:

```
<CR><LF>HTC<CR><LF>
<CR><LF>OK<CR><LF
```

The first line is the information response of the AT command +CGMI, and the second line is the final result code. Note that <CR> and <LF> represent a carriage return character and a linefeed character, respectively. The final result code "OK" marks the end of the response. Hence, the command line "AT+CGMI<CR>" that we entered and the corresponding response will be displayed like this in a terminal program:

```
AT+CGMI (Command line entered)
HTC     (Information response)
OK      (Final result code)
```

3.2.4 Result Code of AT Commands

Result codes are messages sent from the GSM/GPRS modem or mobile phone to provide user with information about the execution of an AT command and the occurrence of an event. Two types of result codes are useful when dealing with AT commands: final result codes and unsolicited result codes [20].

3.2.4.1 Final Result Code of AT Commands

A final result code marks the end of an AT command response. It is an indication that the GSM/GPRS modem or mobile phone has finished the execution of a command line. Two frequently used final result codes are OK and ERROR. Only one final result code will be returned for each command line. Thus, you will not see both OK and ERROR in the response of a command line.

The OK final result code indicates that a command line has been executed successfully by the GSM/GPRS modem or mobile phone. It always starts and ends with a carriage return character and a linefeed character.

The ERROR final result code indicates that an error has been occurred during the execution of a command line from the GSM/GPRS modem or mobile phone. After the occurrence of an error, the GSM/GPRS modem or mobile phone will stop processing the remaining AT commands in the command-line string. Some of the common causes of the occurrence of an error are listed below:

1. The syntax of the command line is incorrect.
2. The value specified to a certain parameter is invalid.
3. The name of the AT command is spelt incorrectly.
4. The GSM/GPRS modem or mobile phone does not support one or more of the AT commands, command parameters, or parameter values in the command-line string.

Like the OK final result code, the ERROR final result code always starts and ends with a carriage return character and a linefeed character.

As far as SMS AT commands are concerned, the +CMS ERROR final result code, notifies the user about the occurrence of a message service failure. Unlike OK and ERROR, the +CMS ERROR final result code is only available to SMS AT commands[20].

3.2.4.2 Unsolicited Result Codes of AT Commands

Finally, unsolicited result codes are messages sent from the GSM/GPRS modem or mobile phone to provide information to the user about the occurrence of an event. For example, the user can use the +CNMI AT command (command name in text: New Message Indications to TE) to request the GSM/GPRS modem or mobile phone to send the unsolicited result code "+CMTI" to user's computer / PC every time a new SMS message is received from the SMS center [20].

4

Functional Requirements and Design of SimMonitor Application

4.1 Application Description

For the purpose of the current thesis an application has been developed, which is able to collect security related data from SIM and USIM cards. For the development of the application several stages took place, such as detecting modem devices of an android phone, determining the security data that we would like to collect, establishing the communication between the modem and the application and finally build the application using the knowledge described in previous chapters. In the next clauses, the functional requirements of the application are described, as well as data requirements and architecture.

4.2 Functional Requirements

a. Security Data Extraction from SIM/USIM card

The application must be able to extract security related data from SIM/USIM modules, by communicating with the modem of the phone. The data that we would like to collect are the subscriber's identity (International Mobile Subscriber Identity - IMSI), ciphering keys, temporary identities that are transmitted to air and finally network based information concerning cell locations and data routing. The process

of data collection should take place at specific time intervals defined by the user as well as every time the screen turns off and on, the phone is powered off, or a call connection is established. Finally, the collection of data should be as transparent as possible to the user and in no case should prevent the proper use of its phone.

b. Metadata and Sensors Data Collection

Along with the extracted data from SIM/USIM modules, the application should be able to collect some metadata which will help future analysis. These data concern of the network provider, network type, SIM/USIM module roaming, the event identifier that triggered data collection and the timestamp of the process. Finally, the exact location of the mobile station will be acquired using GPS sensor and in case of no availability Wi-Fi access points and cell towers by means of a network lookup shall be used.

c. Parsing of Data

After the collection of all data and metadata, the parsing of them should take place in order to extract the desired information. The metadata will be attach to every piece of information in order to form an understandable format of data. Afterwards, parsed data will be delivered for further handling.

d. Data Storage and Display

The application should be able to store the parsed data into a database. Furthermore, it should be able to display the parsed data on the screen by user's choice. All the inserted into database records, should be at any time available for exporting and further handling.

e. Data Uploading

The application should give to the user the option to upload the records of the database to a secure server via SSH. After upload completes, the application should remove all uploaded records from the database and store them in a special history folder of internal memory, to prevent possible data loss. The server IP address, as well as server credentials, should be selected and inserted by the user.

f. Application Settings

The application should provide some settings that concern data collection process, such as the interval between two points in time where data collection took place. Furthermore, settings concerning the upload of data should be provided, such as the IP address of the server, credentials and server directory in which data will be uploaded.

5

SimMonitor Application Implementation

5.1 Introduction

As mentioned in previous chapters, the main purpose of the present thesis is the implementation of an android application, which will extract from SIM/USIM cards and collect security related data. In this chapter the implementation process is described, as well as the tools that were used to achieve our goal.

5.2 Preparation Process

Before starting the development of the application, some stages needed to take place. All development devices needed to be "rooted", in order to be able to gain super user privileges. Finally, the tools that would be used for the development needed to be gathered and installed.

5.2.1 Rooting the device and disabling security features

In Android devices, the vendors do not allow to a user to communicate directly with the modem due to security reasons. As a result a process must be followed in order to give to the phone Super User (root user) privileges, which is known as "rooting of the device". As Android was derived from the Linux kernel, rooting an Android device

is similar in practice to accessing administrative permissions on Linux or any other Unix-like operating system such as FreeBSD or OS X.

The process of rooting varies widely by device, but usually includes exploiting a security flaw of the firmware of the device, and then copying the su binary to a location in the current process's PATH (e.g. /system/xbin/su) and granting it executable permissions with the 'chmod' command. A supervisor application like SuperUser or SuperSU can regulate and log elevated permission requests from other applications. Many guides, tutorials, and automatic processes exist for popular Android devices facilitating a fast and easy rooting process.

In addition, in some devices extra security features must be disabled. For example, HTC devices must be S-OFF (security off) in order to be able to communicate with the modem [3].

5.2.2 Tools and devices that were used

The application implemented using the following tools:

- Java programming language
- Eclipse IDE (Integrated Development Environment)
- Android SDK (Standard Development Kit)
- Eclipse ADT plug-in
- Shell programming
- JSch (Java Secure Channel) library

The phones, in which the implementation and experiments took place, are the following:

- Samsung S-5500
- Samsung S-6500
- ZTE Blade
- HTC Sensation XE with Beats Audio

5.3 Data Collection Process

After having determined the data types and metadata that will be collected, the next step is to actually collect data. As mentioned in the previous chapter, our main purpose is to extract the required data directly from SIM/USIM module. In order to achieve this, we have to communicate with the modem of the mobile phone and send the corresponding to data AT commands. Capturing and parsing the responses of the modem to these commands, we were able to collect the required data for further analysis. In the following clauses the communication with the modem and data collection processes are described.

5.3.1 Detecting and communicating with the modem

On a modern Android based "smart phone", there are essentially two processors: the Application Processor (AP) that is used to run Android Operating System (AOS) and user interface (UI), and the Baseband/Cellular Processor (BP/CP) where all the GSM and other communication operations take place, including the modem we wish to communicate with. In the most modern phones the BP and the AP and all possible other peripheral devices are integrated into one piece of hardware, loosely known as a Smartphone or System on a Chip (SoC). On this SoC there are a number of peripheral devices such as RTC, UARTs, SPI, I2C, USB ports, SD/MMC card controllers and an ISO7816 SIM card reader. However, to preserve the layered hardware structure, the AP and BP still communicates via UART (serial line), USB, SPI or through shared RAM and/or a combination of these. Therefore, there will always be a path directly accessible from the outside that one could use to communicate directly with the BP. Exactly how this is done, is mostly unknown due to the closed source, to the great dismay of the developer community [19].

The first step is to find the correct serial device to communicate with the android phone modem. In an android phone, one could find hundreds of devices listed under /dev directory[1]. Knowing which one is the serial device used for communicating with

[1] We should keep in mind that Android is a Linux-based operating system.

the Baseband Processor's (BP) Modem, is key in achieving a useful AT communication with it. Furthermore, one should take into account that there are several serial devices connected to the BP. These connections are working in parallel through a multiplexer (MUX). So it is very likely that several different devices could be used to send AT commands to BP modem [19].

Therefore, in order to detect the correct serial device we could try to connect via some terminal application to all devices and send some AT commands, looking for a response. Nevertheless, this method is neither very practical nor efficient.

For this reason, a good practice is to examine the build.prop file of an android device which is located in /system directory and contains several build properties of the AOS. Among all other, there is a block of properties that refer to android RIL and contains not only the libraries that the vendor RIL should use, but also the serial device that is used from RIL to communicate with the Baseband Modem. A typical example of the block of RIL properties listed in android /system/build.prop file, are shown below:

```
rild.libpath=/system/lib/libsec-ril.so
rild.libargs=-d /dev/smd0
ro.sf.lcd_density=240
dalvik.vm.heapsize=64m
ro.opengles.version=131072
```

From the above properties, we reach the conclusion that the vendor RIL uses /system/lib/libsec-ril.so library which includes all modem operations and the serial device used is smd0, which is located in /dev directory.

In order to verify that the modem device is the one we discovered above and additionally taking into account that many devices do not include rild.libargs property in build.prop file (like most HTC devices), we have to detect the serial devices by listing all the available tty drivers:

```
# cat /proc/tty/drivers

/dev/tty             /dev/tty        5  0       system:/dev/tty
msm_serial_hsl       /dev/ttyHSL   241  0-5     serial
msm_serial_hs_brcm   /dev/ttyHS    242  0-1     serial
pty_slave            /dev/pts      136  0-1048575 pty:slave
pty_master           /dev/ptm      128  0-1048575 pty:master
pty_slave            /dev/ttyp       3  0-255   pty:slave
pty_master           /dev/pty        2  0-255   pty:master
smd_tty_driver       /dev/smd      250  0-36    serial
```

Examining the above results we are able to verify that the serial device representing the modem is /dev/smd0. As mentioned before, if one is not able to detect the modem from build.prop file due to the lack of rild.libargs property, then he should try to communicate with all the devices that are listed under /proc/tty/drivers file and are marked as "serial".

After identifying the device to communicate with the Baseband Modem, the next step is to actually communicate with the latter by sending AT commands. In order to achieve this we have to execute some commands:

To get the response from the modem we execute cat command. The command and the result of it are shown below:

```
# cat /dev/smd0 &
cat /dev/smd0 &
[1] 3931
```

To send an AT command to the modem we will use echo command. The AT command that we will send is ATI, which lists the Manufacturer, Model, Revision and IMEI of the phone. The command and the response from modem are shown below:

```
# echo -e 'ATI\r' > /dev/smd0
echo -e 'ATI\r' > /dev/smd0
ATI
Manufacturer: SAMSUNG ELECTRONICS CORPORATION
Model: GT-S6500
Revision: S6500BULK2
IMEI: 354312052096915
```

5.4 The manifest file of the Application

Every android application must have an AndroidManifest.xml file in its root directory and thus SimMonitor application has one too. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. The manifest file of our application along with the description of its significant components, are shown below:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.dg.simmonitor"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="15" />

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
    <uses-permission android:name="android.permission.GET_TASKS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_UPDATES" />
    <uses-permission android:name="android.permission.ACCESS_GPS" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.dg.simmonitor.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="com.dg.simmonitor.SettingsActivity"
            android:label="@string/settings"
            android:parentActivityName="com.dg.simmonitor.MainActivity" >
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"
                android:value="com.dg.simmonitor.MainActivity" />
        </activity>
        <service android:name=".MonitorService" />
    </application>
</manifest>
```

The "package" attribute of manifest element, names the java package for the application. The package name server is a unique identifier for the application. As shown to the figure above, our package name is "com.dg.simmonitor".

The <uses-sdk> element along with its attributes, declares the minimum level of the Android API that the application requires. Our minimum Android API support, as shown, is for API level 8 which corresponds to Android 2.2.x version (Froyo), while the target API level is 15 which corresponds to Android 4.0.x version (Ice Cream Sandwich - ICS).

Under <activity> elements, all the application activities are listed. As we can see, the application consists of two activities: MainActivity which is the starting activity and SettingsActivity, which includes application settings. It is worth noting the existences of *android:parentActivityName* attribute under SettingsActivity activity element, which denotes the parent activity, namely the one from which it is called.

Under the <service> element, our application service is declared. The latter, named SimMonitorService, simulates the whole processing unit as shown in *figure 11*. The processing unit has been implemented with a service, because it has to run even when the application doesn't run in the foreground.

It has to be mentioned that, the Events Receiver unit has been implemented using the Android BroadcastReceiver class. This should be normally declared in the manifest file, but taking into account that the Events Receiver runs locally into the implemented service, there is no need to declare it publically into the manifest.

Finally, the most important sector of the android manifest is the permissions declaration. Permissions are used in order to limit access to a protected feature without the user consent. Therefore, when the application is installed on the device, the installer determines whether or not to grant the requested permission by checking the authorities that signed the application's certificates and, in most cases, asking the user. The requested permissions are declared under the <uses-permission> element. As one can see in the manifest file the permissions that the application demands in order for its features to work properly are the following:

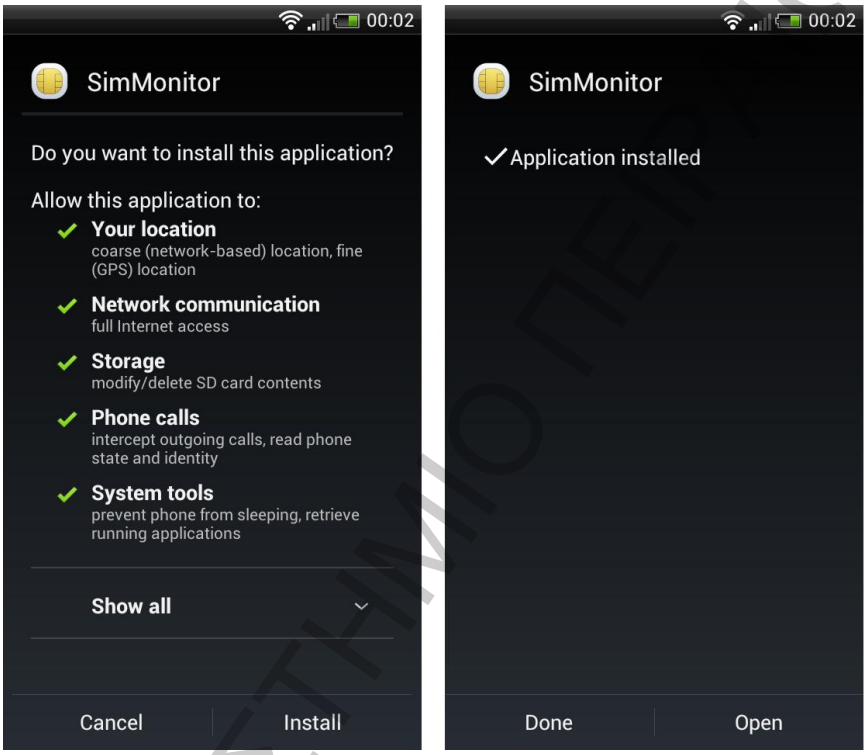
- android.permission.INTERNET: Allows to application to open network sockets. This is required for the uploading of data to the secure server.

- android.permission.ACCESS_NETWORK_STATE: Allows applications to access information about networks. This is used to acquire the described metadata, such as network type, network location etc.
- android.permission.WAKE_LOCK: Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming. This is used because if the processor go to sleep mode, then the AOS pauses the execution of the service, leading to data collection interruption.
- android.permission.GET_TASKS: Allows an application to get information about the currently or recently running tasks. This is used to check whether the application is running or not.
- android.permission.READ_PHONE_STATE: Allows read only access to phone state. This used in some administrative operations.
- android.permission.PROCESS_OUTGOING_CALLS: Allows an application to monitor, modify, or abort outgoing calls. This is used from the Telephony Listener in order to detect if an outgoing call is taking place.
- android.permission.ACCESS_COARSE_LOCATION: Allows to access approximate location derived from network location sources such as cell towers and Wi-Fi.
- android.permission.ACCESS_COARSE_UPDATES : Allows to get continuous location updates from network location sources.
- android.permission.ACCESS_GPS : Allows to use GPS sensor.
- android.permission.ACCESS_FINE_LOCATION: Allows an app to access precise location from location sources such as GPS, cell towers, and Wi-Fi.

5.5 User Interface and Mode of Operation

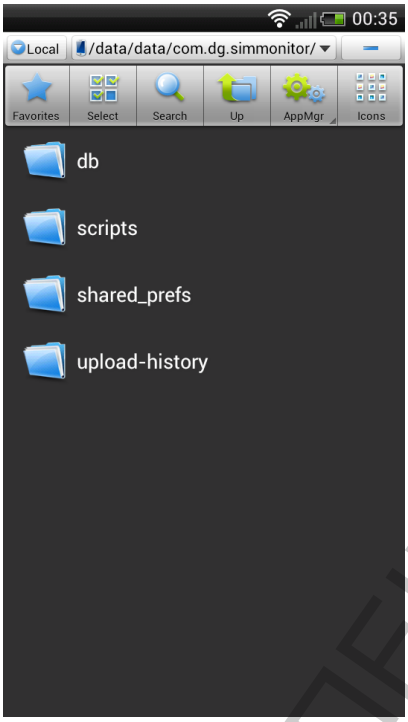
5.5.1 Main Screen

The first step is to install the application. Before the execution of this process, the installer interprets the permissions defined in the manifest file and asks for user's consent. The screenshot below shows this process:



Screenshot 1. Application Installation

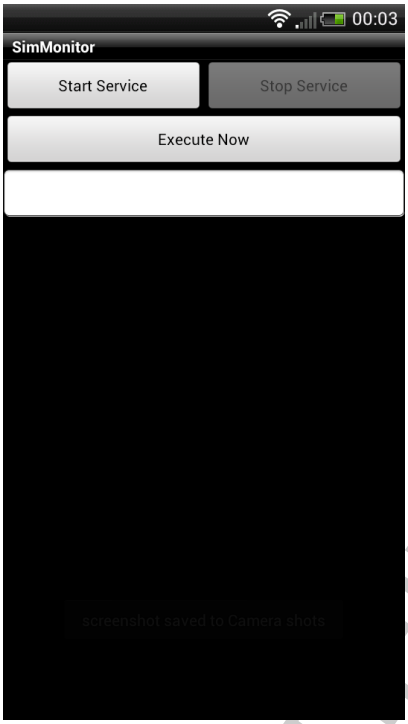
After user's consent, the installation process begins. During that process, the application directories are created. For each Android application, a directory for administrative reasons is created under directory /data/data/<application_name>, and thus our directory is /data/data/com.dg.simmonitor. Under this directory we create the following directory structure:



Screenshot 2. Application File Structure

In shared_prefs folder, the user settings as XML files are saved. The user settings will be described later. In scripts folder, the scripts that run the communication with the modem are copied during installation, while in db folder the database .csv file is saved. Finally, in upload-history the uploaded data are saved in order to keep a history in case of data loss. The user is able to delete history manually whenever he wants.

After the installation, the user is able to start the application. When started the main activity is shown:



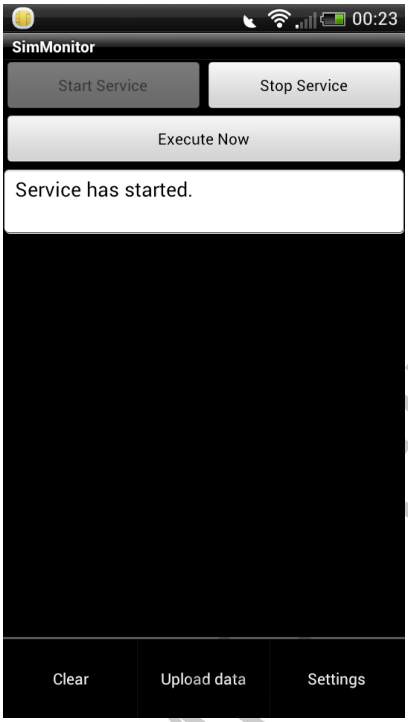
Screenshot 3. Main Activity

As we can see there are three buttons:

- "Start Service" Button: This button starts the service which simulates the whole process. From that time and then, the application starts data collection as described in chapter 4. The service and thus the collection of data never stops, until this is set by the user explicitly, either by pressing "Stop Service" button or by powering off the device.
- "Execute Now" Button: This button executes the whole process immediately and prints the results on the screen. In case the service hasn't started, after the first execution it does.
- "Stop Service" Button: It stops the services and disposes all the resources.

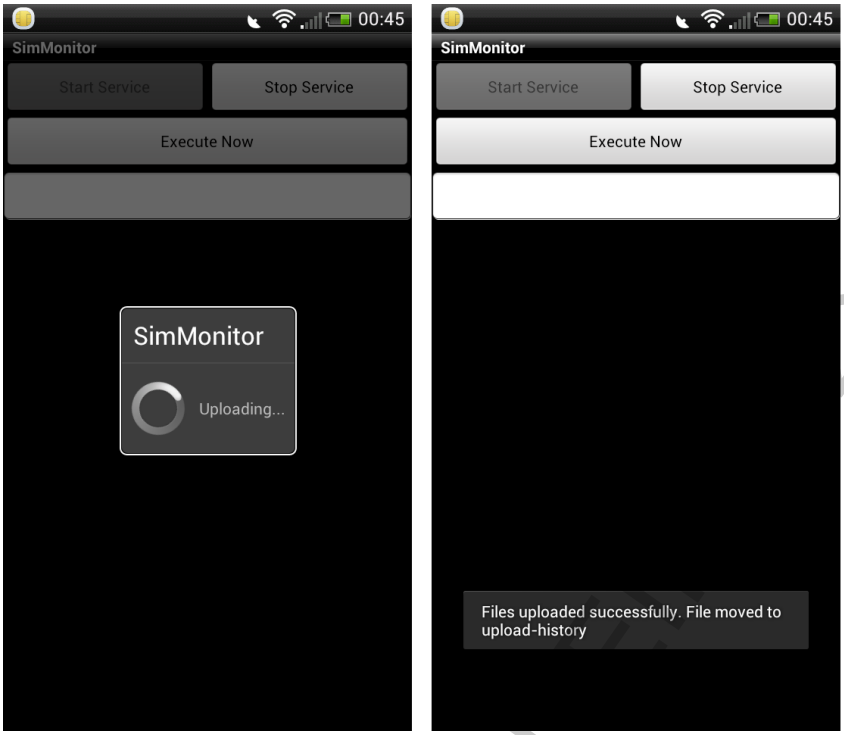
5.5.2 Application Menu

As shown in the following screenshot the application menu consists of three buttons.



Screenshot 4. MainActivity, service started

"Clear Button" clears the text with the results that is originated by pressing the "execute Now" button. "Upload data" button uploads data to the secure server where we store all data from all of our experiments. Below the screenshots from data uploading are shown:

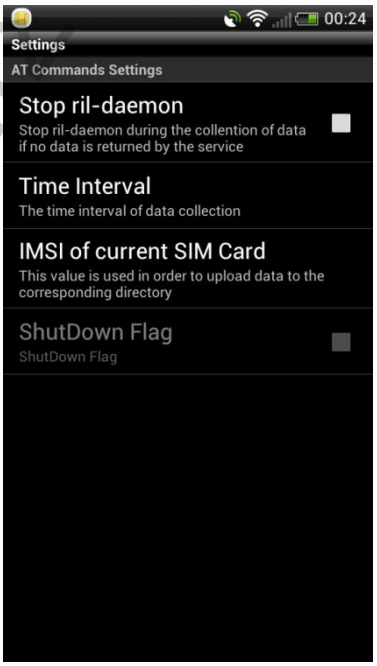


Screenshot 5. Data Uploading

Finally, "Settings Button" starts the setting activity. Settings are described in the next clause.

5.5.3 Settings Screen

Below application settings are displayed:



Screenshot 6. Application Settings Activity

The stop ril-daemon setting, defines whether the RIL daemon of the phone will be stopped during data collection and started again after the completion of process. As described in clause 5.3.1, some phones are not able due to vendor's baseband implementation to communicate with more than one threads at a time. If the user realizes something like that when the "execute now" button is pressed, then it is mandatory for this option to be set to true.

Time interval option defines the interval in which two periodic data collection processes take place. The default value of this option is set to 5 minutes. When user decides to change this option, he should take into account the great amount of data collected, the power loss and modem continuous operation, if he decides to set a short time interval.

"IMSI of current SIM card" option, is used in order to define the directory name of the secure server in which data will be uploaded. This option is set automatically after the first execution of data collection and parsing. However, the user is able to change it, if he wants to define a different directory name.

6

Conclusions - Related Work

In the present thesis, we were able to collect and store some security related data. These data are going to be the subject of further analysis in future, in order to draw conclusions concerning the implementation of security aspects on 2G/3G networks by network providers.

Related works focus on cracking the encryption algorithms of SIM/USIM modules, as well as ways of executing eavesdrop and man in the middle attacks. Our work focuses clearly on different perspective, demanding only a phone and a user and no other kind of equipment. By analyzing the collected data, we will be able to reveal well hidden network features concerning the way encryption keys and temporary identities are produced and possibly reused, leading as eventually to communication crack.

7

References

[1] Subscriber identity module, Wikipedia, http://en.wikipedia.org/wiki/Subscriber_identity_module

[2] Serial Programming/Modems and AT Commands, Wikibooks, http://en.wikibooks.org/wiki/Serial_Programming/Modems_and_AT_Commands

[3] Android rooting, Wikipedia, http://en.wikipedia.org/wiki/Android_rooting

[4] Universal Mobile Telecommunications System, Wikipedia, http://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System

[5] GSM, Wikipedia, http://en.wikipedia.org/wiki/GSM

[6] Hayes command set, Wikipedia, http://en.wikipedia.org/wiki/Hayes_command_set

[7] ETSI TS 131 102 V8.4.0 (2009-01), Universal Mobile Telecommunications System (UMTS), LTE, Characteristics of the Universal Subscriber Identity Module (USIM) application (3GPP TS 31.102 version 8.4.0 Release 8)

[8] ETSI TS 151 011 V4.15.0 (2005-06), Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface (3GPP TS 51.011 version 4.15.0 Release 4)

[9] 3GPP TS 11.11, Mobile Equipment (SIM - ME) interface

[10] 3GPP TS 33.102, 3G Security, Security Architecture\

[11] ETSI TS 100 977 V8.3.0 (2000-08), Digital cellular telecommunications system (Phase 2+), Specification of the Subscriber Identity Module, Mobile Equipment (SIM - ME) interface

[12] 3GPP TS 27.007 V11.5.0 (2012-12), 3rd Generation Partnership Project, Technical Specification Group Core Network and Terminals, AT command set for User Equipment (UE)(Release 11)

[13] 3GPP TS 27.007 V2.0.0 (1999-06), 3rd Generation Partnership Project, Technical Specification Group Terminals, AT command set for 3GPP User Equipment (UE)

[14] ETSI TS 102 221 V9.0.0 (2010-02), Smart Cards, UICC-Terminal interface, Physical and logical characteristics (Release 9)

[15] ETSI TS 131 101 V11.0.0 (2012-11), Universal Mobile Telecommunications System (UMTS), LTE, UICC-terminal interface, Physical and logical characteristics (3GPP TS 31.101 version 11.0.0 Release 11)

[16] SIM Forensics, Forensic Magazine, August 05, 2011, http://www.forensicmag.com/article/sim-forensics-part-3?page=0,1

[17] End-to-End Encrypting Android Phone Calls, I. Burns, K. Gabert, and J. Zheng, Department of Computer Science and Engineering, New Mexico Institute of Mining and Technology

[18] How to talk to the Modem with AT commands, xda developers, http://forum.xda-developers.com/showthread.php?t=1471241

[19] Introduction to AT commands and its uses, Code Project, http://www.codeproject.com/Articles/85636/Introduction-to-AT-commands-and-its-uses

[20] Android Developer, Official Android Development Site, http://developer.android.com

[21] Android Platform Development Kit, Radio Layer Interface, Netmite, http://www.netmite.com/android/mydroid/development/pdk/docs/telephony.html

[22] Android RIL Architecture, e-consystems.com, http://www.e-consystems.com/blog/android/?p=498

[23] Working with the Radio Layer Interface (RIL) in Android, afewe, http://afewe.wordpress.com/android-arm-development/working-with-the-radio-layer-interface-ril-in-android

[24] Android Application Development, Rick Rogers, John Lombardo, Zigurd Mednieks, G. Blake Meike, O'Reilly Media, May 13, 2009

[25] AT Command Set in Gobi, Application Note, Qualcomm, February 4, 2010, http://mod-book.ru/files/Gobi2k/Documents/AT_Command_Set_Gobi.pdf

[26] AT commands for Sony Ericsson phones, Sony Ericsson, http://dl-developer.sonymobile.com/documentation/DW-65054-dg_at_2006--10_r17a.pdf