University Of Piraeus

Department Of Digital Systems

Postgraduate Programme

“Techno-Economic Management and Digital Systems Security”

MSc Thesis

“Security Assessment of Mobile Networks by Data Extraction from SIM Cards via AT Commands”

Student: Andi Anastasis

Supervisor: Assistant Professor Xenakis Christos

Piraeus

June 2012

Acknowledgements

The author would like to thank Assistant Professor Christos Xenakis for his guidance and his experienced input on the subject, as well as for the trust he showed on this subject. Also thanks to Dr Christoforos Dadoyan and Stefanos Malliaros for their valuable input and their elaboration on the project.

Abstract

In the last two decades the growth of the mobile phone industry has been significant. This growth has led to the usage of mobile devices for everyday actions such as communication, banking, networking etc. It is common knowledge that the more a service is used, the more attackers will try and exploit it. The improvement of computers' processing power has made security measures, which once were adequate, easy to overcome. Moreover, mobile service providers are not up-to-date with the latest standards mainly for two reasons. The first reason is that mobile carriers purchase their equipment and once it is installed it acts as a black box, meaning that the provider does not interfere with the equipment's internals. The second reason is that in order to improve security for the mobile services the provider offers, there need to be a tradeoff and waste more resources. The need for more processing power will lead to a more expensive functioning of the infrastructure and this means less profit for the provider.

This thesis documents the attempt to extract information about the mobile networks from the mobile's SIM card. With the use of AT Commands it was made possible to extract security related data and perform a security assessment in the current state of the mobile service networks. Using custom made tools, a database of logs was built and processed in order to arrive to conclusions.

Index

1. Environment

In order to conduct the necessary experiments a suitable environment must be first set up. The most suitable environment in which the testing took place consists of:

- **iPhone**: The devices chosen were Apple's iPhone 4 GSM with iOS version 5.1.1 and iPhone 3GS with iOS version 6.
- **Jailbreak**: The devices underwent the jailbreaking process in order to get root access to the phone. This process removes the limitations set by Apple for the iOS by the use of certain exploits [2].It is a necessity in order to install additional tools that will help with the experiments and are not available through the official App Store. The needed components are available via Cydia. Cydia is an alternative software distribution channel for applications, tweaks and themes for jailbroken iOS devices [4].
- **OpenSSH**: OpenSSH (OpenBSD Secure Shell) is a collection of network programs providing encrypted communication sessions in a network via the Secure Shell (SSH) protocol [3].It should be easily installed via Cydia.
- **Minicom**: Minicom is a text-based program used for modem control and terminal emulation for Unix-based operating systems [5]. One of its main uses is serial communication with a device. Minicom is the most essential program for the experiment as it utilizes the communication between the user and the phone's modem. Through minicom we can send commands to the iPhone's baseband and get replies.
- **Ruby**: Ruby is a dynamic, object-oriented, open-source programming language [6]. It is well-known for its simplicity and its natural syntax making it easy to read and comprehend.In order to automate the process of data extraction a script language is very helpful. The choice was ruby, due to previous experience and the simplicity of the language.
- **Signal**: Signal is a utility that displays information about cell towers the device is connected, as well as neighbour towers along with signal strength in dBm and the tower's location in a map. In some devices communication via minicom is not possible, probably due to an issue with the X-Gold 618 baseband [8]. It is suggested that the "Signal" app is installed via Cydia in order to achieve a successful communication with the iPhone serial ports.
- **adv-cmds:** The adv-cmds (short for advanced commands) is a set of unix commands ported for the iOS [20]. The command required is the "ps" (process status) which lists all running processes in the background. This is used in order to check whether an AT Command is still running in the background.
- **Core Utilities:** This package is not necessary for the main process but it is required for monitoring live the process via the "tail" command. The logs will still be saved in the device and will they can be read with any text viewer or editor.

2. Setting Up

The technical steps in order to set up the environment are described below.

2.1 Jailbreak

There are numerous of up-to-date tutorials on how to jailbreak your iPhone device. The jailbreaking process will not be described more and will be considered as a prerequisite.

2.2 Programs' Installation

After opening the Cydia app, searching for the packets "OpenSSH", "minicom", "Signal", "adv-cmds", "core utilities" and "ruby" and installing is a simple process.

2.3 Setting up minicom

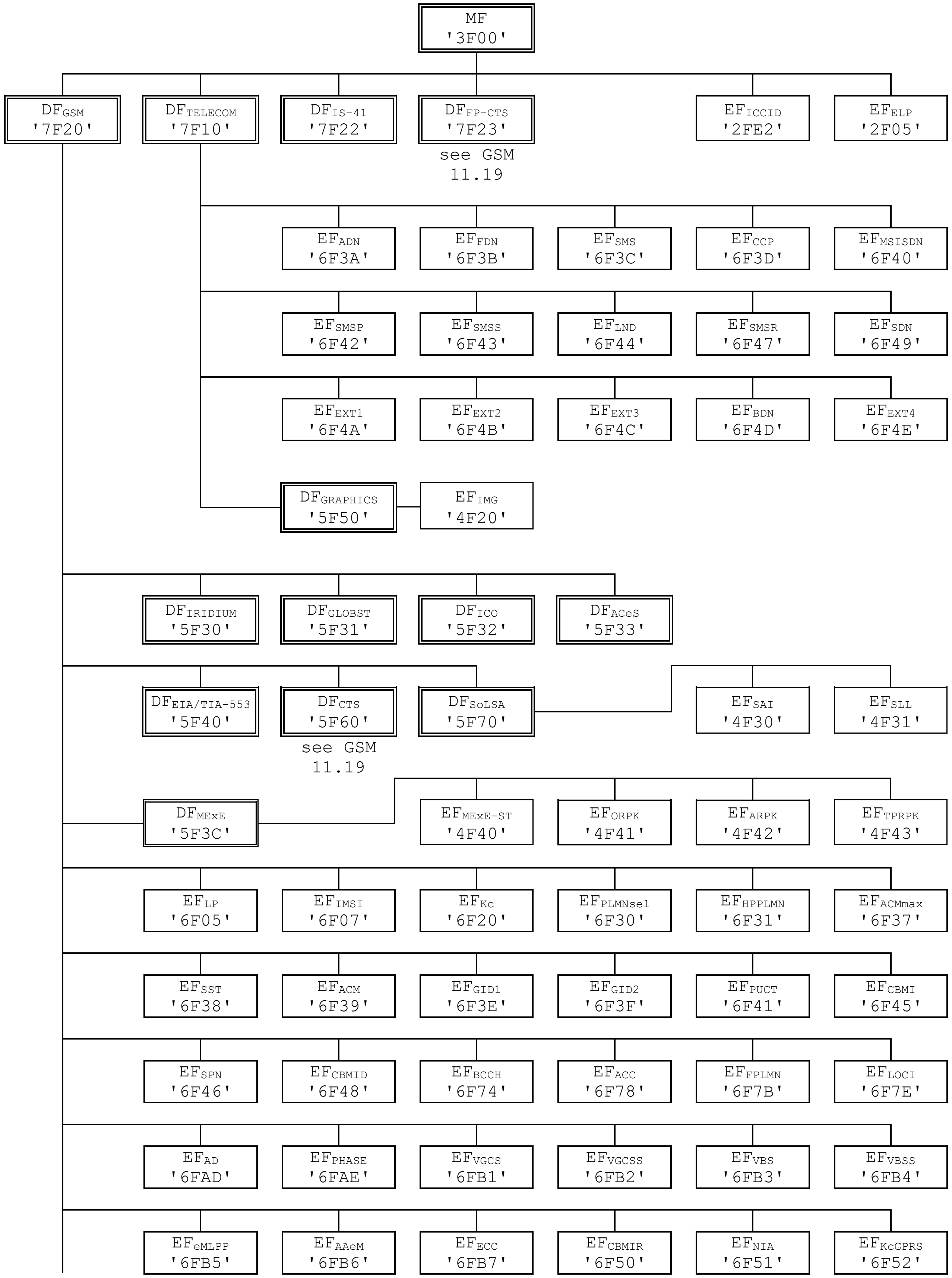
A very detailed guide on how to set up minicom is located at the LetsUnlockIphone [8] site. A recommended port should be the "cu.debug" instead of the "tty.debug". You can start minicom by typing "minicom -w -c on" in a terminal. When a message "AT" is send and a reply "OK" is received, it is verified that minicom works.

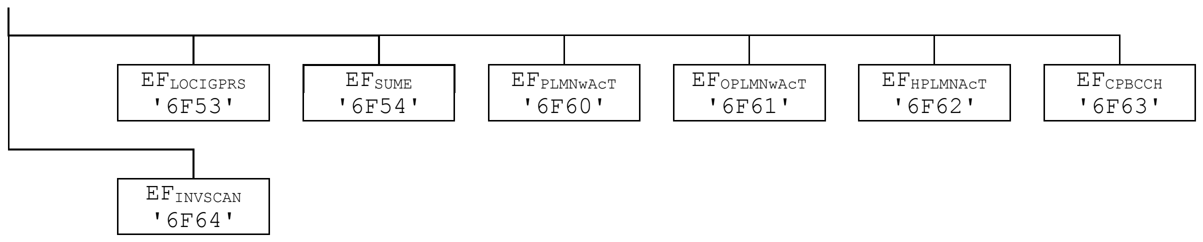
2.4 Signal

The communication with the modem is not always successful and minicom might not be getting any responses to the "AT" messages. After running the Signal app, the communication initializes. The initialization of the serial port by minicom may be at fault.

3. Subscriber & Universal Subscriber Identity Module (SIM/USIM)

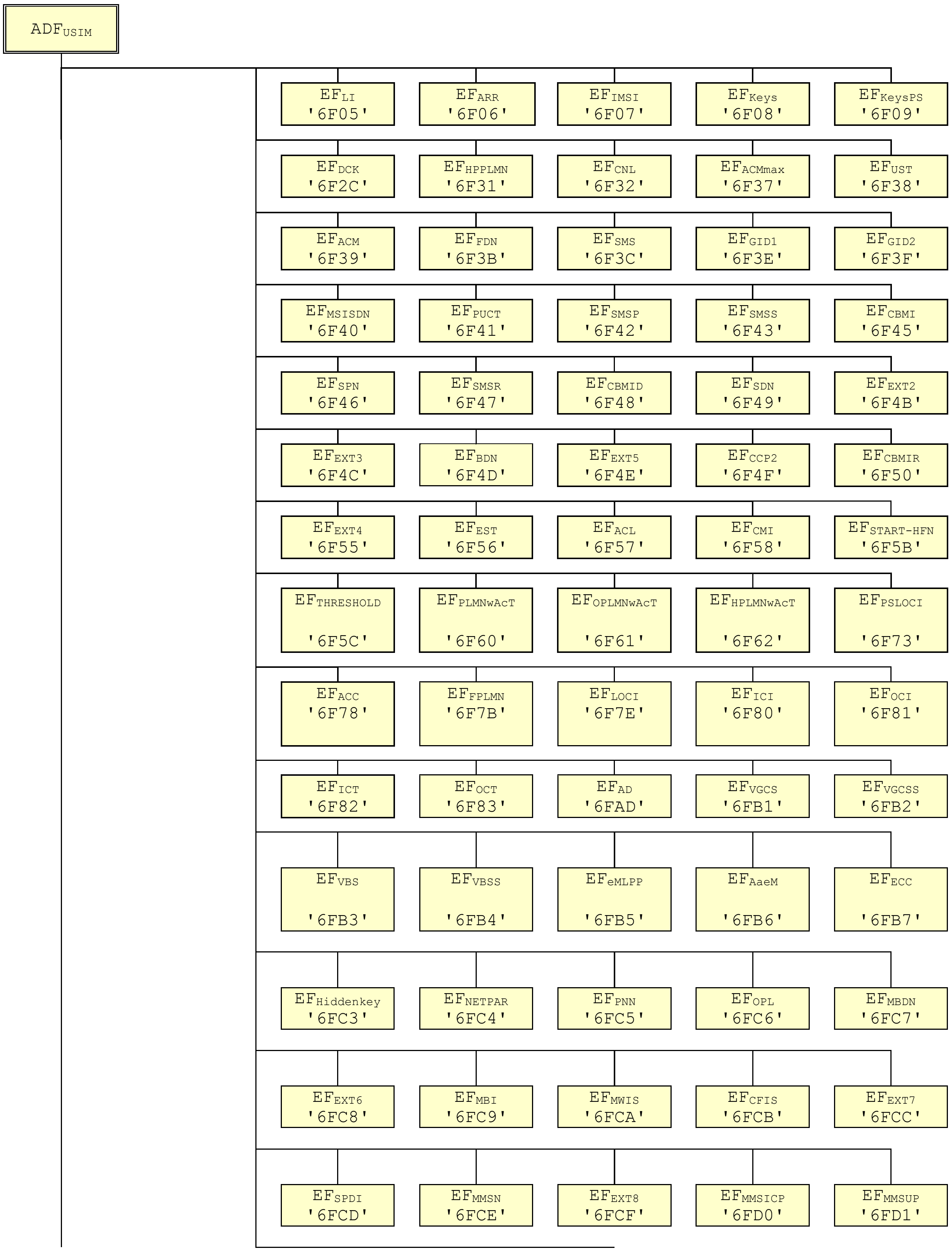
The Subscriber Identity Module (SIM) is an embedded system installed in a smartcard, also known as Integrated Circuit Card (ICC) [10]. It is used for the secure storage of the International Mobile Subscriber Identity (IMSI) [11] as well as encryption keys that are used to verify the modules identity and secure the mobile communication as far as confidentiality and integrity are concerned. Furthermore, a SIM card contains the Integrated Circuit Card Identifier (ICCD), the Authentication Key (Ki), Location Area Identity (LAI) [12], contacts and SMS messages. The above mentioned values are stored in special files in the SIM module called Elementary Files (EF). A figure is presented below, visualizing the tree structure of folders and EF files in the SIM module.


MF '3F00'
DF$_{GSM}$ '7F20'
DF$_{TELECOM}$ '7F10'
DF$_{IS-41}$ '7F22'
DF$_{FP-CTS}$ '7F23'
see GSM 11.19
EF$_{ICCID}$ '2FE2'
EF$_{ELP}$ '2F05'
EF$_{ADN}$ '6F3A'
EF$_{FDN}$ '6F3B'
EF$_{SMS}$ '6F3C'
EF$_{CCP}$ '6F3D'
EF$_{MSISDN}$ '6F40'
EF$_{SMSP}$ '6F42'
EF$_{SMSS}$ '6F43'
EF$_{LND}$ '6F44'
EF$_{SMSR}$ '6F47'
EF$_{SDN}$ '6F49'
EF$_{EXT1}$ '6F4A'
EF$_{EXT2}$ '6F4B'
EF$_{EXT3}$ '6F4C'
EF$_{BDN}$ '6F4D'
EF$_{EXT4}$ '6F4E'
DF$_{GRAPHICS}$ '5F50'
EF$_{IMG}$ '4F20'
DF$_{IRIDIUM}$ '5F30'
DF$_{GLOBST}$ '5F31'
DF$_{ICO}$ '5F32'
DF$_{ACeS}$ '5F33'
DF$_{EIA/TIA-553}$ '5F40'
DF$_{CTS}$ '5F60'
see GSM 11.19
DF$_{SoLSA}$ '5F70'
EF$_{SAI}$ '4F30'
EF$_{SLL}$ '4F31'
DF$_{MExE}$ '5F3C'
EF$_{MExE-ST}$ '4F40'
EF$_{ORPK}$ '4F41'
EF$_{ARPK}$ '4F42'
EF$_{TPRPK}$ '4F43'
EF$_{LP}$ '6F05'
EF$_{IMSI}$ '6F07'
EF$_{Kc}$ '6F20'
EF$_{PLMNsel}$ '6F30'
EF$_{HPPLMN}$ '6F31'
EF$_{ACMmax}$ '6F37'
EF$_{SST}$ '6F38'
EF$_{ACM}$ '6F39'
EF$_{GID1}$ '6F3E'
EF$_{GID2}$ '6F3F'
EF$_{PUCT}$ '6F41'
EF$_{CBMI}$ '6F45'
EF$_{SPN}$ '6F46'
EF$_{CBMID}$ '6F48'
EF$_{BCCH}$ '6F74'
EF$_{ACC}$ '6F78'
EF$_{FPLMN}$ '6F7B'
EF$_{LOCI}$ '6F7E'
EF$_{AD}$ '6FAD'
EF$_{PHASE}$ '6FAE'
EF$_{VGCS}$ '6FB1'
EF$_{VGCSS}$ '6FB2'
EF$_{VBS}$ '6FB3'
EF$_{VBSS}$ '6FB4'
EF$_{eMLPP}$ '6FB5'
EF$_{AAeM}$ '6FB6'
EF$_{ECC}$ '6FB7'
EF$_{CBMIR}$ '6F50'
EF$_{NIA}$ '6F51'
EF$_{KcGPRS}$ '6F52'

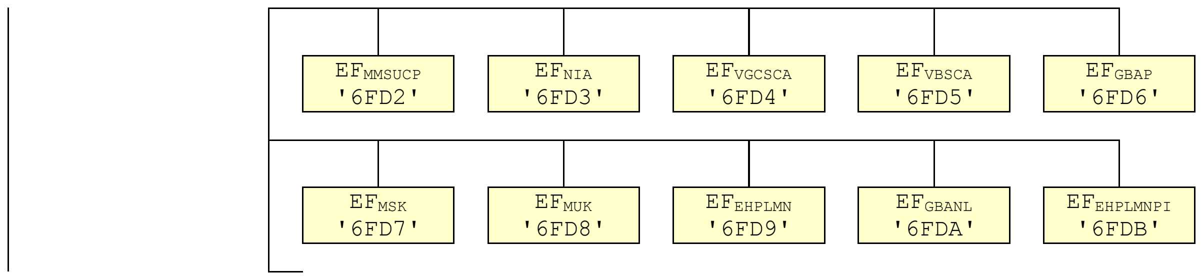


SIM File System Structure [13]

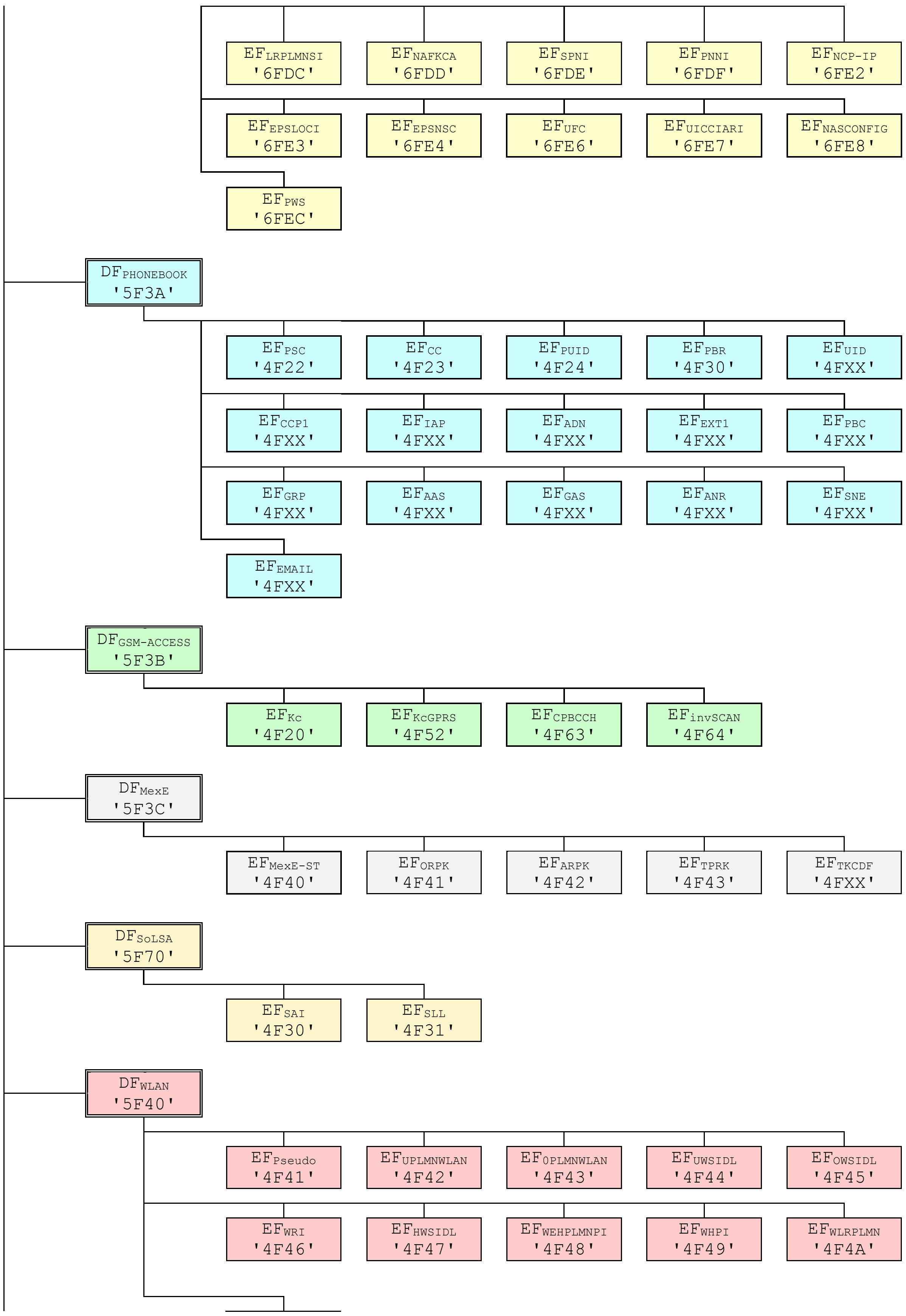
The Universal Subscriber Identity Module (USIM) is similar to the SIM with a number of differences. The main differential factor is that the USIM is used for the UMTS [14] network and the SIM for the GSM [15] network. A figure of the USIM folders and EF files structure is presented below.

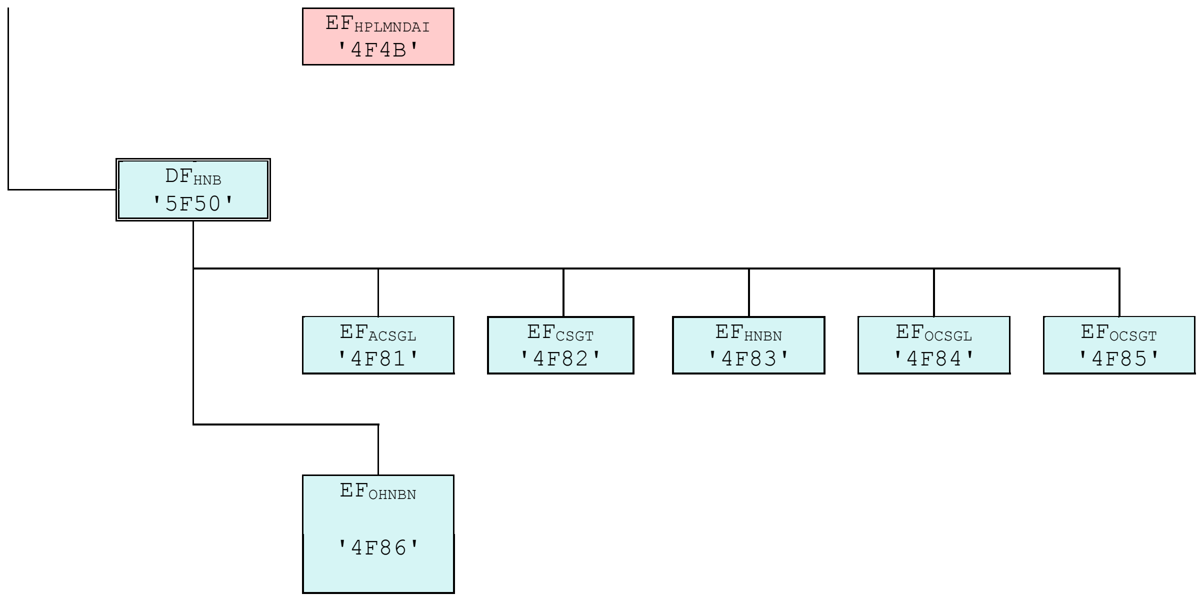


ADF$_{USIM}$
EF$_{LI}$ '6F05'
EF$_{ARR}$ '6F06'
EF$_{IMSI}$ '6F07'
EF$_{Keys}$ '6F08'
EF$_{KeysPS}$ '6F09'
EF$_{DCK}$ '6F2C'
EF$_{HPPLMN}$ '6F31'
EF$_{CNL}$ '6F32'
EF$_{ACMmax}$ '6F37'
EF$_{UST}$ '6F38'
EF$_{ACM}$ '6F39'
EF$_{FDN}$ '6F3B'
EF$_{SMS}$ '6F3C'
EF$_{GID1}$ '6F3E'
EF$_{GID2}$ '6F3F'
EF$_{MSISDN}$ '6F40'
EF$_{PUCT}$ '6F41'
EF$_{SMSP}$ '6F42'
EF$_{SMSS}$ '6F43'
EF$_{CBMI}$ '6F45'
EF$_{SPN}$ '6F46'
EF$_{SMSR}$ '6F47'
EF$_{CBMID}$ '6F48'
EF$_{SDN}$ '6F49'
EF$_{EXT2}$ '6F4B'
EF$_{EXT3}$ '6F4C'
EF$_{BDN}$ '6F4D'
EF$_{EXT5}$ '6F4E'
EF$_{CCP2}$ '6F4F'
EF$_{CBMIR}$ '6F50'
EF$_{EXT4}$ '6F55'
EF$_{EST}$ '6F56'
EF$_{ACL}$ '6F57'
EF$_{CMI}$ '6F58'
EF$_{START-HFN}$ '6F5B'
EF$_{THRESHOLD}$ '6F5C'
EF$_{PLMNwAcT}$ '6F60'
EF$_{OPLMNwAcT}$ '6F61'
EF$_{HPLMNwAcT}$ '6F62'
EF$_{PSLOCI}$ '6F73'
EF$_{ACC}$ '6F78'
EF$_{FPLMN}$ '6F7B'
EF$_{LOCI}$ '6F7E'
EF$_{ICI}$ '6F80'
EF$_{OCI}$ '6F81'
EF$_{ICT}$ '6F82'
EF$_{OCT}$ '6F83'
EF$_{AD}$ '6FAD'
EF$_{VGCS}$ '6FB1'
EF$_{VGCSS}$ '6FB2'
EF$_{VBS}$ '6FB3'
EF$_{VBSS}$ '6FB4'
EF$_{eMLPP}$ '6FB5'
EF$_{AaeM}$ '6FB6'
EF$_{ECC}$ '6FB7'
EF$_{Hiddenkey}$ '6FC3'
EF$_{NETPAR}$ '6FC4'
EF$_{PNN}$ '6FC5'
EF$_{OPL}$ '6FC6'
EF$_{MBDN}$ '6FC7'
EF$_{EXT6}$ '6FC8'
EF$_{MBI}$ '6FC9'
EF$_{MWIS}$ '6FCA'
EF$_{CFIS}$ '6FCB'
EF$_{EXT7}$ '6FCC'
EF$_{SPDI}$ '6FCD'
EF$_{MMSN}$ '6FCE'
EF$_{EXT8}$ '6FCF'
EF$_{MMSICP}$ '6FD0'
EF$_{MMSUP}$ '6FD1'



EF$_{MMSUCP}$ '6FD2'
EF$_{NIA}$ '6FD3'
EF$_{VGCSCA}$ '6FD4'
EF$_{VBSCA}$ '6FD5'
EF$_{GBAP}$ '6FD6'
EF$_{MSK}$ '6FD7'
EF$_{MUK}$ '6FD8'
EF$_{EHPLMN}$ '6FD9'
EF$_{GBANL}$ '6FDA'
EF$_{EHPLMNPI}$ '6FDB'


EF$_{LRPLMNSI}$ '6FDC'
EF$_{NAFKCA}$ '6FDD'
EF$_{SPNI}$ '6FDE'
EF$_{PNNI}$ '6FDF'
EF$_{NCP-IP}$ '6FE2'
EF$_{EPSLOCI}$ '6FE3'
EF$_{EPSNSC}$ '6FE4'
EF$_{UFC}$ '6FE6'
EF$_{UICCIARI}$ '6FE7'
EF$_{NASCONFIG}$ '6FE8'
EF$_{PWS}$ '6FEC'
DF$_{PHONEBOOK}$ '5F3A'
EF$_{PSC}$ '4F22'
EF$_{CC}$ '4F23'
EF$_{PUID}$ '4F24'
EF$_{PBR}$ '4F30'
EF$_{UID}$ '4FXX'
EF$_{CCP1}$ '4FXX'
EF$_{IAP}$ '4FXX'
EF$_{ADN}$ '4FXX'
EF$_{EXT1}$ '4FXX'
EF$_{PBC}$ '4FXX'
EF$_{GRP}$ '4FXX'
EF$_{AAS}$ '4FXX'
EF$_{GAS}$ '4FXX'
EF$_{ANR}$ '4FXX'
EF$_{SNE}$ '4FXX'
EF$_{EMAIL}$ '4FXX'
DF$_{GSM-ACCESS}$ '5F3B'
EF$_{Kc}$ '4F20'
EF$_{KcGPRS}$ '4F52'
EF$_{CPBCCH}$ '4F63'
EF$_{invSCAN}$ '4F64'
DF$_{MexE}$ '5F3C'
EF$_{MexE-ST}$ '4F40'
EF$_{ORPK}$ '4F41'
EF$_{ARPK}$ '4F42'
EF$_{TPRK}$ '4F43'
EF$_{TKCDF}$ '4FXX'
DF$_{SoLSA}$ '5F70'
EF$_{SAI}$ '4F30'
EF$_{SLL}$ '4F31'
DF$_{WLAN}$ '5F40'
EF$_{Pseudo}$ '4F41'
EF$_{UPLMNWLAN}$ '4F42'
EF$_{0PLMNWLAN}$ '4F43'
EF$_{UWSIDL}$ '4F44'
EF$_{OWSIDL}$ '4F45'
EF$_{WRI}$ '4F46'
EF$_{HWSIDL}$ '4F47'
EF$_{WEHPLMNPI}$ '4F48'
EF$_{WHPI}$ '4F49'
EF$_{WLRPLMN}$ '4F4A'



USIM File Structure [16]

3.1 Authentication process for SIM & USIM

The authentication process of the GSM network was altered in order to offer improved security features in the UMTS network. A more detailed view is presented below.

3.1.1 GSM Authentication

The Mobile Equipment (ME) requests to authenticate with the network. The network receives the request and replies with a random value (RAND) to the ME. The ME passes the RAND value to the SIM in the command RUN GSM ALGORITHM. The SIM returns a Signed Response (SRES) and a Ciphering Key (Kc) value. The network receives the SRES value and compares it with another SRES value calculated by itself. If the SRES is the same, the ME is successfully authenticated to the GSM network [17]. The RAND, SRES and Kc values consist the Triplet.

3.1.2 UMTS Authentication

The authentication process in the UMTS network is considered more secure, since the algorithms are stronger and it offers mutual authentication, both for the ME and the network [18]. The ME requests to authenticate with the network. The network receives the request and replies with a random value RAND and with an Authentication Token (AUTN). The AUTN contains a Message Authentication Code (MAC). The ME calculates a new MAC called XMAC and compares the two values. If they are equal the ME authenticates the network. Otherwise, the ME sends an authentication failure message to the network. Similarly to the GSM process, the ME sends the SRES value to the network, which compares it with the XRES value calculated by itself. If the values are equal then the ME is authenticated to the UMTS network.

4. AT Commands

The AT Commands ("AT" standing for "attention") is a set of Hayes commands, originally developed for the Hayes Smartmodem 300 baud modem. The group of commands is made of a set of short text strings which combine together to form complete operational commands, for example dialing, terminating connections, changing parameters and extracting information of many sorts. The vast majority of modems use the Hayes commands. However, due to the large number of firmware and baseband devices, AT commands are not supported completely in all devices. A specific firmware may support commands another does not and vice versa. Furthermore, a baseband may support proprietary AT commands, available only for a specific device.

4.1 AT Commands in GSM

The ETSI GSM 07.07 (3GPP TS 27.007) specifies AT style commands for controlling a GSM phone or modem.

Examples of GSM commands [9]

Command	Description
AT+CPIN=1234	Enter PIN code
AT+CPWD="SC","old","new"	Change PIN code from 'old' to 'new'
AT+CLCK="SC",0,"1234"	Remove PIN code
AT&V	Status
ATI	Status (Manufacturer, Model, Revision, IMEI, capabilities)
AT+COPS=?	List available networks
AT+CSQ	Get signal strength
ATD*99#	Dial access point

4.2 AT Commands for Data Extraction

In order to extract data from the SIM/USIM we need to use AT Commands. Minicom can enable the user to send AT Commands to the phone's modem and get a response. The SIM/USIM access can be accomplished with two commands:

- **AT+CSIM** (Generic SIM Access) [1]

  Command: +CSIM=<length>,<command>

  Response: +CSIM: <length>,<response>

  The <length> field is the number of bytes the command consists of.The <command> field is populated with APDU commands that are sent to the SIM/USIM module. The <response> field is the data sent by the SIM/USIM module.

- **AT+CRSM** (Restricted SIM Access) [1]

  Command: +CRSM=<command>[,<fileid>[,<P1>,<P2>,<P3>[,<data>[,<pathid>]]]]

  Response: +CRSM: <sw1>,<sw2>[,<response>]

  The <command> field is one of the following:

● 176	READ BINARY
● 178	READ RECORD
● 192	GET RESPONSE
● 214	UPDATE BINARY
● 220	UPDATE RECORD
● 242	STATUS
● 203	RETRIEVE DATA
● 219	SET DATA

  The <fileid> field is the integer of the file id. For example the integer for file '6F08' is '28424'. The <P3> field is the number of bytes the response should have. If the <sw1> and <sw2> fields have values equal to "144" and "0" respectively, this means the command execution was successful.

Examples of both commands:

CSIM:

```
AT+CSIM=14,"A0A40000023F00" (Go to 3F00 directory*)

AT+CSIM=14,"A0A40000027F20" (Go to 7F20 directory*)

AT+CSIM=14,"A0A40000026F20" (Go to 6F20 file)

AT+CSIM=10,"A0B0000009" (Get previous file response)
```

** Not mandatory*

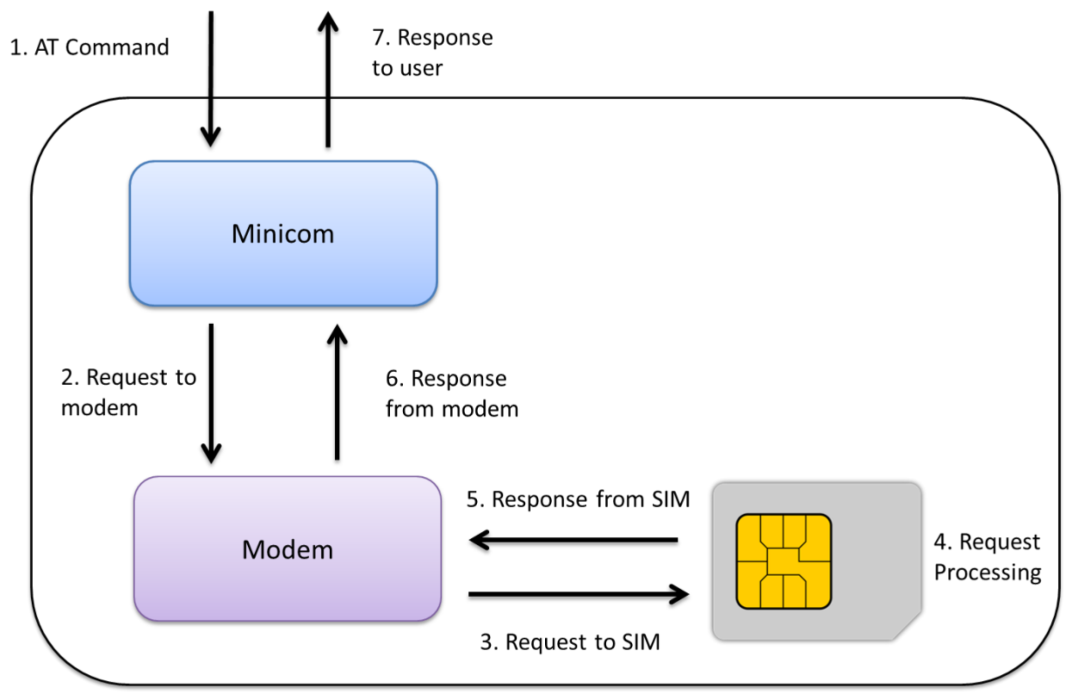
CRSM:

```
      AT+CRSM=176,28448,0,0,9
```

Both commands can be used to access the SIM/USIM module data. It was observed, however, that some USIM modules do not support the CSIM command so the CRSM is the only possible way in data access.

4. Experiment Process

With the environment described above installed and set, the data extraction can be started. The following figure explains the process, the modules that are participating and the information flow.



3. Process Flow

The process that takes place is:

1. The mobile phone takes as input an AT Command. The input source can be the user or an automated script and the AT Command is the AT+CSRM. The AT Command also includes the file that will be read from the SIM/USIM card.
2. The AT Command is received by minicom and is processed. Then minicom performs a request on the mobile device's modem.
3. The modem receives the request and performs a new one to the SIM/USIM card.
4. The command is received and processed by the SIM/USIM card in order to respond with the requested data. In this case, the data is the content of an EF file.
5. A response is generated by the SIM/USIM card and forwarded to the modem. The response in this case is the value of the EF file.
6. The response is received by the modem and forwarded to minicom.
7. The final step of the data flow is the receipt of the request from minicom and the presentation of the result. The presentation may be the result being printed on the phone screen for the user to see or the saving of the value in a database of some sort.

This is the description of the data flow and the nodes from which the information is passed through. Further processing of the data may take place, such as timestamp addition, value parsing, location added and altogether database storage.

5. Data collection and value

The SIM/USIM module contains numerous files for different purposes and not all of them are related to the current experiment or to security in general. Below follows the description of the important information we can access.

IMSI : The IMSI [11] is a 8 bytes value and is unique for a SIM/USIM card. It is used by the network for acquiring data for the specific subscriber. To prevent unwanted tracking and eavesdropping, the IMSI value is sent as rarely as possible. Instead the value used is the Temporary Mobile Subscriber Identity (TMSI) [19]. It is located in the EF(IMSI) with identifier 6F07 [17].

TMSI : The TMSI is a 4 bytes value and it is the identifier that is most commonly sent between the ME and the network. It is set by the network and only for the ME that are connected to a specific station. If the ME changes location and connects to a new station, the TMSI will be changed too. It is located in the EF(LOCI) within the first 4 bytes and with identifier 6F7E [17].

Kc : The Kc is the ciphering key used by the SIM module. It is used for encrypting the voice communication between the ME and the network. Its length is 8 bytes and it is located in the EF(Kc) with identifier 6F08 [17]. This value is also present in the USIM module in the EF(Kc) file with identifier 4F20 [16].

KcGPRS : It is similar to the Kc with the difference that it is used for the encryption of the GPRS data between the ME and the network. Its length is 8 bytes and it is located in the EF(KcGPRS) with identifier 6F52 [17].This value is also present in the USIM module in the EF(KcGPRS) file with identifier 4F52 [16].

CK : CK is the ciphering key used by the USIM module. It is used for encrypting communication between the ME and the network. Its length is 16 bytes and it is located in the EF(Keys) with identifier 6F08, within the 2nd and the 17th byte [16].

IK : IK is the integrity key used by the USIM module. It is used for ensuring integrity for packets exhanged between the ME and the network. Its length is 16 bytes and it is located in the EF(Keys) with identifier 6F08, within the 18th and the 33rd byte [16].

Ciphering Indicator : This is a value of 1 bit. When the ME and the network communicate through an encrypted channel the value is set to 1. When the communication is unencrypted the value is set to 0. It is located in the EF(AD) with identifier 6FAD and within the 3rd byte [17].

P-TMSI : This value is the Packet TMSI used for data packets. It is located in the EF(PSLOCI) with identifier 6F73 [17].

RAI : This value represents the Routing Area Information.It is located in the EF(PSLOCI) along with the P-TMSI value [17].

RAUS: The Routing Area Update Status value is, as well, located in the EF(PSLOCI).

TMSI TIME : This value represents the time interval in which the next TMSI update will take place. It is located in the EF(LOCI) with identifier 6F7E [17].

THRESHOLD : This value represents the time interval in which a keys' update will take place. It is located in the EF(THRESHOLD) with identifier 6F5C [17].

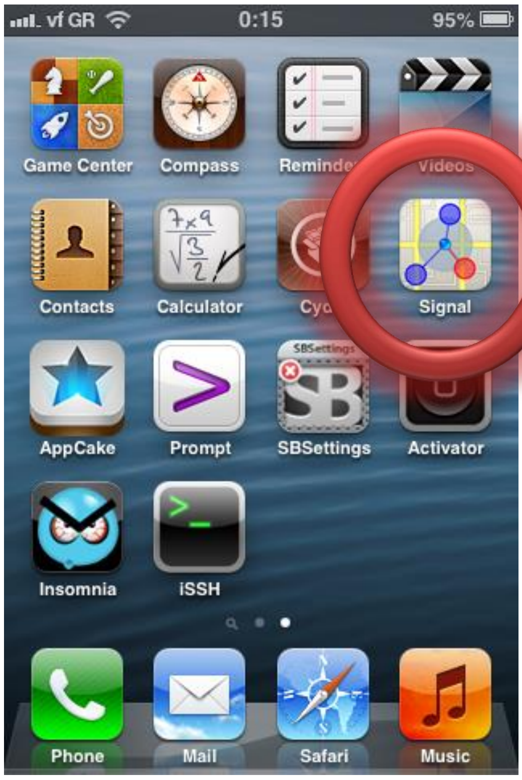
6. Information gathering

The next step is the automation of the extraction of data from the SIM/USIM module. The framework used is a minicom wrapper written in ruby. After the extraction, data is saved is a comma separated value (csv) type database along with a timestamp, the provider name, the Location Area Code (LAC) and the cell tower id. A description of the database structure follows:

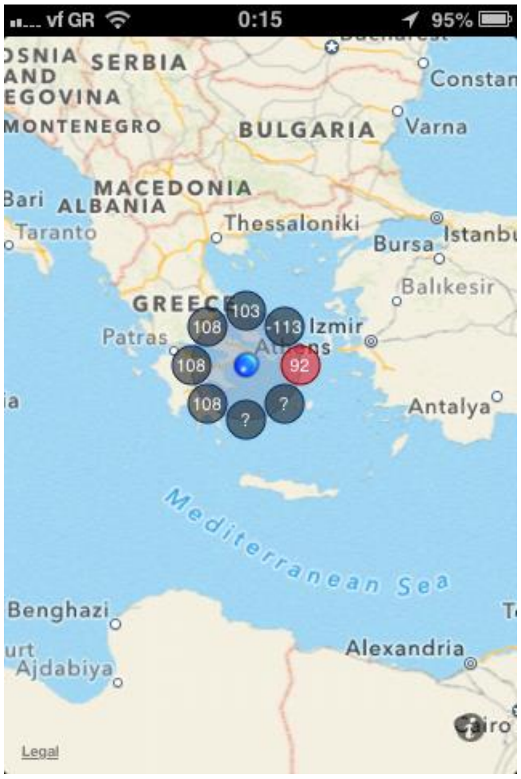
- **Type:** This value described the type of value that populates a specific row. The different types are:
    - Ciphering Mode
    - IMSI
    - Kc
    - KcGPRS
    - CK
    - IK
    - TMSI
    - PTMSI
    - RAI
    - RAUS
    - TMSI TIME
    - THRESHOLD
- **Value:** The value of the type. For the Ciphering Mode, the options are “ON”, “OFF” or the actual value if it does not match any filter rules..
- **Provider:** This is the name of the service provider. It is set to return a name for the greek providers. Otherwise, a provider id is returned.
- **LAC:** This is the Location Area Code and it is helpful for estimating the area of the current value.
- **CellID:** This is the cell tower id and it is helpful for the security assessment of each tower individually (eg a specific tower offers only 2G services).
- **Time:** A timestamp of the current value. The structure is Year - Month - Day, Hours - Minutes - Seconds, GMT. Example: 2013-01-16 03:40:10 +0200.

7. Tutorial

The first step is to run the Signal app (Screenshot 1). The reason we run this is that the serial port is initially locked and Signal enables us to initialize it and achieve serial communication.

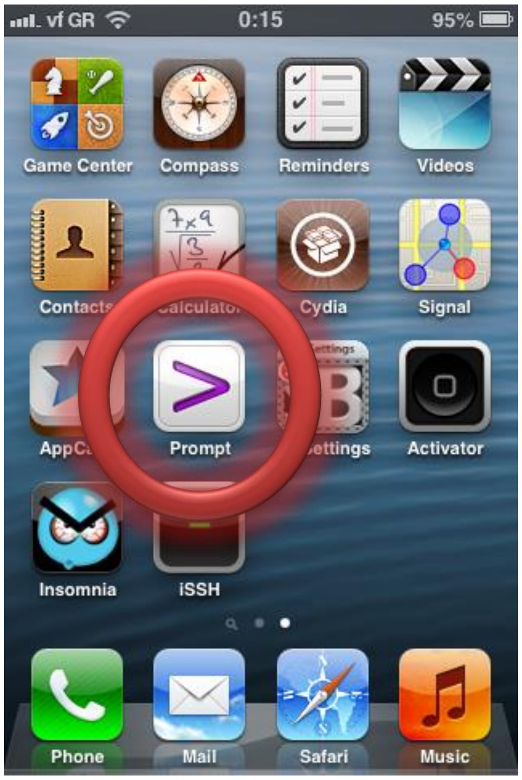


Screenshot 1 Signal App Icon



Screenshot 2 Signal Initial Screen

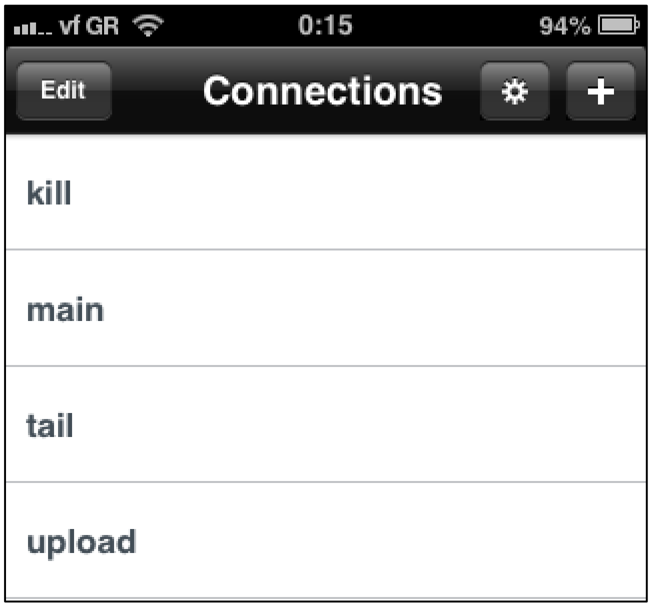
After the initialization of Signal, we minimize the app using the home button and then start an SSH client. We chose to use Prompt app for this, but any compatible SSH client should work properly.



Screenshot 3 Prompt App Icon

Once we start Prompt, we have a preconfigured set of profiles to run. The set consinsts of 4 profiles which are described below:

- **kill:** This profile is configured to log in via ssh protocol and execute the "killminicom.rb" script. The purpose of this script is to kill minicom when an AT command has finished loading, thus enabling the next AT command to be executed.
- **main:** This profile is configured to run the "main.rb" script which is responsible for initiating the AT Commands as well as logging the data.
- **tail:** This profile is not required for the logging process. It is helpful, however, as it executes the "tail –f db.csv" command which monitors the database of the logs and presents real time the new data that is logged.
- **upload:** This profile is configured to run the "upload.rb" script. This function is responsible for uploading the logged data to a remote server. The purpose of this is the collection of all data from all devices in a central location for better management and storage.



kill

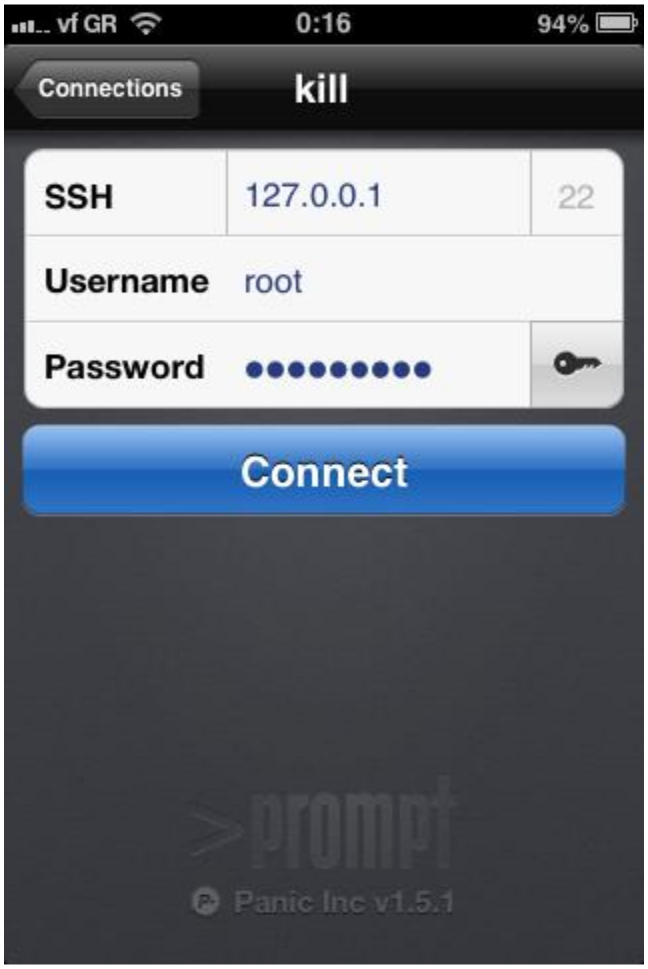
main

tail

upload

Screenshot 4 Prompt Profiles

The next step is running the kill profile. Once taped, a connection window appears (Screenshot 5). We tap connect and the next screen shows us the "killminicon.rb" script running (Screenshot 6). In order to do this manually go to the script directory and type "ruby killminicom.rb".

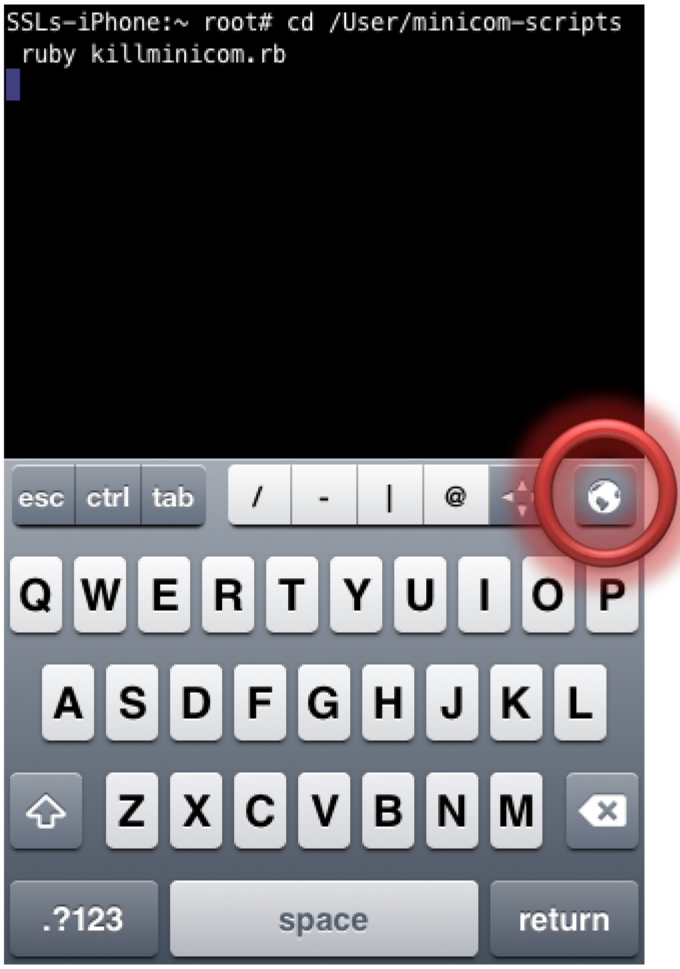


SSH 127.0.0.1 22

Username root

Password ●●●●●●●●●

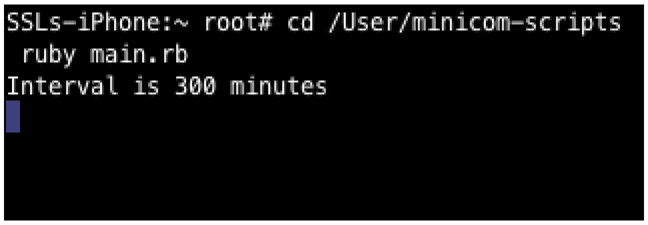
Screenshot 5 Prompt Connection Window



```
SSLs-iPhone:~ root# cd /User/minicom-scripts
 ruby killminicom.rb
```

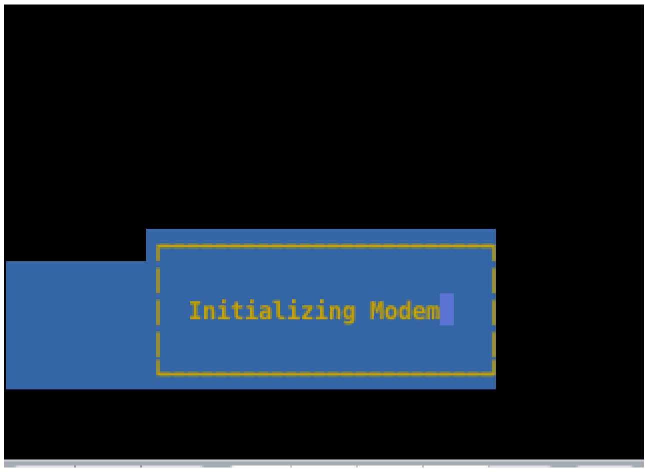
Screenshot 6 kill profile running

Once “killminicom.rb” is running, we start the main profile. We tap on the globe icon (Screenshot 6 – highlighted in red circle) and go back to the profile list. We choose the “main” profile and after tapping “Connect” we see a message saying “Interval is X minutes” (Screenshot 7). This value represents the frequency a data collection will be made, for example every 5 minutes. After that, you will see the initialization screen of minicom with the message “Initializing Modem” (Screenshot 8). To do this manually go to the script directory and type “ruby main.rb”.



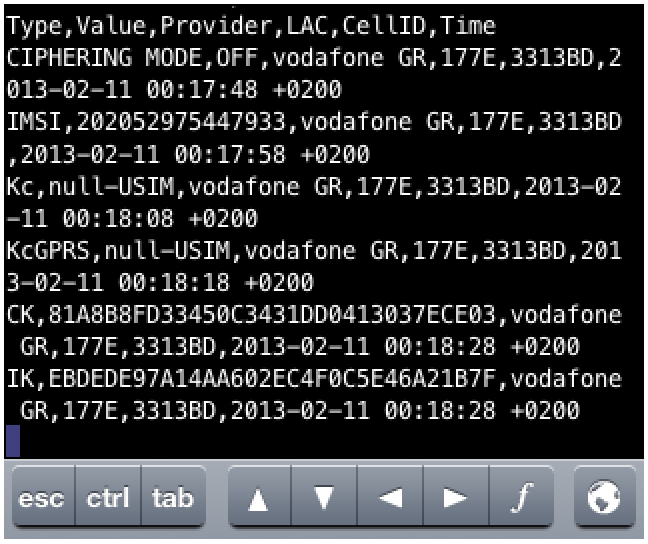
```
SSLs-iPhone:~ root# cd /User/minicom-scripts
 ruby main.rb
Interval is 300 minutes
```

Screenshot 7 Interval Message



Screenshot 8 Minicom initialization screen

Now the process has started and data are being logged in the database. If you want to monitor the process, press the globe icon to go back to the profile list and tap on the "tail" profile. Once tapped and connected, the monitor command will be executed and you will be presented with the last lines of the database (Screenshot 9). Every new addition will be shown to you in real time. To do this manually, go to the database location and type "tail –f db.csv".

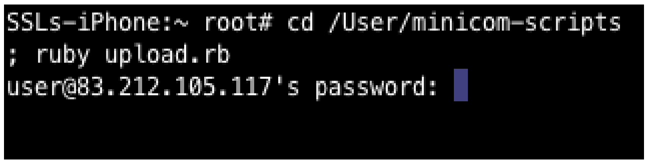


```
Type,Value,Provider,LAC,CellID,Time
CIPHERING MODE,OFF,vodafone GR,177E,3313BD,2
013-02-11 00:17:48 +0200
IMSI,202052975447933,vodafone GR,177E,3313BD
,2013-02-11 00:17:58 +0200
Kc,null-USIM,vodafone GR,177E,3313BD,2013-02
-11 00:18:08 +0200
KcGPRS,null-USIM,vodafone GR,177E,3313BD,201
3-02-11 00:18:18 +0200
CK,81A8B8FD33450C3431DD0413037ECE03,vodafone
 GR,177E,3313BD,2013-02-11 00:18:28 +0200
IK,EBDEDE97A14AA602EC4F0C5E46A21B7F,vodafone
 GR,177E,3313BD,2013-02-11 00:18:28 +0200
```

Screenshot 9 tail command output

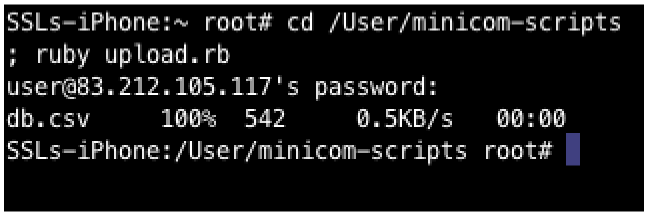
ATTENTION: Once the main and/or tail profile is running you must <u>bring the Signal app back in the foreground</u>. However, if the Prompt app is kept in the background for ten minutes then the log process will stop. This is due to a restriction in the iOS architecture. In order to run the process constantly, you should bring the Prompt app in the foreground every 5-8 minutes for a few seconds and then bring Signal in the foreground again.

For uploading the logged data, tap the globe icon and go back to the profile list. Then tap on the "upload" profile and the upload command will be executed. You will be prompted to enter a password in order to upload data to the server and once the upload is complete you will be informed by the screen. In order to do this manually go to the script folder and type "ruby upload.rb".



```
SSLs-iPhone:~ root# cd /User/minicom-scripts
; ruby upload.rb
user@83.212.105.117's password: 
```

Screenshot 10 Prompt for upload server password



```
SSLs-iPhone:~ root# cd /User/minicom-scripts
; ruby upload.rb
user@83.212.105.117's password:
db.csv      100%  542      0.5KB/s   00:00
SSLs-iPhone:/User/minicom-scripts root# 
```

Screenshot 11 Upload completion

8. Assistive Scripts

Due to the large sum of data acquired by the experiment process, the creation of a number of assistive scripts was necessary. We currently use two scripts in order to parse data from the logs.

8.1 keylife.rb

When using this script, the user gives as input the type of key they want to examine, e.g. CK. The script parses all the logs and locates the entries where the type of key is that of CK and saves the value of the key and the time they key was logged. After that, it checks whether the same key exists with a different timestamp. It saves the first time and the last time value the key has appeared, therefore creating a time period when the key was active. An example result of the script output is the following:

```
Key-- 58EAFA7AEA6BF906ACA8A5C2D469CD72 Duration-- 1182.0 (19 minutes 42 seconds)

Key-- DE7C0C806806788AD66E32E83302F91C Duration-- 797.0 (13 minutes 17 seconds)

Key-- 3C2EA40B6333496D21B367A025472B64 Duration-- 1186.0 (19 minutes 46 seconds)

Key-- 80F56540E9C520E48C546E50A1C973D4 Duration-- 0.0 ()
```

When the duration is 0, this means the key has appeared only once in our logs.

8.2 unique.rb

This script is similar to the previous. It also takes as input a key type from the user, e.g. CK. The main difference is that it will search the logs for multiple appearances of the key. If a result is found, it is printed among with the different timestamps of the key. Due to the nature of our logs, it is possible to have false-positive results, therefore a verification by the user is required as to define whether or not the time difference is large enough to conclude that a key was user again after a period of time. A result example follows:

```
80F56540E9C520E48C546E50A1C973D4 -- 3

2012-02-28 13:27:56 +0200

2013-02-28 14:53:22 +0200

2013-02-28 15:13:09 +0200
```

9. Related Work

Related works and projects focus mostly on ways and or improved suggestions on cracking the encryption algorithms such as A5/1 and A5/2. After cracking the encrypted communication, the confidentiality is breached and a security issue is thereby proved. Our work focuses on a different perspective by setting the same goal but reaching via another route. Our method collects essential security data from within the device and makes the breaching of the communication a much easier task. This is due to the fact that we already possess the encryption keys and no cracking process is needed.

References

[1] 3GPP TS 27.007, Technical Specification Group Core Network and Terminals - AT command set for User Equipment (UE)

[2] iOS Jailbreaking, Wikipedia https://en.wikipedia.org/wiki/IOS_jailbreaking

[3] OpenSSH, Wikipedia https://en.wikipedia.org/wiki/OpenSSH

[4] Cydia, Wikipedia https://en.wikipedia.org/wiki/Cydia

[5] Minicom, Wikipedia https://en.wikipedia.org/wiki/Minicom

[6] Ruby Language Official Site http://www.ruby-lang.org/en/

[7] Signal App for iPhone Shows Detailed Info of Cellular Towers Around You

http://www.redmondpie.com/signal-app-for-iphone-shows-detailed-info-of-cellular-towers-around-you/

[8] http://www.letsunlockiphone.com/install-minicom-iphone-4-baseband/

[9] Hayes command set, Wikipedia https://en.wikipedia.org/wiki/Hayes_command_set

[10] Smartcard, Wikipedia https://en.wikipedia.org/wiki/Smart_card

[11] https://en.wikipedia.org/wiki/International_Mobile_Subscriber_Identity

[12] https://en.wikipedia.org/wiki/Location_Area_Identity

[13] ETSI TS 100 977, Specification of the Subscriber Identity Module -

Mobile Equipment (SIM-ME) Interface

[14] https://en.wikipedia.org/wiki/Universal_Mobile_Telecommunications_System

[15] https://en.wikipedia.org/wiki/GSM

[16] ETSI TS 131 102, Characteristics of the Universal Subscriber Identity Module (USIM) application

[17] 3GPP TS 11.11, Mobile Equipment (SIM - ME) interface

[18] 3GPP TS 33.102, 3G Security, Security Architecture

[19] https://en.wikipedia.org/wiki/Temporary_Mobile_Subscriber_Identity#TMSI

[20] adv-cmds Cydia packet page http://cydia.saurik.com/package/adv-cmds

Code Appendix

Filename : getciphering.rb

```ruby
require 'csv'

require './log.rb'


system('minicom -c on -S script_getciphering -C ciphering.txt')


if File.exist?('ciphering.txt')

  keys = File.open('ciphering.txt').readlines

  key_line = keys[1]

  keys_tuple = key_line[14..-3]

  if keys_tuple.length == 6

    ciphering = keys_tuple[4..5].to_i(16).to_s(2)

    if ciphering[-1] == '1'

      ciphering = 'ON'

    elsif ciphering[-1] == '0'

      ciphering = 'OFF'

    else

      ciphering = "ERROR"

    end

    log("CIPHERING MODE", ciphering)

    File.delete('ciphering.txt')

  else

    print "Length not matching"

  end

else

  print "Keys file not found."

end
```

Filename : getimsi.rb

```
require 'csv'

require "./log.rb"


system('minicom -c on -S script_getimsi -C imsi.txt')


if File.exist?('imsi.txt')

  keys = File.open('imsi.txt').readlines

  key_line = keys[1]

  keys_tuple = key_line[0..-2]

  if keys_tuple.length == 15

    imsi = keys_tuple

    log("IMSI", imsi)

    File.delete('imsi.txt')

  else

    print "Length not matching"

  end

else

  print "Keys file not found."

end
```

Filename : getkc_gprs.rb

```
require 'csv'

require './log.rb'


system('minicom -c on -S script_getkc_gprs -C kc_gprs.txt')


if File.exist?('kc.txt')

  keys = File.open('kc.txt').readlines

  key_line = keys[1]
  if key_line.include? "+CRSM: 106,130"
    system('minicom -c on -S script_getkc_gprs_usim -C kc_gprs_usim.txt')
              if File.exist?('kc_gprs_usim.txt')
              keys = File.open('kc_gprs_usim.txt').readlines
              key_line = keys[1]
                     if key_line.include? "+CRSM: 106,130"
                            log("Kc_GPRS_USIM","Unsupported USIM")
                            File.delete("kc_gprs_usim.txt")
                     else
                            keys_tuple = key_line[14..-3]
                       if keys_tuple.length == 18
                                   kc_gprs_usim = keys_tuple[0..15]
                                   log("Kc_GPRS_USIM", kc_gprs_usim)
                                   File.delete('kc_gprs_usim.txt')
                            end
                     end
              end
              File.delete('kc_gprs_usim.txt')
  end
  keys_tuple = key_line[14..-3]
  if keys_tuple.length == 18
    kc = keys_tuple[0..15]
    log("Kc", kc)
    File.delete('kc.txt')
  else
    print "Length not matching"
  end
else
  print "Keys file not found."
end
```

Filename : getkc.rb

```
require 'csv'

require './log.rb'


system('minicom -c on -S script_getkc -C kc.txt')


if File.exist?('kc.txt')

  keys = File.open('kc.txt').readlines

  key_line = keys[1]

  if key_line.include? "+CRSM: 106,130"

    system('minicom -c on -S script_getkc_usim -C kc_usim.txt')

             if File.exist?('kc_usim.txt')

             keys = File.open('kc_usim.txt').readlines

             key_line = keys[1]

                    if key_line.include? "+CRSM: 106,130"

                           log("Kc_USIM","Unsupported USIM")

                           File.delete("kc_usim.txt")

                    else

                           keys_tuple = key_line[14..-3]

                      if keys_tuple.length == 18

                                  kc_usim = keys_tuple[0..15]

                                  log("Kc", kc_usim)

                                  File.delete('kc_usim.txt')

                           end

                    end

             end

             File.delete('kc_usim.txt')

  end

  keys_tuple = key_line[14..-3]

  if keys_tuple.length == 18
```

```
    kc = keys_tuple[0..15]

    log("Kc", kc)

    File.delete('kc.txt')

  else

    print "Length not matching"

  end

else

  print "Keys file not found."

end
```

Filename : getkeyspacket.rb

```
require 'csv'

require './log.rb'


system('minicom -c on -S script_getkeyspacket -C keyspacket.txt')


if File.exist?('keyspacket.txt')

  keys = File.open('keyspacket.txt').readlines

  key_line = keys[1]

  keys_tuple = key_line[14..-3]

  if key_line.include? "+CRSM: 148,4"

    log("CK-Packet", "null-SIM")

    log("IK-Packet", "null-SIM")

    File.delete('keyspacket.txt')

  end

  if keys_tuple != nil && keys_tuple.length == 66

    #ksi = keys_tuple[0..1]

    ck = keys_tuple[2..33]

    ik = keys_tuple[34..65]

    log("CK-Packet", ck)
    log("IK-Packet", ik)
    File.delete('keyspacket.txt')
  else
    print "Length not matching"
  end
else
  print "Keys file not found."
end
```

Filename : getkeys.rb

```
require 'csv'
require './log.rb'

system('minicom -c on -S script_getkeys -C keys.txt')

if File.exist?('keys.txt')
  keys = File.open('keys.txt').readlines
  key_line = keys[1]
  keys_tuple = key_line[14..-3]
  if key_line.include? "+CRSM: 148,4"
    log("CK", "null-SIM")
    log("IK", "null-SIM")
    File.delete('keys.txt')
  end
  if keys_tuple != nil && keys_tuple.length == 66
    #ksi = keys_tuple[0..1]
    ck = keys_tuple[2..33]
    ik = keys_tuple[34..65]
    log("CK", ck)
    log("IK", ik)
    File.delete('keys.txt')
  else
    print "Length not matching"
  end
else
  print "Keys file not found."
end
```

Filename : getlaccellid.rb

```
system('minicom -c on -S script_creg -C creg.txt')

if File.exist?('creg.txt')
  creg = File.open('creg.txt').readlines
  info = creg[1].scan(/"([^"]*)"/)
  File.open('lac_cellid.txt', 'w') do |lid|
    lid << "#{info[0]}\n#{info[1]}"
  end
  File.delete('creg.txt')
else
  print "creg file not found."
end
```

Filename : getprovider.rb

```
system('minicom -c on -S script_cops -C cops.txt')

if File.exist?('cops.txt')
  cops = File.open('cops.txt').readlines
  provider_code = cops[1].scan(/"([^"]*)"/).to_s
  if provider_code.include?('20201')
    provider_name = "GR COSMOTE"
  elsif provider_code.include?('20205')
    provider_name = "vodafone GR"
  elsif provider_code.include?('20209')
    provider_name = "GR Q-TELECOM"
  elsif provider_code.include?('20210')
    provider_name = "TIM GR"
  else
    provider_name = provider_code
  end
  File.open('provider.txt', 'w') do |prv|
    prv << provider_name
  end
  File.delete('cops.txt')
else
  print "COPS file not found."
end
```

Filename : getptmsi.rb

```
require 'csv'
require './log.rb'

system('minicom -c on -S script_getptmsi -C ptmsi.txt')

if File.exist?('ptmsi.txt')
  keys = File.open('ptmsi.txt').readlines
  key_line = keys[1]
  if key_line != nil && key_line.length > 1
    ptmsi = key_line[14..21]
    rai = key_line[28..39]
    raus = key_line[40..41]
    if ptmsi != nil
      if ptmsi.length == 8
        log("PTMSI", ptmsi)
        log("RAI", rai)
        log("RAUS", raus)
        File.delete('ptmsi.txt')
      elsif key_line == "+CRSM: 106,130\n"
        system('minicom -c on -S script_getptmsi_usim -C ptmsi_usim.txt')
        if File.exist?('ptmsi_usim.txt')
          keys = File.open('ptmsi_usim.txt').readlines
          key_line = keys[1]
          if key_line != nil && key_line.length > 1
            ptmsi = key_line[14..21]
            rai = key_line[28..39]
            raus = key_line[40..41]
            log("PTMSI", ptmsi)
            log("RAI", rai)
            log("RAUS", raus)
            File.delete('ptmsi_usim.txt')
          end
        end
      else
        print "Length not matching"
        log("PTMSI", "invalid")
        File.delete('ptmsi.txt')
      end
    end
  end
else
  print "Keys file not found."
```

```
end
```

Filename : getthreshold.rb

```
require 'csv'

require "./log.rb"


system('minicom -c on -S script_getthreshold -C threshold.txt')


if File.exist?('threshold.txt')

  keys = File.open('threshold.txt').readlines

  key_line = keys[1]

  keys_tuple = key_line[14..19]

  if key_line == "+CRSM: 148,4\n"

    log("THRESHOLD", "null-sim")

    File.delete('threshold.txt')

  end

  if keys_tuple.length == 6

    threshold = keys_tuple

    log("THRESHOLD", threshold)

    File.delete('threshold.txt')

  else

    print "Length not matching"

  end

else

  print "Keys file not found."

end
```

Filename : gettmsi.rb

```
require 'csv'

require './log.rb'


system('minicom -c on -S script_gettmsi -C tmsi.txt')


if File.exist?('tmsi.txt')

  keys = File.open('tmsi.txt').readlines

  key_line = keys[1]

  keys_tuple = key_line[14..-3]

  if keys_tuple == nil

    log("TMSI","null")

    File.delete('tmsi.txt')

  end

  if keys_tuple!= nil && keys_tuple.length == 22

    tmsi = keys_tuple[0..7]

    tmsi_time = keys_tuple[18..19]

    log("TMSI", tmsi)

    log("TMSI TIME", tmsi_time.to_i(16))

    File.delete('tmsi.txt')

  else

    print "Length not matching"

  end

else

  print "Keys file not found."

end
```

Filename : killminicom.rb

```
while 1 == 1

  check = `ps`

  #print check
  if check.include? "minicom -c"

    sleep(5) # 5sec is on average enough time for minicom to initialize and send the
command

    system('killall -9 minicom')

  end

end
```

Filename : log.rb

```
def strip(string)

  string = string.tr '"', ''

  string = string.tr '[', ''

  string.tr ']', ''

end


def log(title, data)

  if File.exist?('provider.txt')

    provider = File.open('provider.txt').readlines

    provider = strip(provider.to_s)

  else

    puts "Provider file not found"

  end


  if File.exist?('lac_cellid.txt')

    info = File.open('lac_cellid.txt').readlines

    if info.length > 1

      lac = strip(info[0][0..-2])

      cellid = strip(info[1])

    else

      lac = cellid = "Unavailable"

    end
  else
    puts "LAC & Cell ID file not found!"
  end
  CSV.open('db.csv', 'a') do |db|
    if File.zero?(db)
      db << ["Type", "Value", "Provider", "LAC", "CellID", "Time"]
    end
    db << [title, data, provider, lac, cellid, Time.now]
  end
end
```

Filename : main.rb

```
if ARGV[0].nil?
  interval = 300
else
  interval = ARGV[0].to_i * 60
end
puts "Interval is #{interval/60} minutes"

while true do
  system("rm -rf *.txt")
  system("rm -rf /var/lock/*")
  system("ruby getprovider.rb")
  system("ruby getlaccellid.rb")
  system("ruby getciphering.rb")
  system("ruby getimsi.rb")
  system("ruby getptmsi.rb")
  system("ruby getkc.rb")
  system("ruby getkc_gprs.rb")
  system("ruby getkeys.rb")
  system("ruby gettmsi.rb")

  system("rm -rf *.txt")

  sleep interval

end
```

Filename : README

```
iSim Sniffer

============



by Andis Anastasis



v0.5



Requirements

-------------

1. Jailbroken iPhone

2. OpenSSH

3. Minicom

4. Signal

5. ruby



Instructions

------------

Connect to your iPhone via ssh from a computer and run "minicom -w -c on". You will
enter the minicom interface. Type "Ctrl+A+E" in order to activate echo mode and see
what you type in the minicom interface. Make sure you have minicom properly
configured(See http://www.letsunlockiphone.com/install-minicom-iphone-4-baseband/).Try
typing "AT" and pressing ENTER. If you get an "OK" message this means you're ready to
go. If you don't get the "OK" after the "AT" then try running the Signal app. You will
see many messages passing through the terminal. Try locking the screen while the app
is running and see if you get only the "OK" when you type "AT" and not the other info
from the Signal app. Try many times if you need to. The goal is to type "AT" and get
only the "OK" response. Once you achieve that, you are ready to start getting data.
```

```
Open two ssh connections from the mobile (Prompt is a nice program). Navigate to the
location of the scripts and in the first ssh window type "ruby killminicom.rb". On the
second window type "ruby main.rb". The process should start. For better indication
that the process in still running and data are still being gathered you can open a
third ssh window, navigate to the location of the scripts and type "tail -f db.csv".
This will show you the changes made on the db.csv file. Every time a new record is
added, you will see the change appended. This means the process is still running
smoothly ;)
```

Filename : script_copn

```
send "AT+COPN"
```

Filename : script_cops

```
send "AT+COPS?"
```

Filename : script_creg

```
send "AT+CREG?"
```

Filename : script_getciphering

```
send "AT+CRSM=176,28589,0,0,3"
```

Filename : script_getimsi

```
send "AT+CIMI"
```

Filename : script_getkc

```
send "AT+CRSM=176,28448,0,0,9"
```

Filename : script_getkc_gprs

```
send "AT+CRSM=176,28498,0,0,9"
```

Filename : script_getkc_gprs_usim

```
send "AT+CRSM=176,20306,0,0,9"
```

Filename : script_getkc_usim

```
send "AT+CRSM=176,20256,0,0,9"
```

Filename : script_getkeys

```
send "AT+CRSM=176,28424,0,0,33"
```

Filename : script_getkeyspacket

```
send "AT+CRSM=176,28425,0,0,33"
```

Filename : script_getptmsi

```
send "AT+CRSM=176,28499,0,0,14"
```

Filename : script_getptmsi_usim

```
send "AT+CRSM=176,28531,0,0,14"
```

Filename : script_getthreshold

```
send "AT+CRSM=176,28508,0,0,3"
```

Filename : script_gettmsi

```
send "AT+CRSM=176,28542,0,0,22"
```

Filename : upload.rb

```
if File.exist?('db.csv')
  time = Time.now.strftime("%Y-%m-%d-%H%M%S")
  system("scp db.csv user@83.212.105.117:/home/user/idata/db-#{time}.csv")
  system("mv db.csv db-#{time}.csv")
else
  puts "File db.csv not found."
end
```