NTNU

Norwegian University of
Science and Technology

Decoding GSM

Magnus Glendrange
Kristian Hove
Espen Hvideberg

Master of Science in Communication Technology
Submission date: June 2010
Supervisor: Stig Frode Mjølsnes, ITEM
Co-supervisor: Danilo Gligoroski, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The A5/1 cipher used in GSM has been under attack, analysis and critique for a long time because of deficiencies, well documented in many papers. However, few findings have evolved into practical attacks. People declaring A5/1 as broken have never released the tools necessary to perform decryption of GSM traffic, nor have they published proof that confirms their claims. The media in particular should also take some part of the blame, as they do not always get the facts right and create inaccurate or misleading impressions as to whether A5/1 is cracked or not.

In August 2009, Nohl and Krissler announced the A5/1 Security Project. Their aim is to generate rainbow tables as a precomputation stage to enable a ciphertext-only attack against the A5/1 encryption algorithm. Using the A5/1 Security Project as a basis, this thesis will attempt to create a set of rainbow tables and investigate the feasibility of capturing GSM traffic. The latter will utilize interception software provided by the AirProbe project.

In addition, this thesis will try to perform an active attack where a GSM network is created. For instance, a USRP together with OpenBTS and Asterisk can provide a generic GSM infrastructure, from the Base Transceiver Station and upwards.
The USRP is an inexpensive hardware device that combined with GNU Radio can facilitate the deployment of Software-Defined Radio. Adding the capabilities of OpenBTS, a complete GSM stack could be set up.

The results of this project might become part of the student lab facilities of the Wireless Security course at NTNU.

Assignment given: 15. January 2010
Supervisor: Stig Frode Mjølsnes, ITEM

Abstract

We have participated in the creation of almost two terabytes of tables aimed at cracking A5/1, the most common ciphering algorithm used in GSM. Given 114-bit of known plaintext, we are able to recover the session key with a hit rate of 19%. The tables are expected to be unique as they provide the best coverage yet known to the authors, and they are the first step in a real-world passive attack against GSM.

An initial investigation and analysis into the air interface of GSM were performed, from both a theoretical and practical point of view. These examinations would be essential in order to utilize the generated tables in a practical attack.

Additionally, a rogue GSM network was built and deployed without enabling ciphering and frequency hopping. This active attack was purely based on open-source software and hardware, implying that real GSM networks could be spoofed with resources available to the general public.

Preface

This Master's Thesis is the result of our work in the 10th semester of the 5-year MSc in Communication Technology at the Department of Telematics, Norwegian University of Science and Technology. The work has been carried out in the spring of 2010 in Trondheim, Norway.

The motivation and aim for this thesis was originally a research into the rainbow table generation project's attack against GSM. However, the GSM standard proved to be a complicated matter that took us some time to fully understand. The GSM specifications constitute well over 1000 documents; and although a considerably large part of the specifications might be redundant, it is way too incomprehensible for the average person. Consequently, the first part of this thesis attempts to describe the specifications, which will later be verified with real-world captured signals. The study can be seen as a preliminary step in the process of capturing known plaintext and speech traffic. Hopefully, this work can be informative and useful as a starting point for people wanting to learn more about the theoretical and practical aspects of GSM.

We would like to thank our professors Stig Frode Mjølsnes and Danilo Gligoroski for feedback and great support during the work. We really appreciate their vast knowledge and skill in key areas such as cryptography, wireless security, ethics, and their assistance in writing reports.

Gratitude also to Pål Sturla Sæther and Asbjørn Karstensen at ITEM for fulfilling our many hardware requests.

A final thank you to Frank A. Stevenson for including us in the A5/1 Security Project and participating in valuable discussions.

- Magnus Glendrange, Kristian Hove and Espen Hvideberg

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Security is a important issue, especially in today's technologically advanced society. GSM is a world-wide standard for digital wireless mobile telephones, currently used by 4.4 billion[14] people. These people use their phones to store, transmit and receive voice and data communications. Unlike a fixed telephone, which offers some level of physical security, a wireless link enables anyone with a receiver to passively intercept the traffic. Because of this, it is highly important that security measures are taken to ensure the confidentiality of users' phone calls and SMS messages.

Despite the vast popularity, the security aspects of GSM has not received the scrutiny it deserves. Theoretical weaknesses have been known for years, with research ranging from the relatively insecure encryption algorithms to the lack of network authentication. Still, very little effort has been put into practical research to confirm the implementations we daily rely on.

Analyzing and capturing GSM traffic was up until recently an unexplored area. The main reasons being the complex signaling mechanisms involved, the expensive hardware requirements and lack of interception software. However, things may be about to change with the emergence of open-source tools, such as the USRP, GNU Radio, AirProbe and OpenBTS. These tools enables anyone with interest in GSM security to investigate the theoretical security principles through practical approach.

1.2 Limitations

There are several angles of attack against GSM; on the ciphering algorithm itself and on the system as a whole with its attached equipment. However, this thesis will only cover the following two attack scenarios:

1. The passive attack utilizing rainbow tables to crack the ciphering algorithm and the USRP together with software for passive interception of traffic.

2. The active attack utilizing the fact that authentication of the network is not performed.

Regarding the architecture of GSM, the main target (of this thesis) is the air interface between the Mobile Station (MS) and the Base Transceiver Station (BTS). The network side of GSM will not be explored in detail.

1.3 Structure

This thesis is divided into 9 chapters. Chapter 1 introduces the purpose and background of our work, limitations and ethical considerations. Chapter 2 provides a GSM foundation, making the reader capable of understanding the

subsequent chapters. Some sections in this chapter may be left out if the reader feels familiar with the topics presented. Chapter 3 presents the A5/1 cipher, as well as the theory behind the attack on A5/1, including an introduction to time-memory trade-off.

Chapter 4 contains the creation of rainbow tables, as well as subsequent utilization by doing lookups. Chapter 5 covers the use of a Nokia 3310 to acquire network information, while chapter 6 presents the use of a USRP, together with GNU Radio and Airprobe, to intercept GSM signaling traffic. The last experiment can be found in chapter 7. This covers the deployment of a rogue GSM network by utilizing the USRP, GNU Radio, OpenBTS and Asterisk.

Chapter 8 discusses the findings and suggests possible future work. This is followed by a conclusion in chapter 9. The report also provides 6 appendices, as well as a list of acronyms at the end.

1.4 Ethical Considerations

During an ethical discussion the authors decided that operating within the legal framework had the highest priority. There was consensus on the fact that cracking somebody else‘s GSM traffic should not be performed. Here are some of the legal implications in Norway:

- GSM security research is allowed
- Receiving GSM traffic is (technically) allowed
- Decoding (e.g. cracking) your own GSM traffic is allowed
- Decoding somebody else's GSM traffic is illegal
- Setting up a BTS is allowed if you acquire a license. This is applied for through the Norwegian Post and Telecommunications Authority (NPT).

Figure 1.1: The Eavesdropping Issue

Chapter 2

GSM

This chapter gives an overview of the basic functions and main entities in the GSM network. The protocol stack is first examined and studied, before the relationship between the entities and their addresses is defined. Physical and logical channels are subsequently explained and illustrated over a series of sections. In addition, channel coding, interleaving and frequency hopping are explained to give an understanding of the many complex procedures involved in GSM. Finally, this chapter outlines some mobility management procedures, such as authentication, call and SMS setup.

If this is your first introduction to GSM, a word of warning might be in order at this point. The European Telecommunications Standards Institute (ETSI) provides a separate document consisting of 10 pages for all the acronyms used in the GSM specifications [36]. This thesis will sadly not be much better. A full list of acronyms is given at the end of the thesis.

2.1 Background

GSM (Global System for Mobile Communications) is a digital cellular technology used for transmitting voice and data services. GSM allows users to roam seamlessly from one network to another, while also providing personal mobility. In addition, both speech and signaling channels are digitalized, which essentially labeled GSM as the second-generation (2G) mobile system. Since its first launch in 1991, GSM rapidly became the most popular mobile phone system in the world. In June 2010, an estimated 4.4 billion subscribers across more than 219 countries were using GSM or 3GSM [14].

2.2 Architecture

The architecture of GSM with its entities and interfaces is shown in figure 2.1. A single GSM network established and operated by a service provider is referred to as a Public Land Mobile Network (PLMN). In the following section, the key entities in a PLMN are briefly described.

Mobile Station (MS)

The MS is used by the subscriber as a communication device in the GSM network. The term MS include the physical phone itself, called the Mobile Equipment (ME), and the Subscriber Identity Module (SIM). The SIM ensures secure access to a particular network as it contains algorithms used for authentication (A3) and for generating an encryption key (A8). It also contains subscriber related information, such as the user name or, more accurately, the International Mobile Subscriber Identity (IMSI), the secret authentication key K_i, and lists of preferred and forbidden PLMNs.

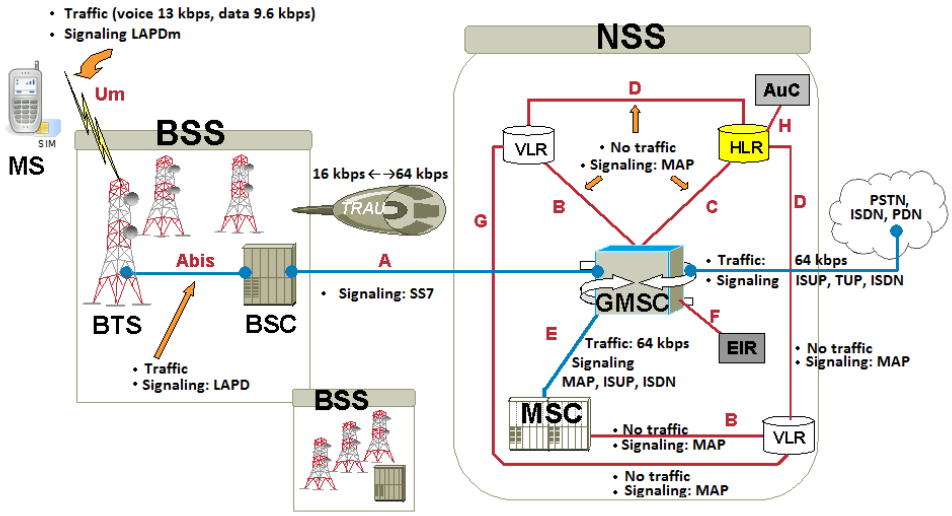

Figure 2.1: Architecture of GSM.

Base Station Subsystem (BSS)

The BSS controls all the radio related tasks and provides connectivity between the network and the MS via the air interface. The BSS is composed of two parts, the Base Transceiver Station (BTS) and the Base Station Controller (BSC). The BTS acts as the communication point for the MS. It transmits and receives signals to and from the MS, while also handling multiplexing, power control, modulation, speech encoding/decoding and ciphering of these signals. The interface between the MS and BTS is known as the Um interface (or more precisely the air interface). The BSC provides the intelligence in a BSS. It controls a set of BTSs and manages handover decisions, radio channels, paging coordination and other needed control functions. It communicates with the BTSs over what is named the Abis interface.

Network Switching Subsystem (NSS)

The NSS controls multiple BSSs and its main role is to set up the communications between two users. It also includes databases needed for additional subscriber and mobility management. The various components of the NSS are described in the remainder of this section.

Mobile Switching Center (MSC)

The MSC is the core component of any NSS. This component controls several BSCs and is responsible for routing of incoming and outgoing calls. It also provides the management functions for terminal mobility such as registration, authentication, location information and handover.

Transcoder and Rate Adaptation Unit (TRAU)

The TRAU is responsible for compressing voice communication at the air interface. It is placed between either the BTS and BSC or between the BSC and MSC.

Home Location Register (HLR)

The HLR is a database storing the subscriber specific parameters. Information in the HLR consists of the telephone number allocated to the subscriber, their current location, K_i, allowed services and the IMSI.

Visitor Location Register (VLR)

The VLR is a database designed to limit the amount of inquiries made to the HLR. When a subscriber roams away from his own network, information is forwarded from the subscriber's HLR to the VLR of the serving network. A VLR can either be allocated to several MSCs, or co-located with a single MSC.

Gateway Mobile Switching Center (GMSC)

The GMSC is an exchange between the PSTN and GSM network that recognizes mobile telephone numbers and is equipped with the capability to access the HLR for routing assistance. This entity is needed as the fixed network is unable to connect an incoming call to the local target MSC (due to its inability to interrogate the HLR).

Short Message Service Gateway (SMS-G)

The SMS-G (not shown in figure 2.1) is the term used to collectively describe the two gateways for handling SMS messages headed in different directions. The SMS-GMSC provides a temporarily 'store and forward' mechanism for incoming SMS messages. If the recipient is not available, it queues the SMS for later retry. The SMS-IWMSC (Short Message Service Inter-Working Mobile Switching Center) is used for outgoing SMS messages.

Authentication Center (AuC)

The AuC holds a secure database storing identification and authentication information related to each subscriber. The attributes in this database include the subscriber's IMSI, K_i and Location Area Identity (LAI). The AuC is responsible for generating authentication triplets of values consisting of a random number (RAND), a signed response (SRES), and session key K_c.

Equipment Identity Register (EIR)

The EIR is a database storing three lists: the white, black, and the gray list. The white list contains all equipment identities that are permitted for communication. The black list contains all equipment identities that are denied. MEs appearing in the gray list are not necessarily denied, but are tracked for specific purposes.

2.3 Protocol Stack

This section offers a short introduction to the protocol stack in GSM as shown in figure 2.2. The protocol stack consists of three layers: the physical layer (layer 1), the data link layer (layer 2), and the network layer (layer 3). This section only looks at the bits sent or received by the MS. A further look into the protocol stack from the network side will not be explored.

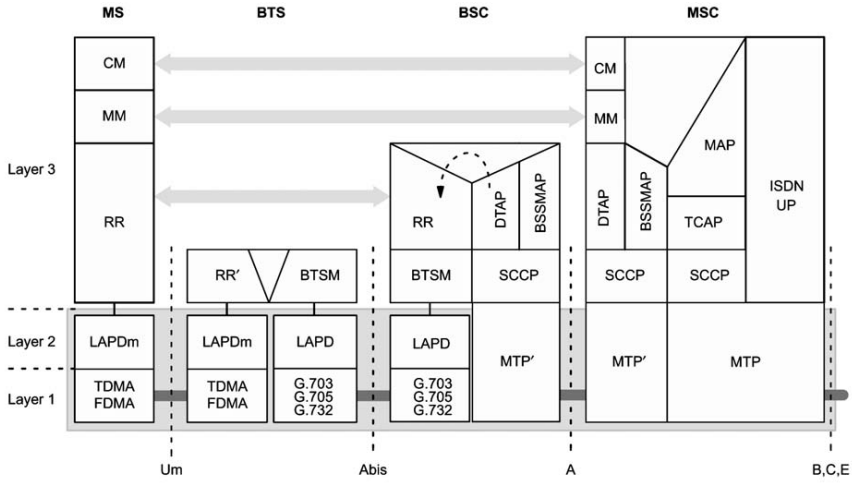

Figure 2.2: Protocol Stack of GSM [30].

Physical layer (layer 1)

All techniques and mechanisms used to make communication possible on the air interface represent the physical layer procedures [40]. These procedures consist of synchronization, power control, coding, and other functions that control the establishment and maintenance of the radio channel.

The physical layer is a combination of frequency-division multiple access (FDMA) and time-division multiple access (TDMA) to allow users to access

the radio resources in a cell. With FDMA, the frequency band is divided into carriers or channels, each 200 kHz wide. Each channel is divided into eight separate timeslots (labeled T0 to T7). This division of frequencies in the time domain is referred to as TDMA. These eight timeslots together is called a TDMA frame. A subscriber is allowed to utilize one timeslot each frame. The duration of a frame is 4.615 ms, thus one single timeslot lasts only 0.5769 ms.

There are thirteen different frequency bands defined in GSM [13]. However, the 850 MHz, 900 MHz, 1800 MHz, and 1900 MHz bands are the most commonly used. The frequency bands employed within each of the four ranges are similarly organized. They differ essentially only in the frequencies, such that various synergy effects can be taken advantage of; hence we only give some details for the usage in the 900 MHz band.

In the 900 MHz band, a total of 70 MHz bandwidth is allocated, two 25-MHz frequency bands for up- and downlink and a 20 MHz unused guard band between them. The MS transmits in the 890- to 915-MHz range (uplink) and the BTS transmits in the 935- to 960-MHz (downlink) band. This corresponds to 124 duplex channels, where each channel within a BTS is referred to as an Absolute Radio Fequency Channel Number (ARFCN). This number describes a pair of frequencies, one uplink and one downlink, and is given a channel index between C0 and C123, with C0 designated as the beacon channel. An ARFCN could be used to calculate the exact frequency (in MHz) of the radio channel. In the GSM 900 band, this is computed by the following equations:

$$F_{uplink}(n) = 890 + 0.2n, \qquad 1 \leq n \leq 124$$

$$F_{downlink}(n) = F_{uplink}(n) + 45, \qquad 1 \leq n \leq 124$$

Similar formulas are also defined for the other GSM frequency bands.

Data link layer (layer 2)

The data link layer is responsible for the correct and complete transfer of frames between the network layer entities over the air interface. The signaling protocol used in this layer, LAPDm, is a modified version of the LAPD (Link Access Procedures on the D-channel) which is particularly adapted to the TDMA burst structure of GSM. The term 'DM channel' is referred to a collection of all the various signaling channels required in GSM.

LAPDm implements the following basic functions [37]:

- The organization of layer 3 (network layer) information into frames.
- The establishment, the maintenance, and the termination of one or more data links on the Dm channel.
- Sequence control in order to maintain the sequential order of frames across data link connections.
- Detection of format and operational errors on a data link.
- Flow control.
- The acknowledged transmission and reception of numbered information (I) frames.
- The unacknowledged transmission and reception of unnumbered information (UI) frames.

Error detection and correction schemes are provided by a combination of block and convolutional coding used in combination with interleaving at the physical layer. This is explained in more detail in section 2.9 and 2.10.

The total length of a LAPDm frame is always 184 bits (23 octets), with segmentation for larger messages. Appendix H will look further into the different types of LAPDm frames.

Network layer (layer 3)

The network layer contains all the functions necessary to establish, maintain, and terminate mobile connections [31][32]. The network layer also provides the control functions to support additional services such as the short message services (SMS). The three sublayers in layer 3 offer services as follows:

- Radio Resource Management (RR): The RR sublayer is responsible for configuration of the physical and logical channels at the air interface. This includes management of frequencies, timeslots, channel configuration, power control, synchronization, and frequency hopping codes. Another important RR procedure is the activation of ciphering
- Mobility Management (MM): The MM sublayer manages subscribers and tracks their movements. These procedures are exclusively performed in cooperation between the MS and MSC. This include:
  - Location update: storing and tracking of the subscriber's location.
  - Mobile authentication: a challenge-response mechanism.
  - IMSI attach: the location update procedure (e.g., when the MS is powered on).
  - IMSI detach: tells the network that the MS is no longer in service (e.g., when the MS is powered off).
  - TMSI reallocation: ensures the confidentiality of a subscriber's IMSI.
- Connection Management (CM): divided further into three sublayers:
  - Call Control (CC): establishment, maintenance and release of calls.
  - Short Message Service (SMS): management of SMS messages.
  - Supplementary Services (SS): handling of all additional services that are not connected to the core functionality of GSM.

2.4 Addressing

GSM specify both user and equipment addressing. Besides these identifiers, several other codes used for characterizing location areas have been defined; they are needed for the mobility management and for addressing network elements. The most important types of addresses and identifiers are presented in this section.

International Mobile Subscriber Identity (IMSI)

The IMSI is a number (up to 15 digits) that uniquely identifies a subscriber within the global GSM network. As shown in figure 2.3, the IMSI consists of the Mobile Country Code (MCC), Mobile Network Code (MNC) and the Mobile Subscriber Identification Number (MSIN). MSIN uniquely identifies a user within the home GSM network. The MCC and MNC combined are used to uniquely identify a GSM PLMN in a country (e.g., 242 as MCC for Norway; and MNC 01, 02 and 05 for the networks of Telenor, NetCom and Network Norway, respectively).

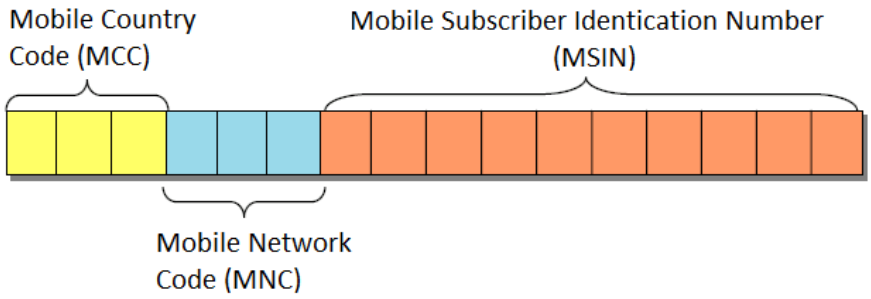

Figure 2.3: Structure of the IMSI

The IMSI is used by the network for identifying a (roaming) subscriber, as well as charging, billing and accounting. This identification is stored in the SIM, AuC, HLR or locally copied to responsible VLR.

Whenever a MS is switched on, it may have to perform an IMSI attach procedure, which results in the transmission of the IMSI in cleartext. This is necessary in order to find the subscription data. When registered, the MS is assigned a temporary identification called the Temporary Mobile Subscriber Identity (TMSI) within the area. This is a 32-bit code used in subsequent location updates, paging and call attempts as a security mean to make it more difficult to track and identify a certain user. The mapping between IMSI and TMSI is kept by the VLR.

Mobile Subscriber ISDN Number (MSISDN)

The MSISDN uniquely identifies a subscription in GSM and is usually the telephone number that a person would dial in order to reach another subscriber. The MSISDN follows the ITU-T E.164[50] recommendation as shown in figure 2.4. It is composed of up to 15 digits; the Country Code (CC), the National Destination Code (NDC) and the Subscriber Number (SN). The CC identifies a country or a geographic area the MS is registered to, whereas the NDC identifies the PLMN within a country. SN is the number assigned to the subscriber by the service provider. A subscriber is assigned a new MSISDN for each associated GSM service (one for voice , another for fax etc).

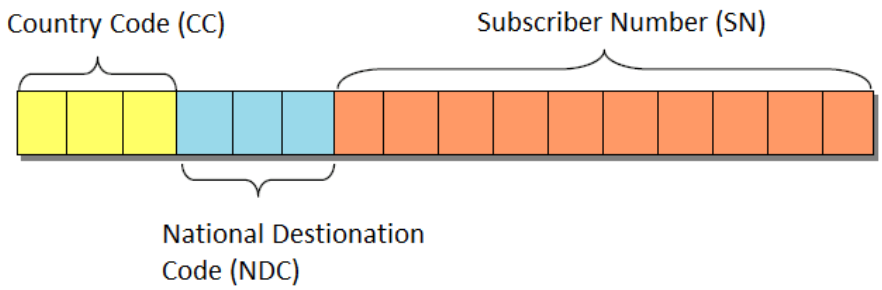

Figure 2.4: Structure of the MSISDN

Mobile Subscriber Roaming Number (MSRN)

The MSRN is a temporary, location dependent ISDN number with the same structure as the MSISDN. Figure 2.5 confirms the identical structure. The MSRN is assigned to the MS by the local responsible VLR and used to route calls to the targeted MSC.

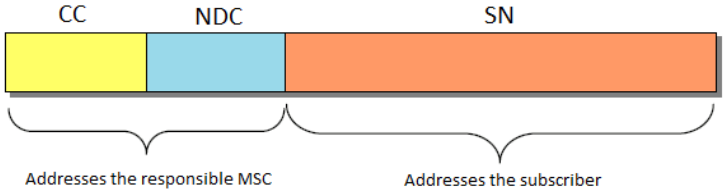

Figure 2.5: Structure of the MSRN

International Mobile Equipment Identity (IMEI)

Every ME has an internationally unique identifier, called the IMEI. This number can be used to deny unlicensed equipment access to the network or track stolen equipment. The IMEI is composed of a Type Allocation Code (TAC), a Serial Number (SNR) and a Spare (SP). The TAC uniquely identifies the model of the ME, the SNR defines the ME's serial number and the SP is a Luhn Check Digit[1]. A newer form of IMEI, named IMEI Software Version (IMEISV) omits the SP and instead adds a Software Version Number (SVN) at the end. The SVN identifies which software version the ME is running. The composition of both the IMEI and IMESI is shown in figure 2.6.

[1]The Luhn check digit is calculated from the rest of the IMEI and used to verify the theoretical integrity of that number

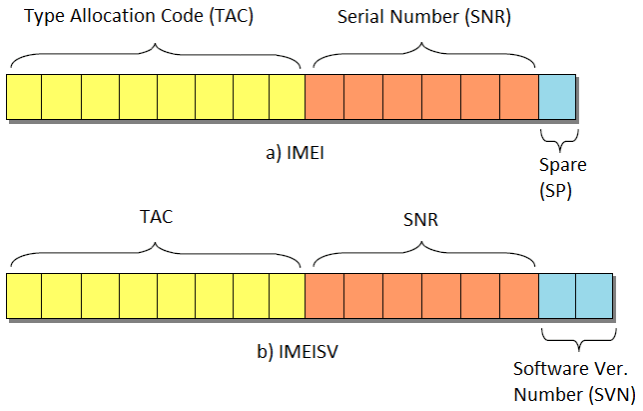

Figure 2.6: Structure of the IMEI and IMEISV

Location Area Identifier (LAI)

Each GSM network is subdivided into location areas identified by a unique LAI within the network. This number consists of the MCC, MNC and a Location Area Code (LAC) as seen in figure 2.7. The LAC is a number of maximum of five digits which identifies a location area within a PLMN. This is used as a unique reference for the current location of a subscriber. The location area is further divided into cells, each being uniquely identified by a Cell Identifier (CI).

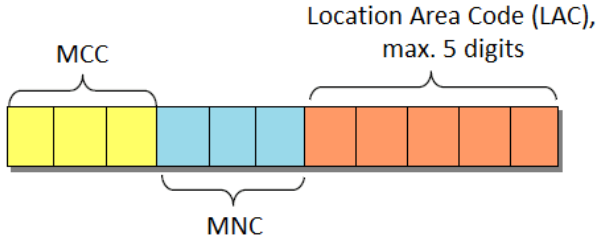

Figure 2.7: Structure of the LAI

Base Station Identity Code (BSIC)

The BSIC is a number that identifies a BTS. The code is 6 bits in length; a 3 bit Network Colour Code (NCC) that identifies the network and a 3 bit Base Station Colour Code (BCC) that identifies the BTS within a certain area. The BSIC is used by the MS to distinguish between BTSs broadcasting on the same frequency. The same code is then sent back to the network by the MS in all its measurement reports. This allows the network to confirm that the MS is tuned to the correct frequencies.

2.5 Logical channels

In the GSM terminology, a distinction is made between physical and logical channels. Physical channels can be described in terms of the frequency and time domain. They are the actual frequencies and the timeslots the MS or BTS are transmitting or receiving on. The logical channels are mapped onto these physical channels. At any particular instant a frequency/timeslot may be either a traffic channel or some control channels. In other words, a logical channel describes the operation and function of a physical channel at a given point in time [42].

There are two types of logical channels that are required to organize GSM: Common Channels (CCH) and Dedicated Channels (DCH). As the figure 2.8 illustrates, the CCH consist of Broadcasting Channels (BCH) and Common Control Channels (CCCH), whereas the DCH is divided into Dedicated Control Channels (DCCH) and Traffic Channels (TCH).

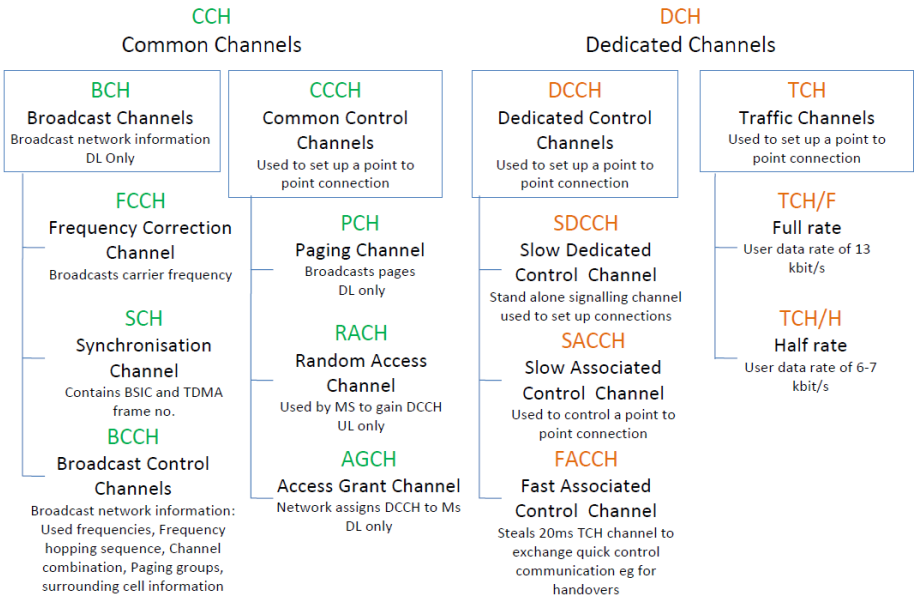

Figure 2.8: Logical channels in GSM

Broadcasting Channels (BCH)

The BCH are defined for downlink only and are used by the BSS to send out the same information to all MSs in a cell. They include the following three subtypes of channels:

- Frequency Correction Channel (FCCH): used for broadcasting frequency synchronization signals that enables the MS to synchronize its carrier frequency and bit timing with the BTS. The MS will always scan its known beacon frequencies to find this channel. Having detected the particular signal, it knows that the next timeslot of the same channel contains the synchronization channel (SCH).

- Synchronization Channel (SCH): broadcasts frame synchronization signals

containing the TDMA frame number (FN) and the BSIC. This allows the MS to synchronize in time with the BTS, and to identify the cell.

- Broadcast Control Channel (BCCH): used to inform the MS about specific system parameters such as location area and network codes, the frequency hopping sequence, surrounding cell information, particular information about the channel configuration, and maximum power level. This channel is always sent in a repeating cycle on the same frequency.

Common Control Channels (CCCH)

The CCCH are mainly used for carrying signaling information necessary for accessing management functions (e.g., allocation of DCH or radio resource on a traffic channel). They are divided into three channels:

- Paging Channel (PCH): a downlink channel used to search or 'page' the MS to inform that there is incoming traffic. The MS could be addressed by its allocated TMSI or IMSI. However, the GSM specifications does not allow paging by IMEI [31].
- The Random Access Channel (RACH): the uplink counterpart to the PCH, typically used when the MS initiates a request to the network, (e.g., to make a call or send a SMS message). It is accessed by the MS in a competitive multiple-access mode using the principle of slotted Aloha [30].
- Access Grant Channel (AGCH): used to set up a connection once a MS has been paged on the PCH or initiated a request on the RACH. It serves the purpose of inviting the MS to another control channel, usually the SDCCH, where the actual signaling set up is performed.

Dedicated Control Channels (DCCH)

The DCCH are used for power control, timing advance and other call related information, as well as various types of traffic channels. They are made up by the following types of channels:

- Standalone Dedicated Control Channel (SDCCH): a two-way signaling channel used for exchange of messages associated with call establishment, authentication, location updating, SMS and other management functions.

- Slow Associated Control Channel (SACCH): always associated with either a SDCCH or a traffic channel, and used to inform the MS about the frequencies of neighboring cells, time alignment and power control on the downlink. For the uplink it is used for transmitting field strength measurements as input parameters to a possible handover decision. It can transmit SMS messages if associated with a traffic channel. This is why SMS messages can be received while being busy in a call.

- Fast Associated Control Channel (FACCH): always associated with a traffic channel and used to transmit urgent signaling messages. Its usage goes at the expense of speech data, as it "steals" timeslots from its associated traffic channel when the SACCH can not operate quickly enough. The FACCH is referred to as "fast" because it can carry up to fifty signaling messages per second against four per second for SDCCH [57].

Traffic Channels (TCH)

TCH are used for transmitting voice and data traffic. They are either full rate (22.8 Kbps) or half rate (11.4 Kbps). A full rate channel occupies a complete timeslot per frame, whereas a half rate channel occupies only one timeslot in every second frame.

2.6 Burst formats

There are five different types of bursts used for transmission in GSM [39]. These bursts each have a distinct format structure as shown in figure 2.9. A burst has a duration of 0.5769 ms and is typically recognized by which logical channel it is transmitted on. The GMSK scheme used at the air interface provides a modulation rate of 270.833 kbit/sec, resulting in transmission of 156.25 bits in one burst. A short introduction to the various burst types are given below:

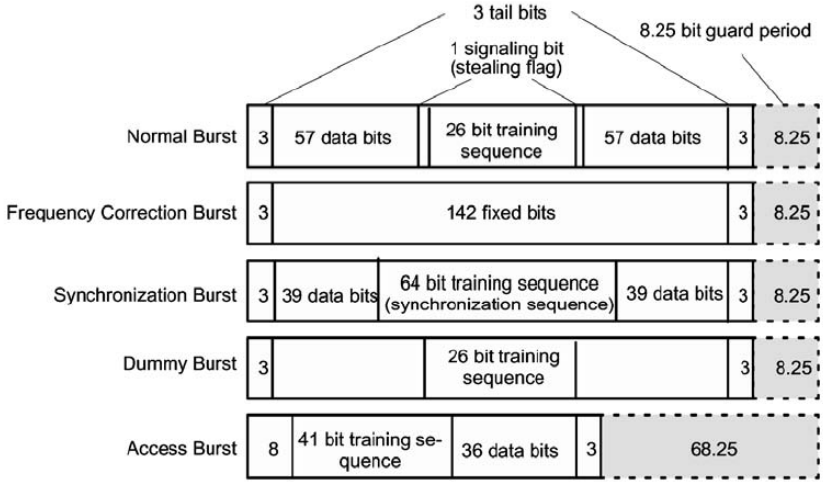

Figure 2.9: Burst Types in GSM. Adapted from [30].

- The normal burst is used to transmit speech, data and signaling information. It is made up by a three-bit tail at each end, two data fields (payloads) consisting of 57 bits each and a mid-amble consisting of two stealing bits and a training sequence. The two stealing bits are used to indicate whether the burst contains traffic (data) or signaling information. The training sequence is for equalization, i.e. to get the BTS and MS in 'tune' with each other. It also determines the start of data field, including

the data itself. Finally, there is a guard time between bursts which is used to separate data packets to avoid that adjacent bursts overlap.

- The frequency correction burst is used for frequency synchronization of the MS. This burst is periodically transmitted by the BTS on the FCCH, allowing the MS to find the beacon frequency.
- The synchronization burst is sent from the BTS on the SCH and allows the MS to synchronize with the TDMA frame. It has two data fields of 39 bits; composed of the FN and BSIC.
- The dummy burst is transmitted by the BTS when no other bursts are to be transmitted. The purpose of this burst is to allow the MS to perform signal power measurements.
- The access burst is transmitted by the MS on the RACH when it wants to request a service (e.g., to establish a call or perform a location update). In order to avoid that the access burst overlaps with other bursts, these bursts are shorter than normal bursts. The BTS can estimate the distance to the MS by determining how much the burst is delayed relative to the burst sent from the BTS. It then tells the MS to start its transmission time a number of bits earlier in order to compensate for the delay (and to ensure arrival within the correct timeslot). This is known as a timing advance procedure [41][15]. MS advances its burst transmission by up to 64 steps (0-63); each step corresponds to a distance of about 550 meters away from the BTS.

2.7 Channel Combinations

The GSM specification permits only certain ways in which logical channels can be mapped onto a physical channel [39]. These combinations are numbered from I to VII and are transmitted on the BCCH where the BTS indicates the used combination. This allows the MS to find the various information elements multiplexed on the channel.

The various combinations are:

I TCH/F + FACCH/F + SACCH

II TCH/H + FACCH/H + SACCH.

III 2 TCH/H + 2 FACCH/H + 2 SACCH.

IV FCCH + SCH + BCCH + CCCH

V FCCH + SCH + BCCH + CCCH + 4 SDCCH + 4 SACCH

VI BCCH + CCCH

VII 8 SDCCH + 8 SACCH

The first three combinations apply to the configuration of traffic channels. These combinations can be used anywhere except in timeslot 0 on the beacon frequency (C0T0). The other four combinations deal with the multiplexing of control channels. Combination IV is the default standard used in C0T0. It cannot be used anywhere else. The same applies to combination V, and this is often used for small cells. Combination VI is mainly used together with combination IV in order to provide more capacity for paging and access grant. Finally, the eight SDCCH/SACCH pairs in combination VII, which is used to provide additional signaling capacity, can be mapped to a number of timeslots. By convention, this is usually mapped into timeslot 1 on the first carrier frequency (C0T1).

The information transmitted in a timeslot is categorized and scheduled in a

predictable manner, as shown in figure 2.10. The first timeslot (T0) in each sector is always reserved for CCH signaling. The CCH multiplexing follow a 51-frame cycle called a 'multiframe', whereas the TCH follow a 26-frame multiframe cycle. T2 through T7 is normally allocated to TCH. However, notice that TCH does not occur on every single frame within the allocated timeslot. There is one frame reserved for SACCH and one that is idle. Thus, in a traffic multiframe, a total of 24 frames are used for speech data.

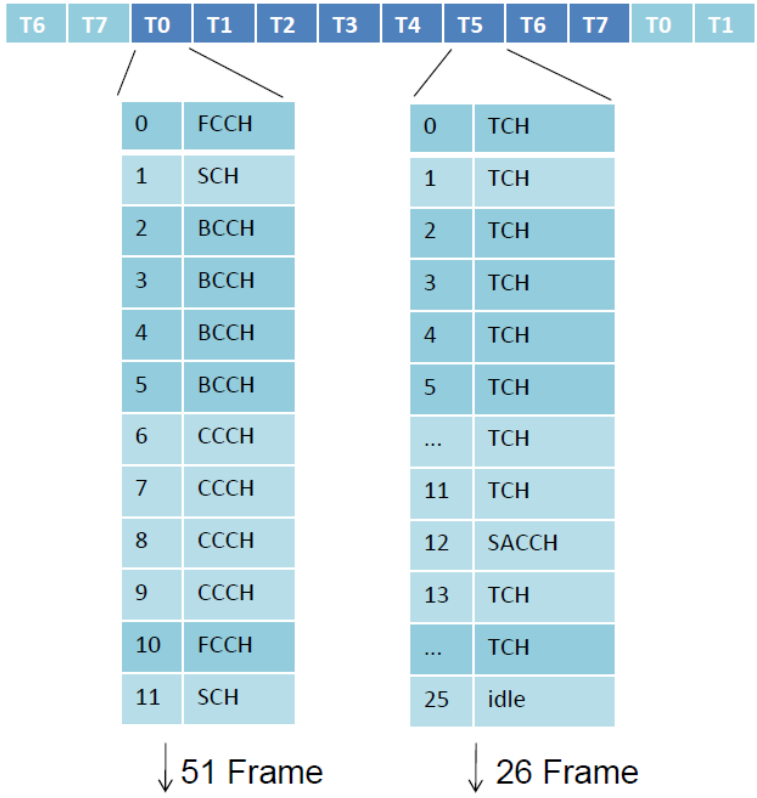

Figure 2.10: Mapping of Logical Channels in a GSM Multiframe

Although GSM uses full duplex channels, the BTS and MS do not transmit at the same time, as illustrated in figure 2.11. The transmit time of the MS is exactly three timeslots later than the corresponding timeslot from the BTS. The arrangement is called time-division duplex (TDD) and has the advantage that the MS does not need to transmit and receive information simultaneously.

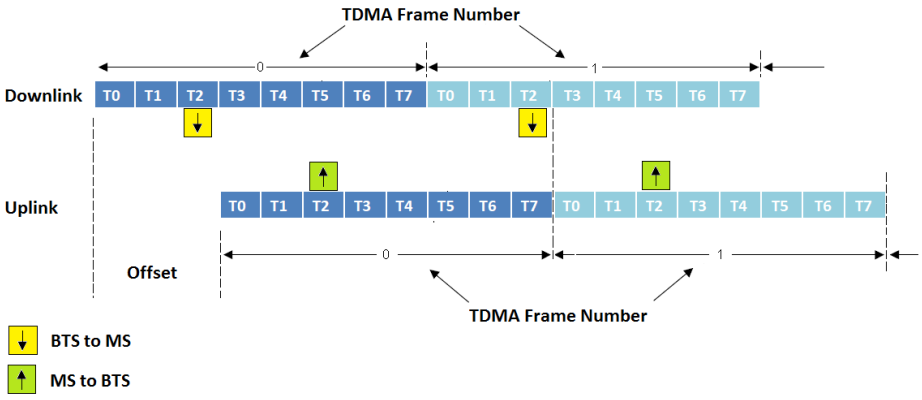

Figure 2.11: Time-Division Duplex in GSM

For instance, if the MS is allocated a traffic channel in T2, the BTS transmits when the downlink is in T2 and the MS sets to receive in T2. At this point, the uplink is three timeslots behind. Once the uplink reaches T2, the MS begins to transmit, and the BTS sets to receive in T2. At this point, the downlink is at T5. When the MS is neither transmitting nor receiving, it monitors the BCCHs of adjacent cells.

2.8 Speech Coding

GSM is a digital communication standard. As the human voice is of an analog quantity, it has to be converted into a digital bit stream[2].

The analog voice is digitized (in the ME) by an analog-to-digital converter (ADC) at a rate of 8000 samples per second. Each sample is quantized with a resolution of 13 bits, resulting in a bit rate of 104 kbps. This bitrate is far too high to be transmitted over the air interface, so it has to be reduced to a bitrate of maximum 13 kbps. A speech encoder is used to compress the speech signals. In GSM, this can be performed by a variety of speech encoders: The Regular Pulse Excitation - Linear Predictive Coding (RPC-LPC), Enhanced Full Rate (EFR) or the most widely used Adaptive Multi-Rate (AMR). A detailed discussion of these procedures are given in [63]. In essence, they use vocal characteristics and previous samples (information that does not change very often) to produce speech frames of 260 bits representing 20ms of audio, corresponding to a compression ratio of 1 to 8.

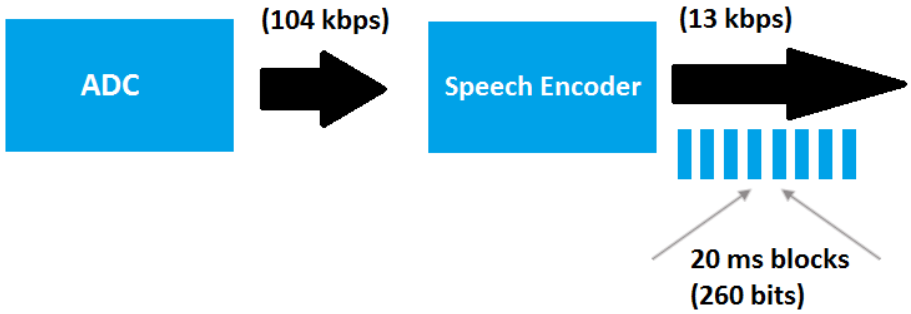

Figure 2.12: Speech Encoding in GSM

[2]If, however, the source of information is signaling or data, the speech coding is not performed

2.9 Channel Coding

Once a compressed digital signal is made, redundancy bits are added to protect the signal from interference. GSM implements block and convolutional coding to achieve this protection. The specific algorithms differ for speech and for signaling channels.

A speech sample, consisting of 260 bits, is first divided into three parts (class Ia, class Ib and class II bits) according to function and importance. The most important class being Ia where 50 bits are assigned. Class Ib and Class II are assigned 132 bits and 78 bits, respectively. Three parity bits are computed for Class 1a and added to the Class Ib bits. An additional 4 bits are then added to the Class I bits (Ia and Ib combined). These bits are all zeros and are needed for the actual convolutional encoding. The encoding outputs two bits for every input bit, thus the number of class I bits are doubled from 189 to 378. Finally, the 78 least significant bits are added without protection, resulting in a speech block of 456 bits.

As for signaling information consisting of 184 bits, a 40 bits fire code is added in order to detect and correct burst errors. In addition, a 4-zero bits is employed before the signal is passed through the same convolution encoding as the speech. The output is also here a block of 456 bits.

The complex coding schemes of the speech channel and the combined broadcast and common control channel are shown in figure 2.13.

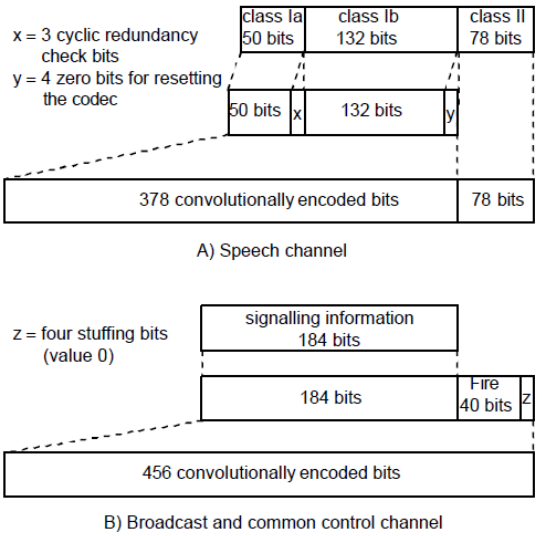

Figure 2.13: Channel Coding in GSM. From [15]

2.10 Interleaving

The channel coding will be not be of any use if the entire 456-bit block is lost or corrupted. One way to alleviate this is to use interleaving. Interleaving spreads the bits into many bursts such that errors can be corrected by simple forward error correction methods.

A 456-bit speech block is partitioned into eight sub-blocks of 57 bits each. The first sub-block of 57 bits contains the bit numbers (0, 8, 16, ...448), the second the bit numbers (1, 9, 17, ...449). Finally, the eighth and last sub-block contains the bit numbers (7, 15, ...455). These sub-blocks are then interleaved onto eight separate bursts.

The first four sub-blocks are mapped onto the even-numbered bits of four consecutive bursts. The other four sub-blocks are mapped onto the odd-

numbered bits of the next four consecutive bursts. Since each normal burst can carry two 57 bit blocks, it contains traffic from two separate 456-bit speech blocks, as seen in figure 2.14.

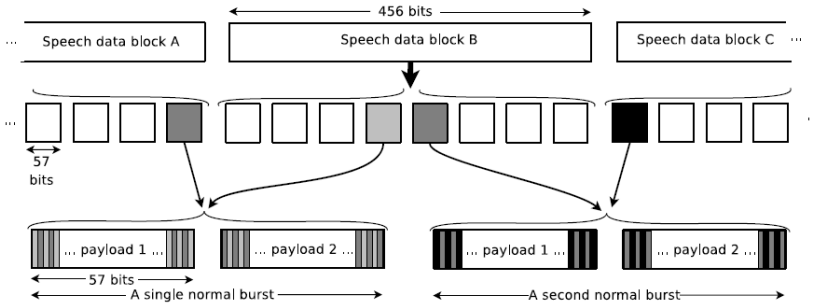

Figure 2.14: Interleaving for Speech Blocks in GSM. Adapted from [66]

Most signaling blocks (or frames) used on the BCCH, SACCH, SDCCH, AGCH and PCH have an identical interleaving scheme to that used for the speech blocks. However, they are spread across four rather than eight interleaving bursts, as shown in figure 2.15.

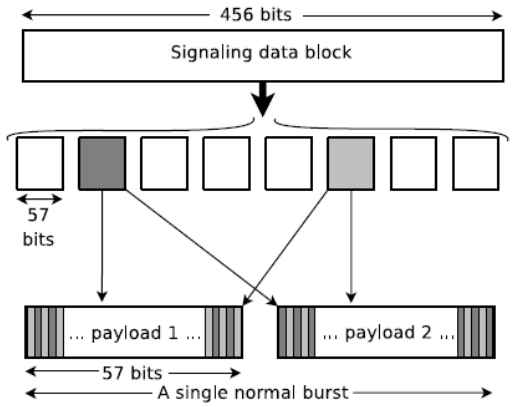

Figure 2.15: Interleaving for Signaling Blocks in GSM. Adapted from [66]

2.11 Frame Structure

A sequence of eight timeslots is known as a TDMA frame. These frames have a duration of 4.615 ms (8 x 0.5769 ms) and are continuously repeated. Several TDMA frames are then grouped together to form multiframes. Exactly 51 frames for the control channels (3060/13 ms) and 26 frames for the traffic channels (120 ms). They are organized in this way to make it possible to establish a time schedule for when a particular logical channel can use a physical channel. Since the two types of multiframes have different lengths, their duration will not be the same. However, since the numbers 26 and 51 are relatively prime, an idle TDMA frame (the 26th) in the traffic multiframe will coincide with every other TDMA frame in the control multiframe. This happens once before the entire sequence is synchronized as a superframe. The idle frame period allows the MS to listen to control channels or perform other necessary operations such as measuring the received signal strength from neighboring cells.

So the multiframes are merged to form superframes, consisting of either 51 traffic multiframes or 26 control multiframes. The two types of superframes will thus have the same duration (6120 ms).

2048 superframes make up a hyperframe. Each TDMA frame is numbered according to its sequence within the hyperframe, starting from 0 and ending at 2,715,647. This is used as an input to the encryption algorithm over the air interface and for generation of the slow frequency hopping sequence. Since the FN changes from burst to burst, each burst is encrypted individually. However, the FN repeats itself after 3 hours, 28 minutes, 53 seconds and 760 milliseconds.

By structuring the signaling and speech into frames, multiframes, superframes and a hyperframe; the timing and organization is set into an fixed format that enables both the MS and BTS to communicate in a efficient and timely manner. The frame structure illustrating the hierarchy of frames is shown in figure 2.16

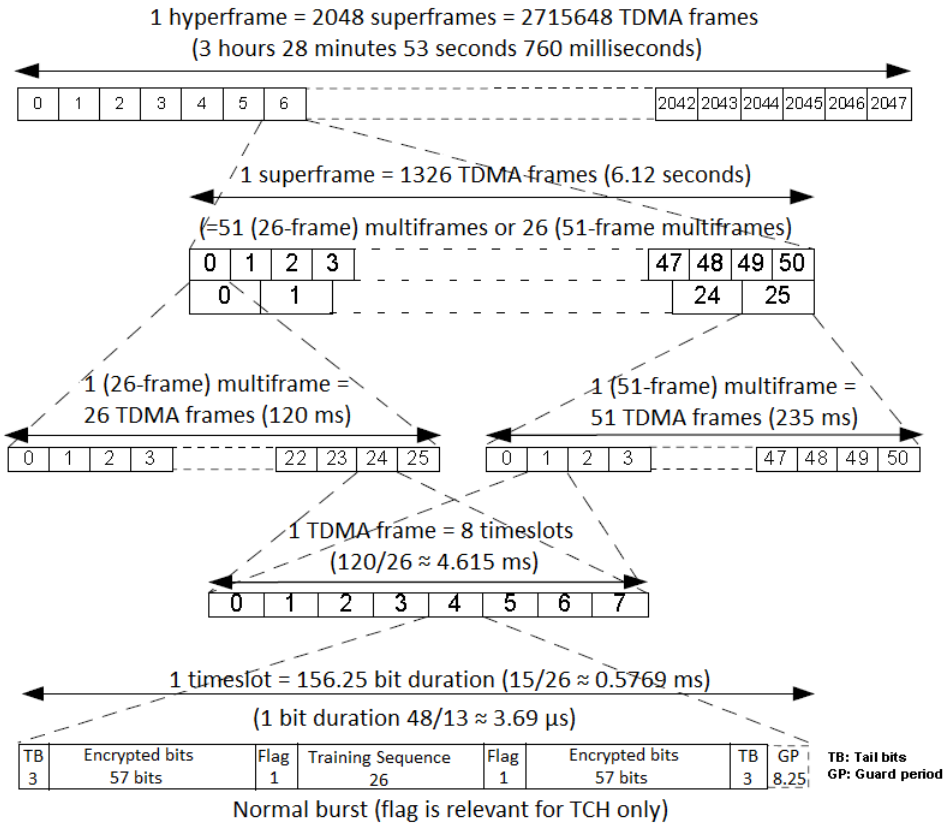

Figure 2.16: Frame Structure in GSM

2.12 Frequency hopping

GSM has introduced an optional frequency hopping procedure, known as slow frequency hopping (SFH). SFH ensures that the interference level is minimized, and that the total traffic in the system can be increased, while still maintaining sufficient quality of each call [15]. In GSM, this is applied to change channel with each burst, meaning that bursts in adjacent frames may appear at different frequencies. The resulting hopping rate is about 217 changes per second, corresponding to the TDMA frame duration.

There are essentially two types of hopping algorithms available to the MS:

- Cyclic hopping: the MS changes transmit frequency in accordance to a predefined list of frequencies in sequential order.
- Random hopping: the MS changes transmit frequency randomly through a set of frequencies.

The hopping algorithm produces the next channel frequency given the current FN, a set of frequencies to hop between; the Mobile Allocation (MA), a start frequency (or more correctly a time delay) within the MA; the Mobile Allocation Index Offset (MAIO), and a parameter used for determining the hopping sequence; the Hopping Sequence Number (HSN). A HSN of zero corresponds to cyclic hopping, whereas values 1 through 63 correspond to different types of random hopping algorithms.

2.13 Authentication

Authentication is one of the most important security functions in GSM [31][35]. It involves several functional components: the SIM card, MSC, VLR and the AuC. The MSC verifies the identity of the subscriber through a challenge-response process. Upon request from the MSC, the VLR returns a authentication triplet[3] consisting of a 128-bit random number (RAND), a 32-bit signed response (SRES) and the session key K_c. When a MS requests service, the MSC challenges it by sending the RAND and a 3-bit Cipher Key Sequence Number (CKSN)[4] in in a *MM Authentication Request* message. The MS must answer this challenge correctly before being granted access to the network.

The RAND is forwarded from MS to the SIM for processing. Figure 2.17 shows the information each entity contains in order to authenticate the SIM. The SIM takes the RAND value and the 128-bit Individual Subscriber Authentication Key K_i and produces a 32-bit signed response (SRES)[5]. This is the MS's response to the challenge which is sent back to the network in an *RR Authentication Response*. If the SRES value is identical to the one given in the authentication triplet, the authentication is successful.

[3]The authentication triplet is originally created in the AuC and sent to the VLR in sets of five.

[4]The CKSN is stored for future *Service Request* messages. If the CKSN sent from the MS is identical to the one stored in the network, the MS is ready to start ciphering without the need for re-authentication

[5]To calculate SRES, an algorithm called A3 is used. It utilizes a hash function called COMP128, which is also used in the generation of a session key needed for confidentiality of calls and data.

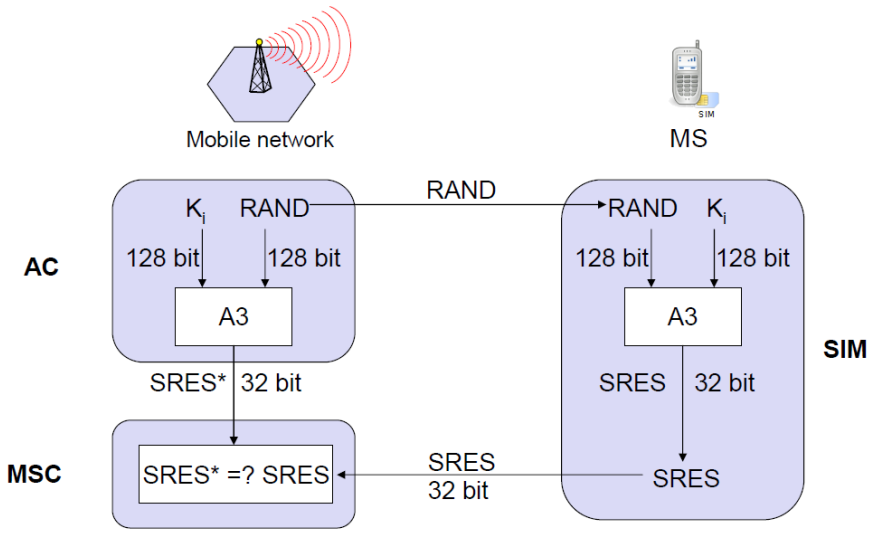

Figure 2.17: Authentication in GSM. Modified from [62]

2.14 Confidentiality

In addition to the parameters needed for the authentication of subscribers, the SIM card also contains parameters needed to provide confidentiality. The figure 2.18 illustrates how the information stream is encrypted over the air interface. An algorithm called A8 is used to generate a session key K_c. By using RAND and the key K_i, the SIM runs the A8 algorithm to produce a 64-bit K_c. K_c is then used by a third algorithm called A5. The A5 is used to produce a key stream of 228 bits from the K_c and the current FN of the timeslot in which the next segment of the message is sent. The key stream is decomposed into two halves. While the first half encrypts the downlink frame, the second half is used to encrypt the uplink frame. For each transmitted frame, a new 228-bit key stream is calculated by A5 to encrypt (and decrypt) the frame.

The A5 algorithm is implemented in the hardware part of the ME, not in the

SIM card. It has to operate quickly and constantly to generate a new set of 228 bits every 4.615 milliseconds. Also, since the terminals are designed to operate in different networks, the A5 must be common to all GSM systems.

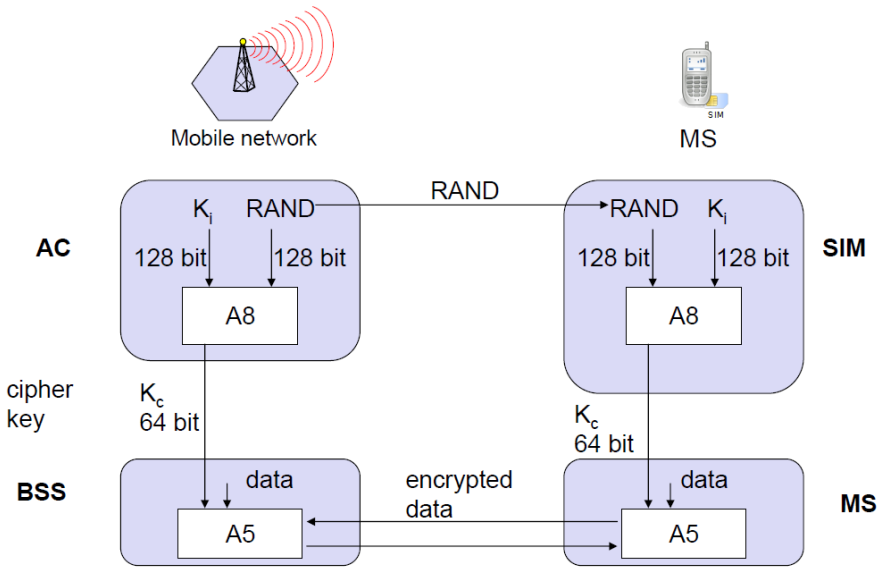

Figure 2.18: Key Generation and Encryption in GSM. Modified from [62]

Network operators have the option between three A5 algorithms for ciphering, namely the A5/1, A5/2 and A5/3. According to the GSM specifications, it is mandatory for A5/1, A5/2 and non-encrypted mode (A5/0) to be implemented on the MS [33]. The network shall not provide service to an MS which indicates that it does not support any of the mentioned ciphering algorithms [43]. A more detailed analysis of A5/1 is found in section 3.2

2.15 Call setup

There is a distinction between a mobile-originating call (MOC) and a mobile-terminating call (MTC) in GSM [31]. A MOC implies that the call originated from a MS, and therefore is an outgoing call. A MTC indicates that the call is ending at the MS, hence an incoming call. Figure 2.19 shows a sequence diagram for the connection setup for both MTC and MOC at the air interface. It also illustrates how the various logical channels are used in principle.

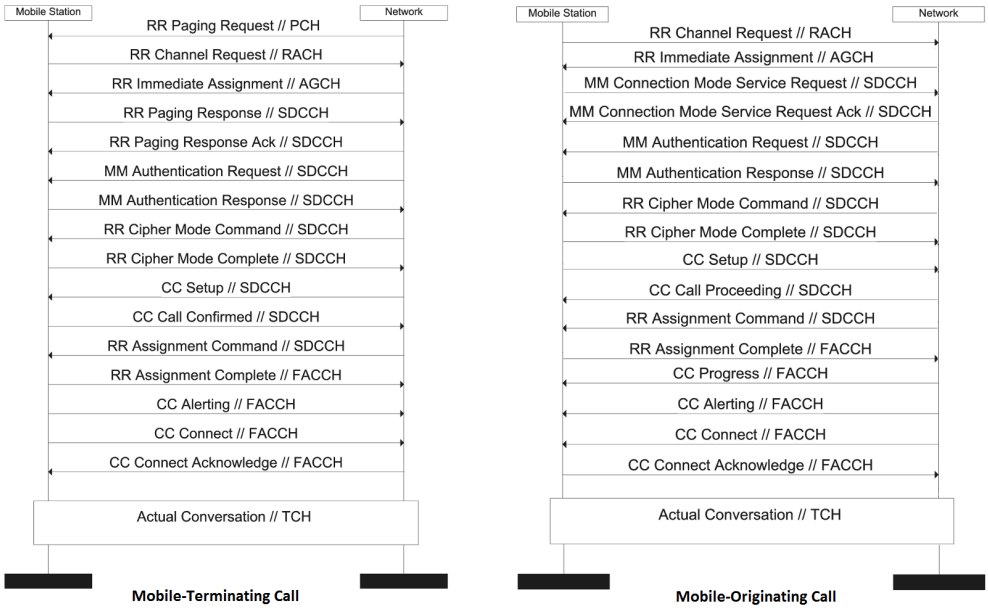

Figure 2.19: Message Flows in a MTC and MOC

Mobile-originating Call (MOC)

The establishment of a MOC starts with the MS initiating a *RR Channel Request* by transmitting an access burst on the RACH. This is sent upling on

the same ARFCN as the BCH. The network responds with an *RR Immediate Assignment* message on AGCH; assigning the MS to an SDCCH. This message also include the FN of when the *RR Channel Request* was received, frequency hopping information and a timing advance parameter.

After the initial setup, the MS sends a *MM Service Request* containing its TMSI/IMSI, CKSN, supported A5 versions and the requested service - in this case a MOC. The network accepts the request with a *MM Service Request Acknowledgment* by including the TMSI for contention resolution purposes.

The network starts carrying out authentication (if needed) as described in section 2.13. This is followed by a ciphering procedure issued by the network using the *RR Cipher Mode Command* message. This is sent to indicate the chosen encryption algorithm and could also be used to request the IMEISV. The MS begins to encrypt (and decrypt), and replies with an encrypted *RR Cipher Mode Complete* message. From this moment on all succeeding communication between the MS and BTS are encrypted.

Once ciphering starts, the MS sends a *CC Setup* message containing the called MSISDN. The network responds with a *CC Call Proceeding* message if the number is valid and sends an *RR Assignment Command* to get the MS off from the SDCCH and onto a TCH+FACCH. The MS acknowledges this transaction with an *RR Assignment Complete* message on the FACCH. From this point on, all signaling transactions are performed on that channel.

An *CC Alerting* message is sent back from the network to indicate that the called subscriber is being alerted (in presence of a ringing tone). If the called party answers, the networks sends a *CC Connect* message confirming that the connection was successfully established. The MS finally replies with a *CC Connect Acknowledge* message, which indicates that it is ready to exchange voice data on the TCH.

The TCH+FACCH assignment can occur at any time during the setup,

depending on the configuration of the GSM network. There is essentially three approaches:

- Late Assignment (Off Air Call Set-Up (OACSU)): A TCH+FACCH is assigned to the MS first when it receives the alerting message.
- Early Assignment (Non-OACSU): A TCH+FACCH is assigned to the MS after it receives the call proceeding message. The call setup is initially performed on the SDCCH, but completed on the FACCH. This is the most common option chosen by operators[24, p. 142] and described in section 2.15.
- Very Early Assignment: A TCH+FACCH is assigned immediately to the MS without using a SDCCH. The entire call setup is performed on the FACCH.

Mobile-terminating Call (MTC)

The process for a MTC is very similar to a MOC. The BTS initiates the transaction by sending a *RR Paging Request* message on the PCH. In reply, the MS transmits a *RR Channel Request* message on the RACH. An *Immediate Assignment* message with the SDCCH number is the response sent by the BTS on the AGCH. The MS replies with a *RR Paging Response* message containing its mobile identity (IMSI or TMSI). The remainder of the call setup is then identical to the MOC as explained in the section 2.15. As with the MOC, the TCH+FACCH assignment can happen at any time.

2.16 SMS setup

The GSM specifications provides information on how SMS messages are transfered between the network and the MS [34]. Similar to the call setup, the

terms mobile-originating and mobile-terminating are also here used to indicate the direction in which the SMS message is sent.

Mobile-originating SMS (MO-SMS)

In a MO-SMS, assuming the delivery is made when no active call is in progress, the MS is assigned a SDCCH applying the same establishment procedure as described in a MOC. A *CM Service Request* message is then sent from the MS shortly after, and it initiates acknowledged mode (multiple frame) with a Set Asynchronous Balanced Mode (SABM) procedure after the cipher mode has been set. The SABM is used as a setup message to establish a data link connection between the MS and BTS.

The MS transmits a *SMS CP-DATA* message containing a *RP-DATA* message as the RPDU. The RP-DATA contains all the needed information for a successful delivery and the actual SMS message. The network replies with a *SMS CP-ACK* message and delivers a *CP-DATA* message to the MS, including the *RP-ACK* payload in the RPDU. The MS answers with a *SMS CP-ACK* message and the network releases the SDCCH with a *RR Channel Release* message.

Mobile-terminating SMS (MT-SMS)

The process for a MT-SMS is in many ways the reverse operation of a MO-SMS. The network starts paging the MS with the standard paging procedure, and the MS is assigned an SDCCH by using the standard *RR Paging Response* reply. The remainder of the SMS setup is then identical to that of a MO-SMS, apart from the flows going in the opposite direction.

Chapter 3

Attacking A5/1

One of the results from this thesis is a set of rainbow tables that can be used to decrypt a GSM conversation. This chapter will provide the necessary background theory and give a detailed explanation of how the rainbow table attack against A5/1 works.

3.1 Related Work

Related work aimed at attacking A5/1 has been carried out for the last two decades. However, a practical attack did not arise until 2003, when Barkan et al. published their findings. They used a passive ciphertext-only time-memory trade-off attack requiring a large amount of precomputed tables [17]. This was the first real attack not requiring large amounts of known plaintext. It is uncertain whether tables were generated. Regardless, tables were never publicly released.

In 2007, two groups headed by Prof. C. Paar (Ruhr University of Bochum) and Prof. M. Schimmler (Christian-Albrechts University Kiel) created a Cost-Optimized Parallel Code Breaker (COPACOBANA), an FPGA-based machine that could be used to create tables employing the same principles as described in Barkan et al. The FPGA solution was made commercially available[2], but no tables were released.

In 2008, the The Hackers Choice (THC), represented by David Hulton and Steve Muller, reiterated the 2003 attack with 68 FPGA boards[49]. This group also had a sister project named AirProbe[1] that developed software to capture and decode GSM signaling traffic. Rainbow tables were claimed to have been computed, but were never released. They made the assumption that the key size used in GSM is only 54 bits. Although this was the case in several countries some years ago, it is not the case as of per today. This means that even if these tables were to exist, they would not be useful in the process of attacking A5/1.

3.2 A5/1

None of the security algorithms implemented in GSM were originally made available to the public, but some were later reverse engineered and leaked [25]. In this section, the cryptographic algorithm used in GSM, A5/1, is described in detail.

Ciphering in GSM is performed using the stream cipher A5/1, which consist of three linear feedback shift registers (LFSR). They are called *R1*, *R2* and *R3*, and are of length 19, 22 and 23, respectively. In an LFSR, all bits are shifted one place towards the end[1] of the register every time the register is clocked. This leaves the least significant bit (lsb) empty, meaning that this bit needs to get its new value in a different way. This is solved by letting each register have a few tapped bits. As seen in figure 3.1, the tapped bits of *R1* are bit *13*, *16*,

[1] To the left in this case

17 and *18*, of *R2* bit *20* and *21*, and of *R3* bit *7*, *20*, *21* and *22*. Whenever a register is clocked the value of these taps are read out before the shifting has been performed. Their values are then XORed together and the result used as input to bit 0. Whether a register is clocked or not is decided by the majority

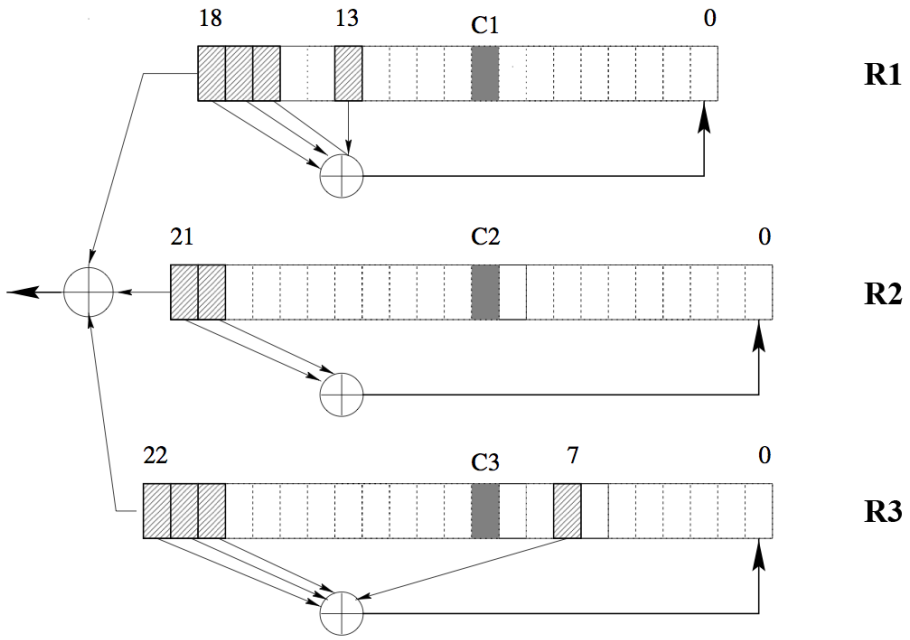

Figure 3.1: The A5/1 stream cipher used for encryption in GSM. Modified from [20]

function. Three clocking bits exist, denoted by *C1*, *C2* and *C3* respectively. *C1* is bit *8* of *R1*, *C2* is bit *10* of *R2* and *C3* is bit *10* of *R3*. At each clock cycle the value of these bits are analyzed, and the two or three registers whose bit agrees with the majority are clocked.

The A5/1 algorithm takes as input a 64-bit key K_c and a 22 bit frame number F_n and produces 228 bits of keystream. The output is the result of bit *18*, *21* and *22* XORed together. Messages between the MS and the BTS are sent in

bursts containing 114-bits of payload, and since the same frame number can be used two times in a row[2] A5/1 needs to produce 228 bits of keystream. The first 114 bits of the keystream are used to encrypt downlink traffic (BTS to MS), while the last 114 bits are reserved for encryption of uplink traffic (MS to BTS).

From initialization to 228-bits of keystream, the A5/1 algorithm goes as follows:

1. All three registers are set to 0. Then, for 64 cycles, the key is mixed into the registers in parallel using the following algorithm:

   for $i = 0$ to 63 **do**
   $R1[0] = R1[0] \otimes K_c[i]$
   $R2[0] = R2[0] \otimes K_c[i]$
   $R3[0] = R3[0] \otimes K_c[i]$
   Clock all three registers according to the regular clocking scheme.
   end for

   Where $Ri[0]$ denotes the lsb of register Ri, $K_c[0]$ the lsb in the key and $K_c[63]$ the most significant bit. It's important to notice that the majority clocking mechanism is not used at this stage.

2. 22 additional cycles are clocked, still overlooking the majority function. During this period the frame number is XORed into the lsb of the registers in the same way as with the key, that is:

   for $i = 0$ to 63 **do**
   $R1[0] = R1[0] \otimes F_n[i]$
   $R2[0] = R2[0] \otimes F_n[i]$
   $R3[0] = R3[0] \otimes F_n[i]$
   Clock all three registers according to the regular clocking scheme.
   end for

3. 100 additional clocks are performed with the the regular majority clocking mechanism activated, but the output is discarded. The content of the

[2]Once for both downlink and uplink.

registers at the end of this state is what we refer to as the *initial state* of A5/1.

4. 228 clocks are performed to produce 228 bits of keystream. This is keystream is then XORed with the plaintext in order to encrypt the data.

3.3 A Cryptanalytic Time-Memory Trade-Off

A5/1 can be attacked using a cryptanalytic time-memory trade-off. The idea of such a time-memory trade-off (TMTO) comes from Hellman, who in 1980 introduced a TMTO-attack against block ciphers that with high probability can recover a N key cryptosystem in $N^{2/3}$ operations using $N^{2/3}$ words of memory [48]. Since then, several improvements, analysis' and optimizations concerning TMTO-attacks has been released. Going into the details of these would be a very extensive task and is also without the scope of this thesis. Instead, this section will focus on providing the relevant background theory needed in order to understand the time-memory trade-off attack against A5/1.

3.3.1 Hellman's Time-Memory Trade-Off

There are two naive approaches on how to break a block cipher; exhaustive search and table lookup. The exhaustive search technique is a known-plaintext attack, where the ciphertext gets deciphered with each possible key and then compared with the known plaintext. This gives a time complexity of $T = O(N)$ and a memory requirement of $M = 1$.

The approach taken in a table lookup attack is to encrypt some fixed plaintext with each of the N possible keys in order to produce N distinct ciphertexts. These ciphertexts are then sorted and stored together with their keys in a table

of size $M = N$. The pre-computation cost of such an attack is N operations and the time complexity for doing lookup is $T = O(1)$.

The problem with these two methods is that the attack time is either to long, i.e $T = O(N)$ in the case of exhaustive search, or that they require a large amount of memory, i.e $M = N$ in the case of table lookup. As an example, consider the amount of memory needed in order to create a lookup table for a 64-bit key:

$$M = 2 \times 64 \times 2^{64}\ bits = \frac{128 \times 2^{64}}{2^3 \times 2^{40}}\ Terabytes = 268\ 435\ 456\ Terabytes$$

A number that leaves such an attack impossible in practice. It is easy to do a comparison with Hellman's time-memory trade-off, which only needs $N^{2/3}$ words of memory. In the same case of a 64-bit key, the memory requirement would be:

$$M = 2 \times 64 \times \left(2^{64}\right)^{2/3}\ bits = \frac{128 \times \left(2^{64}\right)^{2/3}}{2^3 \times 2^{40}}\ Terabytes \approx 102\ Terabytes$$

Although still a large memory requirement, it is well within the limits of what is practically feasible. With the price for a 2 TB hard drive being around \$130[3], obtaining 102 Terabytes of storage would cost around $\$130 \times \frac{102}{2} = \$6630$. That's a price that must be considered to be a small investment for larger organizations.

Hellman introduced his time-memory trade-off for block ciphers in 1980. Given a fixed plaintext block P_0, this can be encrypted with the cipher S. The ciphertext C is thus given by:

$$C = S_k(P_0)$$

The attack works by creating chains consisting of key-ciphertext pairs. The chain starts with a key value k_i. This key is used to encipher P_0 in order to

[3]As of per June 2010

get $C_i = S_{k_i}(P_0)$. The ciphertext C_i can then be converted into a key k_{i+1} by applying a *reduction function*[4] R on C_i. Chains of alternating keys and ciphertexts can thus be created by successively applying the cipher S and the reduction function R:

$$k_i \xrightarrow{S_{k_i}(P_0)} C_i \xrightarrow{R(C_i)} k_{i+1} \xrightarrow{S_{k_{i+1}}(P_0)} C_{i+1} \xrightarrow{R(C_{i+1})} k_{i+2} \longrightarrow \cdots$$

$R(S_k(P_0))$ is the sequence of operations that generates a key from a key. It is called $f(k) = R(S_k(P_0))$ and leads to a chain of keys:

$$k_i \xrightarrow{f} k_{i+1} \xrightarrow{f} k_{i+2} \xrightarrow{f} \cdots$$

As can be seen in figure 3.2, a table consist of m chains of length t. However,

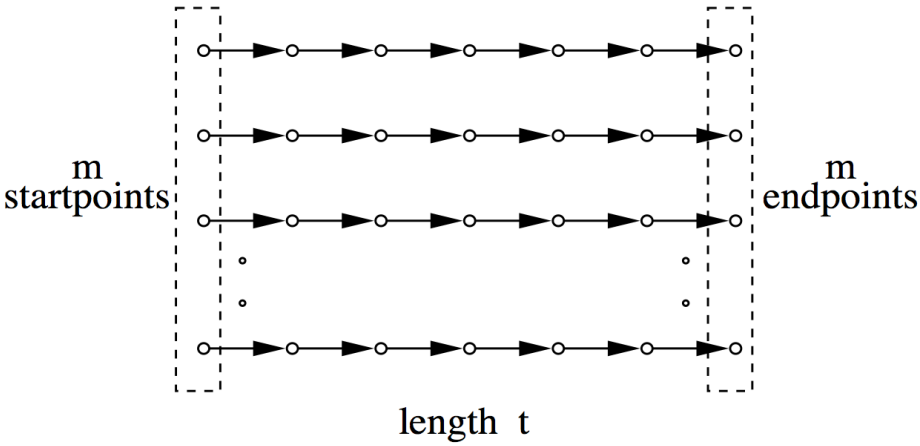

Figure 3.2: Illustration of a single Hellman table of size $m \times t$[19]

only the first and last element in each chain are stored in order to reduce the memory requirements. Given a ciphertext C, the aim is to find out whether the key used to generate C is in the table. The first step is to apply R to C in order to obtain a key Y_1. This key is compared to the endpoints of the table, and

[4]The cipher text is longer than the key, hence the reduction

if a match is found it means that the key used to encipher C might be found in the next to last column of the corresponding chain. The chain can then be reconstructed from the startpoint in order to possibly find this key. If there is no matching endpoint, the function f is applied until a matching endpoint is found, or until it has been applied a maximum of $t-1$ times. A match with one of the endpoints in the table does not necessarily mean that the key can be found in that chain. This is due to so-called false alarms. A detailed description of false alarms will be given later in this subsection.

One of the drawbacks of Hellman's time-memory trade off is that chains starting at different keys may collide and merge. This is a consequence of the fact that R only provides an arbitrary reduction from the space of ciphertexts into the space of keys. Merges lead to less efficient tables, since there will be keys that are covered by more than one chain. The chance of finding a key by using a table of m rows of t keys is given by[48]:

$$P_{table} \geq \frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1} \tag{3.1}$$

As can be seen, the efficiency of a single table decreases rapidly with its size. In order to obtain a high probability of success it is therefore better to generate multiple tables, where a different reduction function is used for each table. There can be collisions between chains of different tables when using more than one table, but they will not merge since different reduction functions are used in different tables. Using l tables, the probability of success is given by[48]:

$$P_{success} \geq 1 - \left(\frac{1}{N} \sum_{i=1}^{m} \sum_{j=0}^{t-1} \left(1 - \frac{it}{N}\right)^{j+1}\right)^{l} \tag{3.2}$$

False alarms

As mentioned earlier, finding a matching endpoint in the table does not

necessarily mean that the key is in the table. False alarms are undesirable, since there needs to be computing time spent in searching through a chain looking for a key, only not to find it.

Let s be the length of the generated chain at the time when a matching endpoint is found, and let this generated chain be denoted by z:

$$C \xrightarrow{R} Y_1 \xrightarrow{f} Y_2 \xrightarrow{f} \cdots \xrightarrow{f} Y_s$$

Given an endpoint E_j that matches Y_s, this chain is then calculated from the startpoint S_j. If Y_1 is not found in this chain it was only a false alarm. This is the result of chain z merging with the chain where we found a false alarm, somewhere later than the column where Y_1 is. Figure 3.3 shows an example of

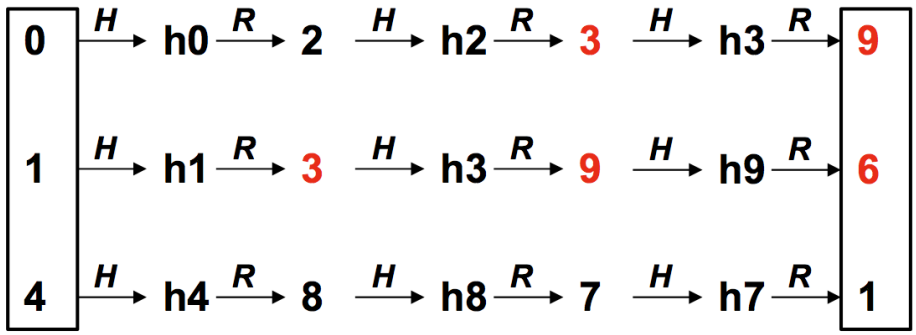

Figure 3.3: Hellman table for hash values showing the occurrence of a false alarm. h_1 is the hash value that we're looking for the password for, and the false alarm happens at the point where h_1 is transformed into the password value 9. Modified from [56]

a false alarm in the case of a Hellman table for hash values. In this example h_1 is the given hash value that we're looking for the corresponding password to, in this case 1. If the reduction function R and the hash function H is applied successively on h_1, then the result after a couple of alternations will be the value 9. Since this is also an endpoint in the table, a false alarm will now occur. Note

that even if the second row had not been a chain in the table, we would still get a false alarm. In other words, false alarms exist as a result of merges with chains both inside and outside the table.

3.3.2 Rivest's Distinguished Point Method

In 1982 Rivest came with the suggestion of using distinguished points as endpoints for the generated chains[29]. A distinguished point is an endpoint that satisfies some easily tested syntactic property, e.g the first ten bits of a key are zero. By letting all endpoints being distinguished points, the number of data lookups needed gets drastically reduced. Given a ciphertext C, a chain of keys is generated until a distinguished point is found. Only then is it looked up in the memory in order to find what chain the key was found in, and what the start point of that chain is.

In [22] Borst et Al. pointed out that one of the advantages with distinguished points is that merges can easily be detected. This is due to the fact that two merging chains will end up having the same endpoint; the next one after the merge. Since the endpoints have to be sorted anyway, the merges can be discovered and removed without any additional cost. This means that merge-free tables can be created simply by removing a chain that merges and replace it with a new one instead.

Another advantage noted by Borst et Al. in [22] is that distinguished points can be used to discover loops. If a distinguished point is not found after a large number of keys[5] has been generated, it is generally a good indication that the chain contain a loop. It is thus discarded in order to obtain loop-free tables.

Unfortunately, using distinguished points also comes with a backside; the variation in chain length. Short chains will lead to an increased storage

[5]Intuitively, this number should of course be larger than the average occurrence of a distinguished point

requirement, whereas long chains will increase lookup time, give more merges, and thus also generate more false alarms. As noted earlier, merges can easily be removed and be replaced by new chains. However, removing a chain does not remove the false alarms, since a false alarm is not dependent upon the chain being part of the table.

3.3.3 Oechslin's Rainbow Tables

The concept of Rainbow Tables comes from Philippe Oechslin who published the idea in his article *Making a Faster Cryptanalytic Time-Memory Trade-Off* [55]. Oechslin's rainbow tables provide a solution to the main limitation of Hellman's time-memory trade-off, the fact that two chains that collide within a single table will merge. With rainbow tables there can still be collisions within the same table, but the chains will not necessarily merge together.

The difference between Hellman's tables and rainbow tables lies in the way the reduction function is used. Whereas the original tables uses the same reduction function for all the $m \times t$ chains within a single table, rainbow tables uses a new reduction function for every point in the chain. A chain of length t starts with the reduction function R_1 and ends with the reduction function R_{t-1}. The big advantage with this approach is that if two chains collide within a table, they will only merge if the collision appears at the same position in both chains. This is due to the fact that both chains will continue with different reduction functions after the collision. It also follows from this that if a collision occurs in a chain of length t, the chance of the two chains merging is only $\frac{1}{t}$.

An illustration of the difference in structure between Hellman's original tables and Oechslin's rainbow tables can be seen in figure 3.4. The figure shows t classical tables of size $m \times t$ on the left, and a single rainbow table of size $mt \times t$ on the right. When doing lookup in rainbow tables, the first step is to check whether key is in the next to last column of the table. This is done by applying

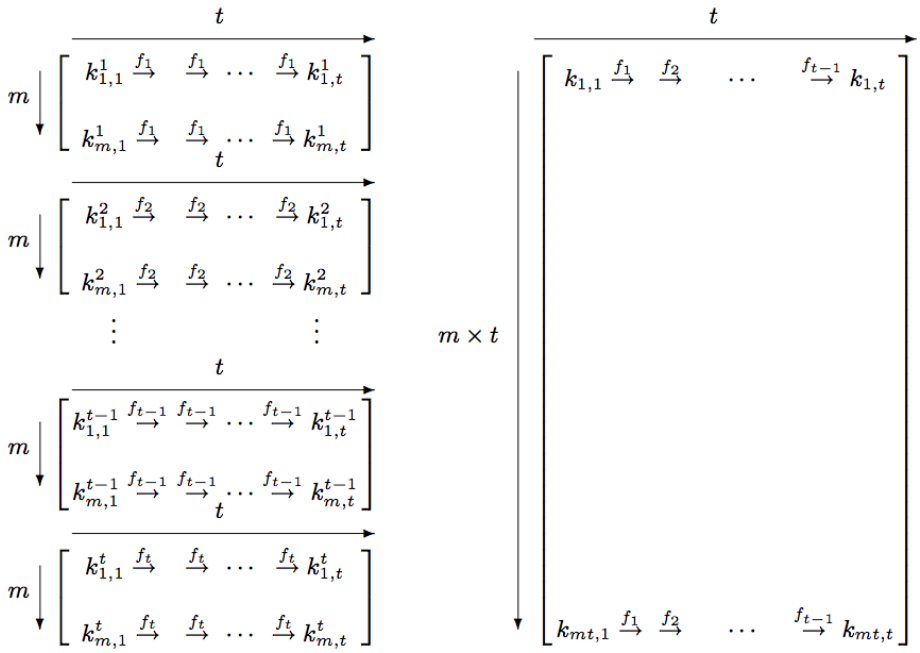

Figure 3.4: Illustration showing the difference in structure between Hellman's original tables and Oechslin's rainbow tables. t classic tables of size $m \times t$ is seen on the left, whereas one rainbow table of size $mt \times t$ can be seen on the right. [55]

R_{t-1} to the ciphertext and then comparing the result with the endpoints of the table. If there is a match, the stored start point can be used to reconstruct the chain and find the key that was used to produce the ciphertext. If the endpoint is not found, the second to last column is checked by applying R_{t-2}, f_{t-1} to the ciphertext. If it is not there, R_{t-3}, f_{t-2}, f_{t-1} is applied, and so forth until the whole chain hs been checked. The total number of calculations needed to be performed during lookup is thus $\frac{t(t-1)}{2}$ [55]. This is half as much as with Hellman's tables, that uses t^2 calculations to search t tables of size $m \times t$.

The probability of success within a single rainbow table of size $m \times t$ is given

by[55]:

$$P_{table} = 1 - \prod_{i=1}^{t} \left(1 - \frac{m_i}{N}\right) \tag{3.3}$$

where $m_1 = m$ and $m_{n+1} = N\left(1 - e^{-\frac{m_n}{N}}\right)$

According to Oechslin [55], the success probability of rainbow tables can be directly compared to that of classical tables. In fact, the success probability of t classical tables of size $m \times t$ is approximately equal to that of a single rainbow table of size $mt \times t$, since in both cases mt^2 keys are covered with t different reduction functions. It is also interesting to note that a collision within a set of mt keys (a single classical table or a column in a rainbow table) results in a merge, whereas collisions with any of the remaining keys does not lead to a merge. The probability of success are compared in figure 3.5. Note that one axis have been relabeled from t to l in order to fit with equation 3.2. As can be seen, rainbow tables seem to have a slightly better probability of success compared to classical tables. However, this might very well just be that the success rate of Hellman's tables are lower bound, whereas the success probability for rainbow tables is the exact expectation [55].

When compared to Hellman's tables with distinguished point it becomes apparent that rainbow tables share some of the same advantages as distinguished points, but without suffering from their limitations. According to Oechslin, these are:

- Compared to Hellman's original method, table look-ups are reduced by a factor of t. However, since disk access is drastically reduced when using distinguished points, rainbow tables are still expected to be outperformed by distinguished points due to the large expenses connected to disk access.

- A merge in a rainbow table will lead to two identical endpoints, meaning that merges are just as easily detectable as with distinguished points. This

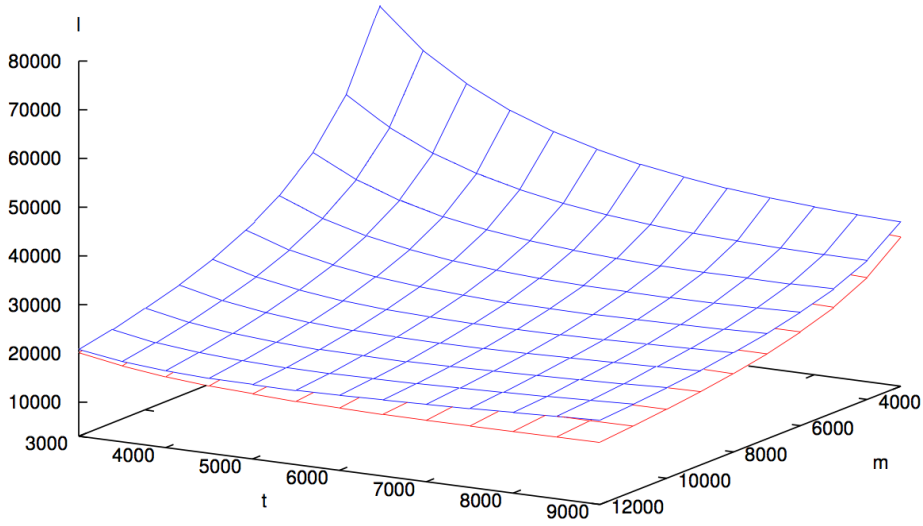

Figure 3.5: Comparison of the success rate of Hellman's tables and rainbow tables. The upper surface represents the constraint of 99.9% success with classical tables, wheras the lower surface is the same constraint, but for rainbow tables instead [55].

also means that Rainbow tables can be generated free of merges.

- Since there is a new reduction function for every column in a rainbow table, no loops can exist. This is better than the loop detection used in distinguished points, since there's no need to spend time generating loops that will be rejected anyway.

- Rainbow chains have constant length, thus avoiding the negative effects of variable chain length that distinguished points suffers from. This means that there are fewer occurrences of merges and false alarms in rainbow tables.

3.3.4 A Cryptanalytic Time/Memory/Data Trade-off for Stream Ciphers

There are two major types of symmetric-key cryptosystems; block ciphers and stream ciphers. Block ciphers take a number of bytes and encrypt them as a single unit, whereas stream ciphers encrypts one bit at a time. Furthermore, encryption in block ciphers is carried out by mixing plaintext with the key in an invertible way, whereas stream ciphers use the key to produce a keystream that is XORed with the plaintext.

Hellman and Oechslin's trade-off's were both directed towards block ciphers. Since a block cipher takes the plaintext and key as input and produce ciphertext as output, each ciphertext corresponds to a particular plaintext for block ciphers. A pre-computed table for a block cipher can thus only be used for one particular known plaintext.

Stream ciphers on the other hand have a very different behavior when it comes to time-memory trade-off attacks. It uses its internal state as input and produce a keystream as output. This means that we have state-keystream pairs that are independent from any particular plaintext, and it is thus possible to create pre-computed tables that can be used independently of the plaintext.

The first time-memory trade-off for stream ciphers was described independently by Babbage[16] and Golic[46]. They connect each of the N possible states of the generator with the first $log_2(N)$ keystream bits produced by the stream cipher from those states. This mapping can be described as:

$$f(x) = y \tag{3.4}$$

where x is the internal states of the stream cipher and y the keystream generated from x. The attack works by picking M random x_i states, computing their corresponding keystream, y_i, and then store all of the (x_i, y_i) pairs on a disk sorted on increasing order of y_i. Only M out of N states are covered, but coverage

can be improved by having more known data points D. Since lookup is done on $log_2(N)$ bits of keystream, D data points give us a total of $D - log_2(N) + 1$ samples of keystream. Thus, lookup can be done on all $D-log_2(N)+1$ keystream samples in the table. This increases the probability of a hit by $\frac{1}{D-log_2(N)+1}$. If a match is found, the rest of the plaintext can be derived by running the keystream generator from that point on.

A general attack that combines the work of both Hellman, Babbage and Golic was proposed by Biryukov et Al. in the article *Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers*[19]. Their idea is to succesively apply $f(x) = y$ from 3.4 and a function that maps the keystream into the state space again. This forms the basis for the attack used in the *A5/1 Security Project* that will be explained in detail in section section 3.5.

It is interesting to note that distinguished points fits especially well together with time-memory trade-offs that have more than one data point. This has to do with the fact that a disk access is an extremely expensive operation, and should be avoided if possible. Thus, when looking up more than one keystream sample the profit becomes even more apparent.

3.4 A5/1's Reduced Keystream Space

As explained in section 3.2, A5/1 gets initialized by mixing in a 64-bit key. In order for A5/1 to be cryptographically strong, it's important that this key is utilized in a way that provides a keystream space of almost equal size. Unfortunately, it turns out that this is not the case with A5/1, and the keystream space gets reduced to only 16% of the original size after the 100 clockings[10]. This section will provide the details around the reduction in A5/1's keystream space.

Clocking back A5/1

In order to understand how the keystream space gets reduced in A5/1, a look at the theory around backclocking is first needed.

In order to decrypt an entire conversation it is important to know either the key or the state of A5/1 just after the key has been mixed in. Both of these are of equal value, since the keystream for every frame in a conversation can be determined directly from both. In the process of obtaining this state, some backclocking is needed.[6]

Clocking back one single register is a trivial task. When clocked forward, all the tapped bits are XORed together and the result used as input to the rightmost bit (lsb). Backward clocking on the other hand, is performed by taking all the tapped bits except the leftmost (msb), and then XORing these together with the rightmost bit. After this, all registers are shifted one step to the right, and the result from the XOR operation is stored in the leftmost bit position. It is when clocking back the entire A5/1 machine that things get more complicated, since majority clocking needs to be taken into account. However, it is also when looking closer at this process that the reason why the A5/1 algorithm only utilizes 16% of the key space gets revealed.

By looking at the clockbits of each LFSR, and the bit to the left of it, it becomes apparent that there are in total $2^6 = 64$ combinations[7] of the values that these bits can have.

Let $S(t) = (S_1(t), S_2(t), S_3(t))$ denote the whole internal state of A5/1 at time $t \geq 0$, where $S(0)$ is the internal state of A5/1 just after the frame number has been mixed in. At the same time, let $C(t)$ denote the clock-control sequence at time $t \geq 0$, such that if $C(t) = \{1, 2\}$, then registers 1 and 2 are to be clocked, if $C(t) = \{1, 2, 3\}$, then registers 1,2 and 3 is clocked, and so on. Let (i,j,k) denote

[6] It is possible to generate rainbow tables that can obtain this state without doing backclocking, but such tables would have reduced efficiency.

[7] 6 bits that each can take two values, 0 or 1

a permutation of (1,2,3). Then the following six events can occur[46]:

A: $for\ any\ k,\ if\ s'_i = s'_j \neq s'_k = s_k,\ then\ C(t-1) = \{i, j\}$

B: $for\ any\ k,\ if\ s'_i = s'_j \neq s'_k \neq s_k,\ then\ C(t-1)$ *can take no values*

C: $if\ s'_1 = s'_2 = s'_3 = s_1 = s_2 = s_3,\ then\ C(t-1) = \{1, 2, 3\}$

D: $if\ s'_1 = s'_2 = s'_3 \neq s_1 = s_2 = s_3,\ then\ C(t-1)$ *can take every of the four values* $\{1, 2\},\ \{1, 3\},\ \{2, 3\}$ *and* $\{1, 2, 3\}$

E: $for\ any\ k,\ if\ s'_1 = s'_2 = s'_3 = s_i = s_j \neq s_k,\ then\ C(t-1)$ *can take every of the two values* $\{i, j\}$ *and* $\{1, 2, 3\}$

F: $for\ any\ i,\ if\ s'_1 = s'_2 = s'_3 = s_i \neq s_j = s_k,\ then\ C(t-1)$ *can take every of the three values* $\{i, j\},\ \{i, k\}$ *and* $\{1, 2, 3\}$

Further on, if an internal state $S(t)$ is randomly chosen according to uniform distribution, then the number of solutions for $S(t-1)$ is a nonnegative integer random variable Z with the probability distribution[46]:

$$P(Z = 0) = \frac{24}{64}, \quad P(Z = 1) = \frac{26}{64}, \quad Pt(Z = 2) = \frac{6}{64},$$

$$P(Z = 3) = \frac{6}{64}, \quad P(Z = 4) = \frac{2}{64}$$

As can be seen, 24 of 64 states cannot be clocked 1 step back. This means that there is no way that such a state can exist, and it is therefore called an illegal state. A conclusion that follows immediately from this is that the keystream space is reduced to $1 - \frac{24}{64} = 0.625 = 62.5\%$ of the original key space. An example of such a state can be seen in figure 3.6.

When looking at the bits to the left of the clocking bits, it becomes apparent that the registers clocked in the previous round was registers R1 and R3, since

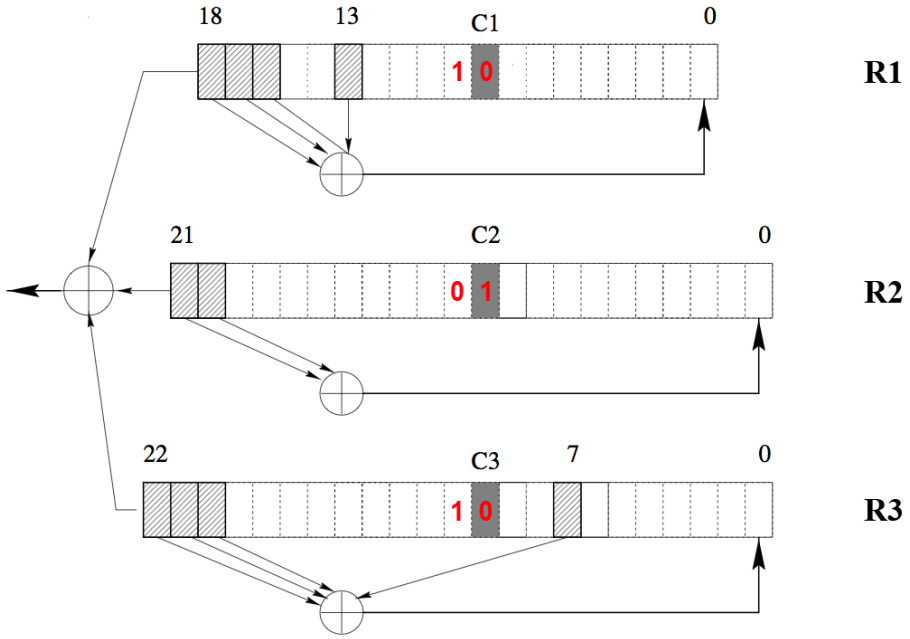

Figure 3.6: Illustration of an illegal A5/1 state, $S(t)$, that can't be clocked back

they make up the majority function. By clocking R1 and R3 one step back the result is the state seen in figure 3.7.

It is easy to see that this state is not a valid one, since R2 now suddenly is a part of the majority clocking function, a contradiction to the backclocking step that was just performed. One would therefore end up with a different state if an attempt to clock it forward from $S(t-1)$ to $S(t)$ was made.

The fact that 24 of 64 states cannot be clocked back 1 step was pointed out by Golic in the paper *Cryptanalysis of Alleged A5 Stream Cipher*[46], but strangely enough Golic fails to see the correlation between this and the reduction in keystream space that follows from this. However, a person under the pseudonym

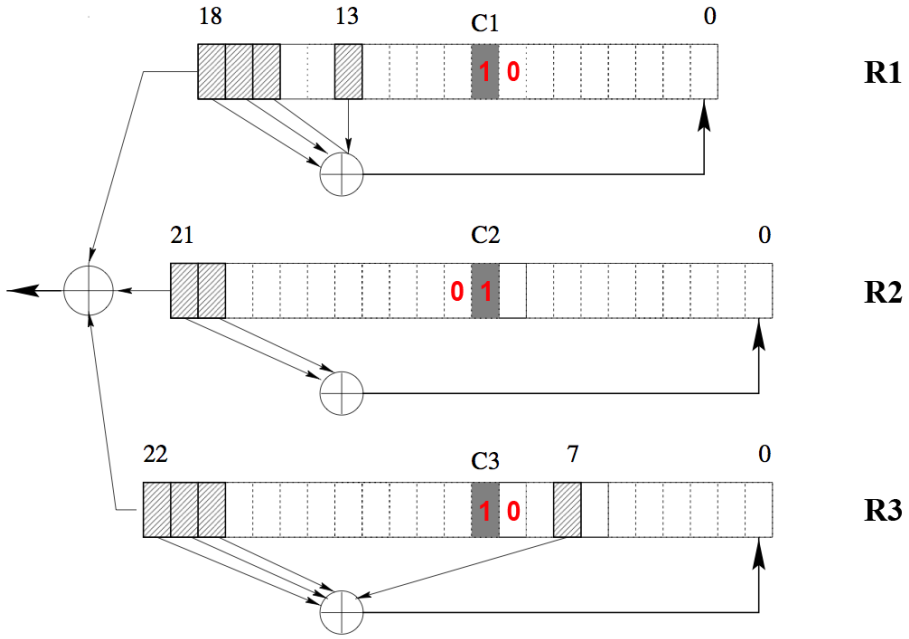

Figure 3.7: Illustration showing an A5/1 state clocked back from $S(t)$ to $S(t-1)$

M vd S discovered this[8] and posted his findings on the A5/1 Security Project mailing list[67]. M vd S took 1 million random states and tried clocking them back 100 times. This was done several times in order to check that the numbers were stable. His discovery was that only 14% of the states can be clocked back 100 times. This means that 86% of the states have no ancestors and are therefore illegal states. It also follows from this that the keystream space of A5/1 is only 14% of the original 2^{64}, since 100 steps of majority clocking is performed before A5/1 starts producing the keystream. His findings are summarized in table 3.1 An illustration of what happens can be seen in figure 3.8.

[8] As far as we know, this has not been published before

Number of backclockings	Amount of illegal states
16 back	57% illegal states
32 back	68% illegal states
48 back	74% illegal states
64 back	78% illegal states
80 back	82% illegal states
96 back	84% illegal states
100 back	86% illegal states
150 back	89% illegal states

Table 3.1: Illegal states

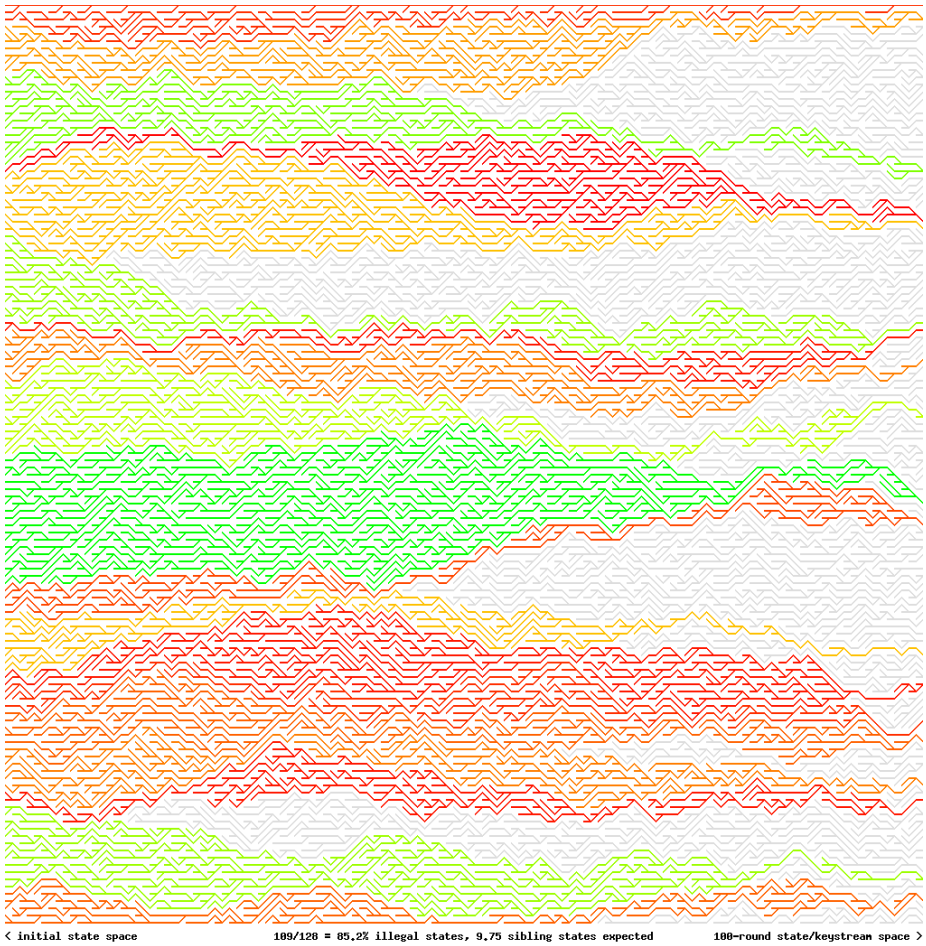

Figure 3.8: State space convergence in A5/1 during 100 clock cycles. Grey paths are not accessible trough forward clocking, green paths have many ancestors states leading to the same keystream, whereas red paths have few ancestors leading to the same keystream

As can be seen, there is a lot of initial states converging and ending up in the same state. It is also illustrated how 86% of the states are illegal[9] and thus end up in a dead end when clocking back.

It becomes apparent when looking at this picture, that a state can have several ancestors. After having clocked back a certain amount of clockings, in general we will be between 1 and 100 sibling states[10]. Figure 3.9 shows the recorded frequency of the number of sibling states in a sample set of one million random states. The blue dots is the distribution of sibling states when first clocking 100 forward and then 100 backwards, while the red dots refer to the distribution when backward clocking 100 times without having done forward clocking first. In the last case, the data point showing that 848000 of 1 million states have no ancestors has been removed.

When taking a uniform distribution of 2^{64}, one would expect the number of ancestors to be $\frac{1}{0.14} = 7.1$. Although figure 3.9 shows a peak at around 7, the weighed average for forward-backward clocking is 13.04 and for backward clocking 1.00. An average of 13.04 seems to indicate that the initial state space converges to $\frac{1}{13.04} = 0.077 = 7.7\%$ of the initial space, but this is not the case. Figure 3.10 illustrates what happens.

As seen, the red dotted line indicates that half the keystreams find their origin in 82% of the inintial states, while the other half find their origin in only 18% of the initial states. In addition, the blue dotted line shows that 10% of the actual keystreams find their origin in 30% of the initial states. In other words, we see that A5/1 seems to have a preferrence towards a smaller group of favoured states. This means that there is a even bigger chance of getting a hit in the table, since 82% of the initial states is covered by only 50% of the keystream.

[9]Here denoted by grey

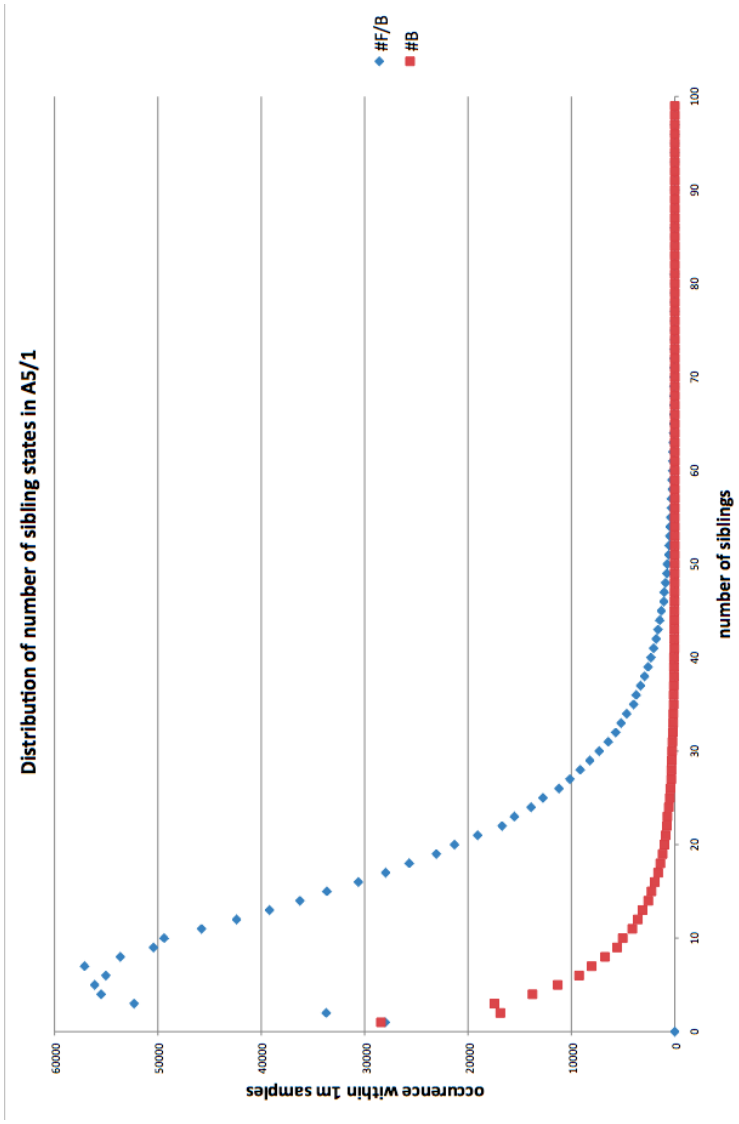

Figure 3.9: Illustration showing the distribution of siblings after having clocked back a state 100 times

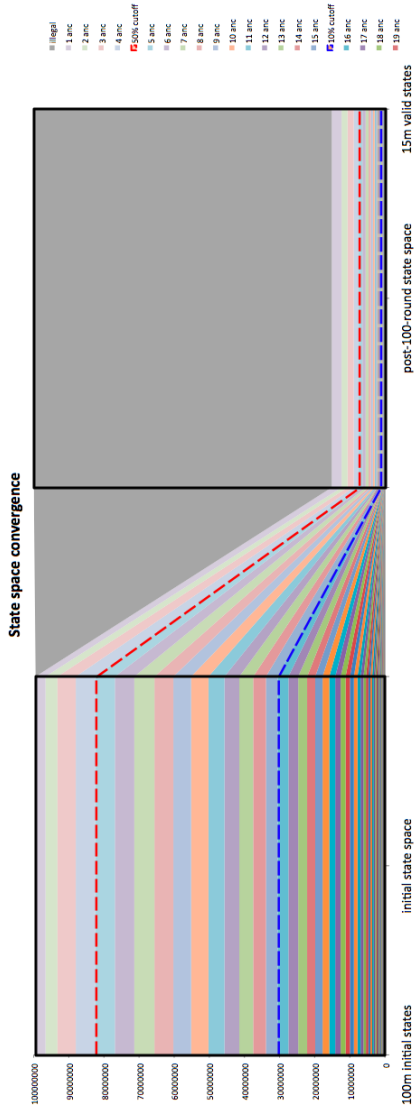

Figure 3.10: Illustration showing how the state space in A5/1 convergences towards a few preferred states

3.5 The A5/1 Security Project

In August 2009, the A5/1 Security Project was launched by Karsten Nohl and others as a reimplementation of the 2008 work by THC. The project aimed to distribute the table generation in a way that anyone could volunteer and start creating tables. Table generating scripts for NVIDIA-devices supporting CUDA computing[10] were made available on the project's website[10] and finished tables were shared via BitTorrent. The goal was to compute and share 2 Terabyte of rainbow tables, but as of per June 2010 only 10% of the targeted goal is being shared [12]. In addition, these tables are suboptimal since they are of an early stage table format that don't take A5/1's reduced keystream space into consideration.

However, since the start of the project there has been some major changes. The most significant one was the development of code based on ATI-graphic cards instead of NVIDIA. This work was done by Frank A. Stevenson[11] on his own initiative, and the first piece of code was posted on the A5/1 mailing list in October 2009[64]. Another significant event was M vd S' post on the mailing list in January 2010[67] where A5/1's reduced keystream space was pointed out, as explained in section 3.4. This discovery lead to a new table format that has been implemented by Frank Stevenson to work with ATI cards. The new table structure is not yet implemented for NVIDIA cards, and the code has been down for a few months now due to a bug in the code.

This section will provide the details around the format of the tables generated by the ATI-code.

[10] http://www.nvidia.com/object/cuda_home_new.html
[11] http://en.wikipedia.org/wiki/Frank_A._Stevenson

3.5.1 Table Structure

The current tables are a combination of Hellman tables with distinguished points and rainbow tables. All the chains within one single table is generated in the same way. The only thing that separates them is that they all have different start points.

The generation of a single chain starts by taking a 64-bit random value and putting it into the registers of A5/1. A5/1 is then first clocked 100 steps forward with the output bits discarded, before another 64 clockings is performed were the output is stored as a keystream. As explained in section 3.4, the reason for doing 100 clocking first is to avoid generating tables that contain keystreams that would never exist after the 100 initial clockings in A5/1. In order to map this keystream from the reduced keystream space into the full space of initial states, a transformation is needed. This transformation is performed by XORing the keystream with a 64-bit round function that fetches its value from the output of an LFSR. This LFSR gets initialized with the same value every time[12], but the thing that separates the round functions from each other is that they all use different parts of the LFSR-output as their value. The LSFR has an *advance* function, and 64-bits of output is generated every time this gets called. Every table takes a number, *id*, as input, calls advance *id* times and uses the bits produced during the last call as the first round function. You can thus avoid having the same round function ever being used again by not using any two *id*s less than 8 numbers apart as input to a table.

Each table uses 8 round functions, and the round function is changed whenever a distinguished point is found. The indication chosen as a distinguished point is that the last 12 bits of the keystream is to be zero, and whenever this occurs the round function gets changed by calling advance once. As mentioned in subsection 3.3.2, distinguished points can also be used to detect loops. A chain is therefore dropped if a distinguished point has not been detected before 3

[12]That is for every round, in every chain, in every table

million A5/1 steps have been produced.

A total of 8,662,000,000 ($8,662 \times 10^9$) chains are produced for every table, where only the startpoint and endpoint is stored for every chain. The chains are then sorted incrementally on their end value, while at the same time removing any merging chains.

3.5.2 Table Lookup

In order to do a table lookup, minimum 64-bits of known keystream is needed. Given that we are already in possession of the ciphertext, knowing the keystream is equivalent to knowing the plaintext. This is due to the fact that keystream K can be deduced by simply XORing the plaintext P and the ciphertext C together:

$$C \otimes P = (P \otimes K) \otimes P = (P \otimes P) \otimes K = K$$

Given a 64-bit keystream sample K, doing lookup is quite an easy task. First step is to check whether K can be found within one of the eight round intervals in a table. To do this, 8 different chains are computed from K, were each chain starts with one of the 8 round functions used by the table we are doing lookup in and ends with the last round function. This produces 8 different chains, whose endpoints needs to be compared to that of the tables. If there is a match between two endpoints, the whole chain is generated from the start point until K is found again. Unless an false alarm, the value that appears just before K is the initial state of A5/1 100 clockings before the start of keystream K.

We are interested in knowing the state of A5/1 just after the session key K_c has been mixed in, but before the frame number is mixed in. By knowing this one would be able to produce the keystream for every frame in an entire conversation, since the frame number is already publicly known. As explained in section 3.2, A5/1 produces 228 bits of keystream. Depending upon where

in those 228 bits our keystream sample is taken from, a different amount of backclocking is needed to be performed in order to find the desired state. Let K_i denote the keystream taken from bit position i and outwards, where $i = 0$ is at the start of the keystream generation. It is then needed to clock back $i + 100 + 22$ steps in order to find the state just after K_c has been mixed in. As an example, consider the case of a 64-bit keystream sample from bit position 10 to 73 in the 228 output bits from A5/1. $10 + 100 + 22 = 132 steps$ would need to be clocked back in order to receive at the desired state. As explained in section 3.4, this will lead to some 13 different candidate states. These are all states that produces the same keystream for that particular frame, but only one of them is the correct state that can be used to decrypt the rest of the conversation. It is thus needed to find the correct candidate key if we want to decrypt an entire conversation, but this is a trivial task since it only needs to be tested on another frame.

As will be discussed in chapter 8, several frames exists that may contain minimum 64-bits of known plaintext, and thus gives us the needed keystream sample. In fact, there are several candidates that are likely to contain even more known keystream. Given $n > 64$ bits of keystream, the technique from Babbage[16] and Golic[46] described in subsection 3.3.4 can be applied in order to increase the probability of success. This would give $n - 64 + 1$ different keystream samples that can be used to do lookup with. This means that the tables only need to $\frac{1}{n-64+1}$ the size and still give the same coverage.

3.5.3 Expected coverage

Doing a mathematical analysis on the expected coverage of the tables is no trivial task, since it's difficult to estimate the number of merges. There is however some general observations that that should be noted.

As pointed out in section 3.4, the keystream space shrinks to only 14% after 100

clockings have been performed. This means that the effective keystream space of A5/1 is no bigger than

$$Keystream\ space = 2^{64} \times 0.14 = 2^{61.16} \tag{3.5}$$

In addition, since 51 keystream samples can be used when doing lookup, only $\frac{1}{51}$ of the keystream space needs to be covered. This means that the tables only need to cover:

$$Cover\ criteria = \frac{2^{64} \times 0.14}{51} = 2^{55,49} \tag{3.6}$$

However, it is when it comes to computing the coverage of the table, things get more complex. Giving an estimate on the number of merges is complex task, since it increases with the table size. In addition, the fact that A5/1 has preferred states will also have an affect on the number of merges. A closer analysis of this is something that could be a part of a future work that aims at improving the table structure.

Chapter 4

Rainbow Table Generation and Lookup

As mentioned in section 3.5, only around 10% of the targeted goal of 2 Terabyte is being shared through torrents as of per June 2010. In addition these are of a suboptimal format, since they cover a range of keystreams that never occur in practice.

We therefore aimed at generating these tables ourself in order to deliver a proof of concept that a time-memory trade-off attack against A5/1 is feasible. The tables were generated in collaboration with Frank Stevenson and Karsten Nohl, and the code used to for generation and lookup was the ATI-based code provided by Mr. Stevenson. This section will explain the details of how the tables were generated and tested.

4.1 Laboratory Setup

Our first approach was to try and generate tables using the CUDA code, since this was the only code referred to on the A5/1 Security Project website. ATI code was only occasionally mentioned on the project's mailing list, so CUDA code seemed to be the natural way to go. Furthermore, we were already in possession of a Tesla C870 CUDA device, while an ATI card would had to bought if we were to use the ATI code. Unfortunately trying to generate tables with our CUDA device turned out to be a dead end. Almost a month work was spent on trying to get the CUDA code to compile correctly. Since the code were dependent on a lot of other software to be installed, we suspected that maybe a wrong configuration of one of these were the reason that the code would not compile. In the end we finally got an answer to one of our questions on the mailing list, where it was confirmed that there was a bug in the CUDA generation code that prevented the code from compiling.

We therefore shifted towards rainbow table generation with ATI cards instead, that turned out to be an almost instant success. Two different laboratory setups were used during this work; one for rainbow table generation and another one for doing lookup in the tables.

For the rainbow table generation, two different computers were used:

- A custom built computer with an Intel i5 750 2.67GHz processor, 8 GB RAM, two 7200 RPM hard drives of size 320 GB & 2 Terabyte, and two ATI HD 5970 graphic cards.
- A HP xw4300 Workstation with an Intel Pentium 4 processor 521 / 2.80 GHz, 3 GB RAM, one 7200 RPM 250 GB hard drive, and one ATI HD 5870 graphics card

The ATI HD 5870 and ATI HD 5970 can be seen in figure 4.1.

They belong to the ATI Evergreen series, a family of GPUs developed by

(a) ATI HD 5870 (b) ATI HD 5970

Figure 4.1: Picture of the ATI HD 5870 and ATI HD 5970

AMD graphics products division, and are equipped with Graphical Processing Units (GPU) that are capable of processing data a lot faster than a regular Central Processing Unit (CPU). Of that reason, they are also used to do supercomputing, in addition to traditional gaming.

A summary of the performance of both cards can be seen in table 4.1.

	ATI HD 5870	ATI HD 5970
Engine cloock speed	850 MHz	2 x 725 MHz
Processing power (single precision)	2.72 TeraFLOPS	4.64 TeraFLOPS
Processing power (double precision)	544 GigaFLOPS	928 GigaFLOPS
Memory size	1 GB	2 GB
Memory clock speed	1.2 GHz	4 GHz
Memory data rate	4.8 Gbps	4 Gbps
Memory bandwidth	153.6 GB/sec	256.0 GB/sec
A5/1 chains/sec	X	X

Table 4.1: Comparison between the ATI HD 5870 and ATI HD 5970

When it comes to doing table lookup, the setup used was quite simple. A total of 7 computers were used during the lookup stage, with all of them set up to run Ubuntu 9.04 (Jaunty Jackalope)[1].

4.2 Method

Since several computers have been used during the experiment, there might be some differences when it comes to parameter values. Providing the details for all the computers would be of little value, and we have therefore chosen to only use the custom built computer with the twin ATI HD 5970 setup as a reference point in this section.

Before the rainbow table generation could start, we first needed to set up the computer correctly. This meant installing the correct driver for the ATI cards, install supporting software for GPU computing, and finally to download and compile the table generation/table lookup code. A description on how this was done can be found in appendix B.

Table generation

The script that generates the tables is *GenTable.py*. Before executing this script, we changed the number of GPUs to use in order to utilize the GPUs 100%. This was done by changing the last parameter of A5BrookInit() from 1 to 4:

[1] http://releases.ubuntu.com/9.04/

```
###
#
# EDIT: last parameter is number of GPUs to use
#
###
if not a5br.A5BrookInit(c_int(int(id)), c_int(8), c_int(12), 4):
    print "Could not initialize Streams engine. Quitting."
    sys.exit(-1)
```

We also needed to change the chain length from 5,400,000,000 to 8,662,000,000:

```
if total>8662000000:
    print "Table completed. - deleting id file"
    idx = int(id)+8
    os.unlink("a51id.cgi")
```

After this was done, a file called a51id.cgi was created. It is this file that sets which table to create, and the only info it needs to hold is the following line:

```
Your id is: 500
```

Where 500 is the table id number.

The table generation was then started by executing the *GenTable.py* script:

```
./GenTable.py
```

Table sorting

After having produced a table, we needed to sort it. This was done by executing

```
./sort2 /media/rt/tables/500 /media/rt/tables/sorted/500
```

where */media/rt/tables/500* is the path to the unsorted table, whereas */media/rt/tables/sorted/500* is the path to the location where we want to store the sorted table. The output from the sorting is a merge free table that has been reduced in size from approximately 87 GB to 63 GB, in addition to a file *index.dat* that holds the sorting.

Table compression

The next step was to compress the table, both to save disk space and to make it compatible with the lookup tool. We executed the following script in order to compress the table:

```
./CompactTable.py /media/rt/tables/sorted/500 /media/rt/tables/
    lookup/500
```

where */media/rt/tables/sorted/500* is the path to the sorted table, while */media/rt/tables/lookup/500* is the path to the location where the compressed table is to be stored. The output from *CompactTable.py* is a folder containing the table and the *index.dat* file from the sorting.

Note that there is a tool that is able to compress the tables even more, and that makes lookup go even faster. This tool is called *SSDwriter*, and it writes a table directly to block 0 and outwards on a disk. Unfortunately it only supports one table pr. disk as per now, but it is still being developed and a new version that supports multiple tables is likely to come in not to long.

Generating a new challenge set

Since the tools needed to capture a GSM-conversation is still being developed, a test set has been created in order to verify that the tables are working. This test set contains 1000 frames with 114-bits of known plaintext from downlink

traffic. Each table has been encrypted using a unique key that has been thrown away afterwards.

However, it is fully possible to create a new set of different size where the keys has not been thrown away by changing some lines of code in *btest.cpp*. The code that needs to be modified is found within this for-loop:

```
for (int i=0; i < 1000; i++) {
    unsigned char cipherstream[15];
    uint64_t key = 0xf5a472fcfa67c694ULL;
    fread( &key, sizeof(uint64_t), 1 , rnd);
    uint64_t mixed = back.Forwards(key, 100, NULL);
    back.Forwards(mixed, 114, cipherstream);

    fwrite(cipherstream,sizeof(unsigned char),15,fd);
}
```

We changed `i < 1000` to `i < 10000` and added two lines of code after `fread( &key, sizeof(uint64_t), 1 , rnd);`:

```
for (int i=0; i < 10000; i++) {
    unsigned char cipherstream[15];
    uint64_t key = 0xf5a472fcfa67c694ULL;
    fread( &key, sizeof(uint64_t), 1 , rnd);
    printf("Frame: i, key:",i+1;
    std::cout << std::hex << key << "\n";
    uint64_t mixed = back.Forwards(key, 100, NULL);
    back.Forwards(mixed, 114, cipherstream);

    fwrite(cipherstream,sizeof(unsigned char),15,fd);
}
```

Having done this, we then compiled *btest.cpp* by running the following command:

```
g++ -o btest btest.cpp Bidirectional.cpp
```

Then at last we generated a new test set of 10 000 frames by executing the following command:

```
./btest mychallenge.bin > mykeys.txt
```

The key used to encrypt every single frame has been kept in a file named *mykeys.txt*, so that hits in the table can be verified.

Table lookup

Having made a new challenge set, it was now time to perform lookup upon this set. When running make in the *tmto-svn/tinkering/A5Util*-folder, a lookup tool called *a5lookup* gets compiled. However, just recently a new and faster tool named *a5faster* was committed. Both of these tools does lookup based on the assumption that we have 50 keystream samples. This corresponds to having only 50 + 64 - 1 = 113 bits of keystream[2], something that is not the case. Before we compiled the *a5faster* tool, the following for loop was set to do 51 iterations instead of 50:

```
for(int i=0; i<51; i++) {
    // std::cout << std::hex << "Looking at " << cipher << "\n";
    uint64_t rev = Bidirectional::ReverseBits( cipher );
```

Since *a5faster* has not yet been made part of the *Makefile* yet, we had to compile the tool on our own. This was done by running the following command in a terminal window:

[2]This was a small error that was done early in the process, but the number has deliberately not been changed due to consistency in the lookup results.

```
g++ -O2 -o a5faster a5faster.cpp Bidirectional.cpp SSDlookup.cpp
    ../a5_cpu/A5CpuStubs.cpp -ldl
```

Lookup could now be ran through the execution of this command:

```
./a5faster mychallenge.bin 500 /media/rt/tables/lookup/500/index.
    dat /media/rt/tables/lookup/500/blockdevicedata.bin >
    crack500.txt
```

where *mychallenge.bin* is the name of the challenge file, 500 is the table-id, */media/rt/tables/lookup/500/index.dat* is the path to the *index.dat* file from sorting, */media/rt/tables/lookup/500/blockdevicedata.bin* is the path to the table and *crack500.txt* defines the name of the output file.

4.3 Results

In collaboration with Frank A. Stevenson and Karsten Nohl we have produced 41 rainbow tables, that are now publicly available [65]. This is the first time in history that tables capable of cracking A5/1 has been published. These were completed during a 4 week period with 6 ATI graphic cards; 2 x ATI HD 5850, 2 x ATI HD 5870 and 2 x ATI HD 5980.

The tables have been named the *Berlin A5/1 rainbow table set* and fill 1898 GB in size. The generated tables are 100, 108, 116, 124, 132, 140, 148, 156, 164, 172, 180, 188, 196, 204, 212, 220, 230, 238, 250, 260, 268, 276, 284, 292, 324, 332, 340, 348, 356, 364, 372, 380, 388, 396, 404, 412, 420, 428, 436, 492, 500

Our contribution has been to generate tables 220, 230, 238, 250, 340, 348, 356, 364, 372, 380, 388, 396, 404, 412, 420, 428, 436 and 500. In addition we have collaborated with Stevenson on producing tables 100, 108, 116, 124, 132 and

140, where Stevenson have computed the lower part and we the upper part. This means that we have computed a total of 18 full tables, and 6 half tables, meaning that we have generated approximately 51% of the tables in the *Berlin A5/1 rainbow table set*.

We found that the setup that gave the best performance was to have the custom built computer[3] running two scripts simultaneously on two different hard drives. By doing so we were able to produce 2 x 43 000 = 86 000 chains/second. This meant that we were able to finish 2 tables within $\frac{8,662,000,000}{43000} \approx 56 hours$. The HP xw4300 Workstation equipped with the ATI HD 5870 card was running one script and producing 19 900 chains per second. This means that it were able to finish one table within $\frac{8,662,000,000}{43000} \approx 121 hours$.

Since the lookup and disk writing tools is still in the development phase, we have not spent time analyzing this. However it should be noted that a distributed cracking tool that will be able to do lookup in real-time is in the planning stages.

We have been able to do lookup on all tables except from table 284, 324 and 492, since they have not been delivered to us yet. Since the challenge set contained 10 000 frames, a script that processes the output from the cracking was needed. By modifying one of Stevenson's script, we were able to make a script that would process the results in order to give us the coverage of the tables. The original script from Frank Stevenson can be seen in appendix D, while our modified script can be seen in appendix C. The output of the script can be seen here:

```
Total number of false positives is: 459
Total percentage of false positives is: 4.59 percent

Total number of frames found is: 1913
Total number of duplicates is: 320
Total coverage is: 19.13 percent
```

[3]This was the one equipped with two ATI HD 5970 cards

As seen, we have a total coverage of 19.13% on 38 tables[4]. This should serve as proof of concept that a time-memory trade-off attack against A5/1 is feasible.

An observation made during the cracking is the occurrence of 4.59 % false positives. A false positive occurs whenever the immediate output of two A5/1 states results in the same keystream for 64-bits or more.

[4] 41 tables, minus 284, 324 and 492

Chapter 5

Acquiring Network Information

This chapter covers the gathering of network information and aims to give an overview of how the Norwegian networks actually operate in practice. The experiment serves as an initial investigation into network operations and may be useful in the preparation of an attack against GSM.

5.1 Laboratory Setup

This section describes the laboratory setup used for conducting the experiment. The equipment was placed in room F-260 in Elektroblokk F, NTNU Gløshaugen, Trondheim, Norway, consisting of the following hardware:

- Acer Aspire 2920Z with Ubuntu[1] 9.04 32-bit
- Nokia 3310 mobile phone

[1] Open-source Linux operating system, `http://www.ubuntu.com/`

- USB Data Cable[2]

The Nokia 3310 is a dual band GSM900/1800 mobile phone from the year 2000. In order to get Ubuntu to communicate with the Nokia, Gammu[3] had to be installed[5]. Gammu is software for managing cellular devices. The installation and configuration of Gammu can be found in appendix A.

The Nokia can be used for acquiring network information in two different ways; by using NetMonitor and by trace logging. NetMonitor is covered in sections 5.2 and 5.3, while sections 5.4 and 5.5 are dedicated to trace logs.

5.2 Method - NetMonitor

NetMonitor[7, 45, 51, 68] is an extra menu item on the MS that is able to show a variety of information, including network parameters, and is useful as an initial investigation to get an understanding of the mobile phone's view of the network. Figure 5.1 shows the two most used NetMonitor tests; 5.1a is test 1 and 5.1b is test 12. Test 1 is useful for learning which ARFCN (5), timeslot (0) and the type of channel (CCCH) the MS is communicating on. Test 12 shows information on which ciphering algorithm is active (A5/1) and whether frequency hopping is enabled. Other information is also shown in the two tests, but has less importance and is not covered.

[2] Acquired from `http://www.cellphoneshop.net/usbdatcabfor.html`
[3] `http://wammu.eu/gammu/`

(a) Test 1 (b) Test 12

Figure 5.1: Two NetMonitor Tests

Gammu was used to enable NetMonitor in Nokia 3310:

```
gammu --nokianetmonitor 243
```

The three networks active in Norway at the time of writing were surveyed, namely Telenor, Netcom and Network Norway. Three scenarios for each of the three networks were executed; Mobile Terminating SMS, Mobile Originating Call and Mobile Terminating Call. Mobile Originating SMS was not performed due to a practical limitation; the text *Sending Message* shows on the display while the SMS is being sent, and thus makes NetMonitor inaccessible.

5.3 Results - NetMonitor

Table 5.1 is a presentation of the extracted network information. The MCC and MNC for Telenor, NetCom and Network Norway are given. Further, the beacon channels' ARFCN, where the MS listens while idle, is showed. Note that in the case of both Telenor and NetCom two beacon channels were detected, probably because two BTSs have similar signal strength at the location of the

experiment, and the MS was therefore being handed over between the two. The BSIC, LAC and CID for each identified BTS are also given. Then follows the data on the different scenarios.

The most notable results are made bold in the table. Both Telenor and NetCom are changing to a different ARFCN than the beacon channel when receiving SMS. However, when receiving SMS on Network Norway, the MS seems to stay on the beacon channel, only changing to timeslot 1. This will turn out to be of specific relevance when performing the interception experiment in chapter 6.

Ciphering with A5/1 is activated for all the scenarios, but the sending and receiving of SMS need some discussion. When the authors of this thesis contacted Telenor and NetCom and asked whether SMS were encrypted, Telenor answered that they were *not* because SMS are signaling traffic. NetCom refused to give an answer. However, NetMonitor showed ciphering to be activated while receiving SMS. It may be the case that NetMontior shows ciphering to be enabled after the Nokia receives the *Cipher Mode Command* and returns the *Cipher Mode Complete* messages, even if ciphering is not applied to the subsequent SMS.

Other important results are that frequency hopping during calls seems to be turned off for the BTSs on ARFCNs 5 and 983 for NetCom and Network Norway, respectively. Telenor has frequency hopping activated on both the detected BTSs.

	Telenor	**NetCom**	**Network Norway**
MCC | MNC	242 | 01	242 | 02	242 | 05
ARFCN of beacon channel (C0)	66/71	5/21	983
BSIC value	56/56	3/12	11
Location Area Code (LAC)	15101/15101	305/305	1040
Cell Identifier (CID)	16510/16085	16873	4112
MT-SMS SDCCH ARFCN	91/52	17/9	**983, timeslot change**
MT-SMS, ciphering	A5/1	A5/1	A5/1
MOC, hopping	On/On	**Off**/On	**Off**
MOC, ciphering	A5/1	A5/1	A5/1
MTC, hopping	On/On	**Off**/On	**Off**
MTC, ciphering	A5/1	A5/1	A5/1

Table 5.1: Captured GSM Network Information

5.4 Method - Trace Logs

Nokia used a simple remote logging facility for debugging their firmwares remotely, but apparently forgot to remove this feature when going into production. It can be enabled by executing a simple bus command with gammu. Debug tracing provides useful information and insight to the operational behavior of GSM.

The file `nhm5_587.txt`[4] is needed to decode trace types. To perform a trace[5]:

```
gammu --nokiadebug nhm5_587.txt v20-25,v18-19
```

The trace runs until `Ctrl+C` is pressed, and the file `out.xml` is produced. Wireshark is able to open the XML-file and give a human-readable interpretation of the trace. Wireshark is an open source network analyzer that among other features allows for analysis of GSM traffic. More information about the tool, as well as download options can be found at the project's website[5].

Five scenarios were executed with trace logging activated; IMSI attach, MOC, MTC, MO-SMS and MT-SMS. Message sequence diagrams (MSD) for the different scenarios were made based on the human-readable interpretation by Wireshark, and were drawn in Microsoft Visio 2007 with the Sandrila[6] add-in. The MSDs can be found in section 5.5.1.

Some of the individual messages encountered were investigated more in detail with help from `gsmdecode` from the AirProbe software project, which will be introduced more thoroughly in section 6.1.3. At this point it is sufficient to know that `gsmdecode` converts the raw bits from the air interface to a human-readable format, similar to the work Wireshark does. However, it provides an

[4] `https://svn.berlin.ccc.de/projects/airprobe/attachment/wiki/tracelog/nhm5_587.txt`

[5] `http://www.wireshark.org/`, version 1.2.7 was used

[6] `http://www.sandrila.co.uk/visio-sdl/`

interpretation in a format much better suited for this report. The following command was used to decode the trace logs:

```
gsmdecode < out.xml > decoded.txt
```

`gsmdecode` uses the output file `out.xml` from gammu and produces the file `decoded.txt`, which is a plain text file containing the decoded data.

The decoded message examples with discussion can be found in section 5.5.2.

5.5 Results - Trace Logs

This section presents the results from the performed trace logs.

5.5.1 Message Sequence Diagrams

Figures 5.2, 5.3, 5.4, 5.5 and 5.6 are the message sequence diagrams of the scenarios. Most of the messages are at layer 3 in the protocol architecture, and in front of each message the type of sublayer has been indicated by RR (Radio Resource Management), MM (Mobility Management) or CC (Call Control). An explanation of these can be found in section 2.3.

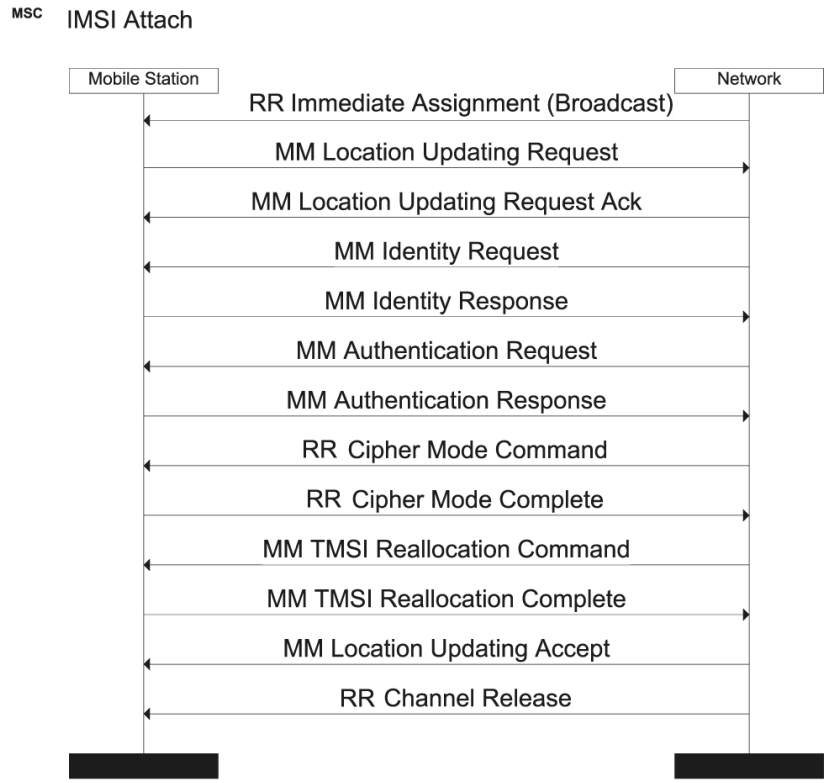

Figure 5.2: Message Sequence Diagram showing the scenario where the Nokia performs an IMSI attach to the network. Notice that the network asks for the IMSI of the MS in an Identity Request message as it does not recognize the received TMSI.

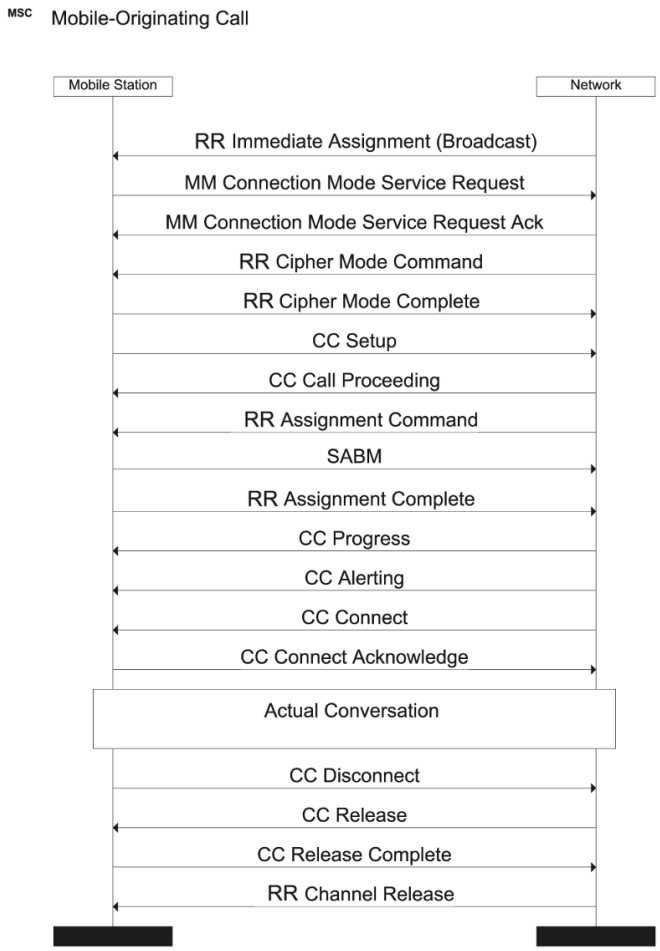

Figure 5.3: Message Sequence Diagram showing the scenario *Mobile-Originating Call* using early assignment (Non-OACSU) as expected. This diagram confirms the theory given in in section 2.15. However, the *Channel Request* could not be seen in Wireshark due to unknown reason, it may be the case that the Nokia software drops this message when trace logging. The MOC proceeds as expected, but note that authentication is skipped because the *Connection Mode Service Request* contains a network acceptable CKSN.

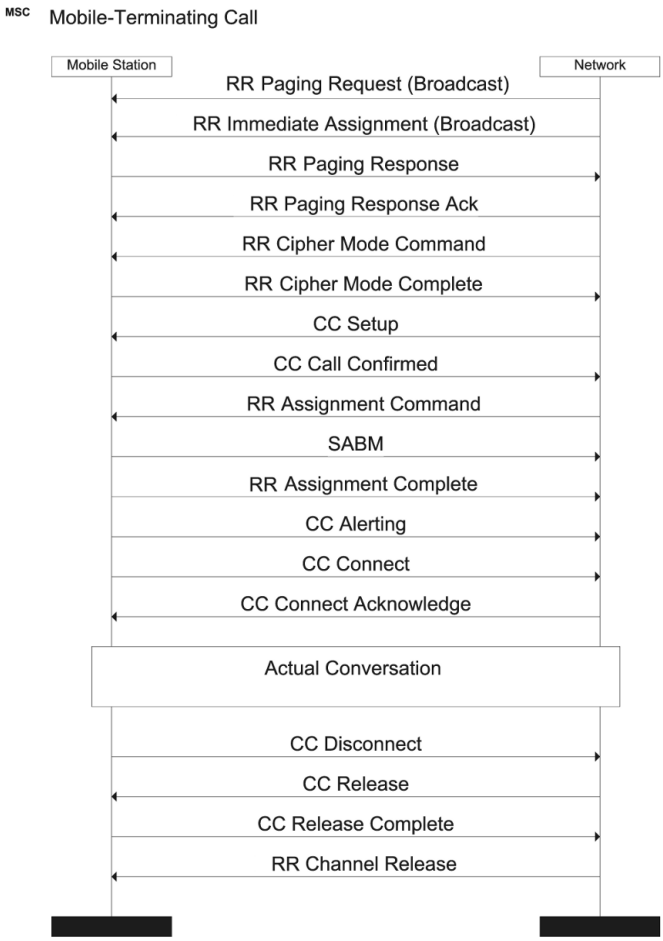

Figure 5.4: Message Sequence Diagram showing the scenario *Mobile-Terminating Call* using early assignment (Non-OACSU). It confirms the theory given in in section 2.15. The *Channel Request* has again been left out.

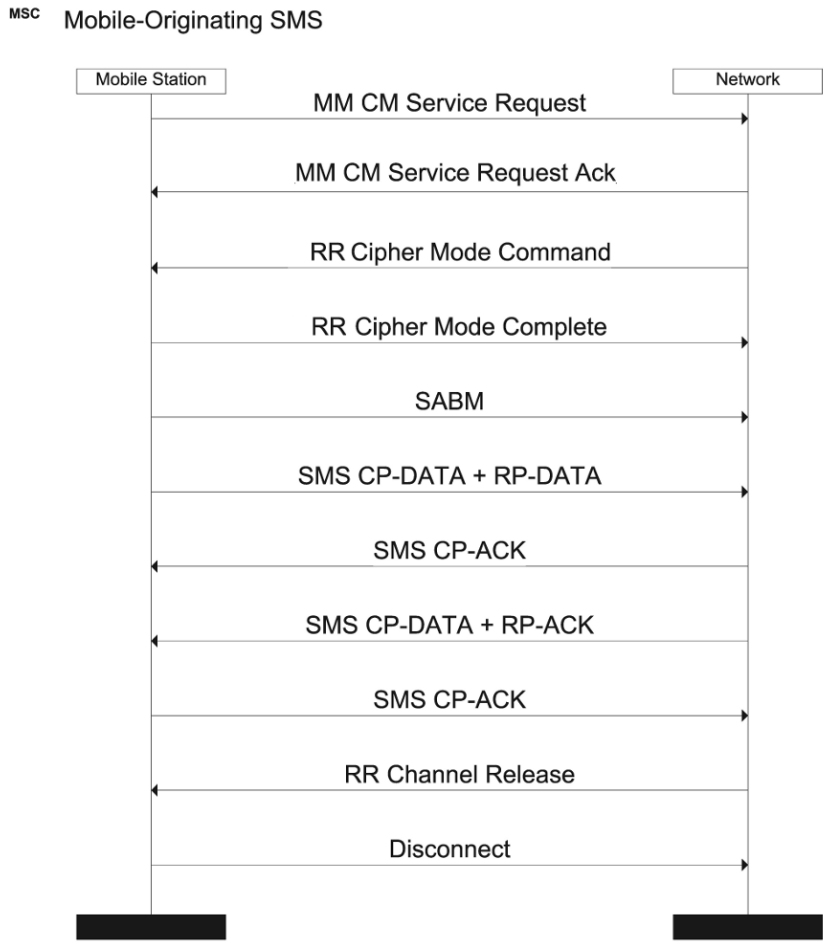

Figure 5.5: Message Sequence Diagram showing the scenario *Mobile-Originating SMS*. Notice that ciphering is performed, even though Telenor stated that SMS messages were not encrypted.

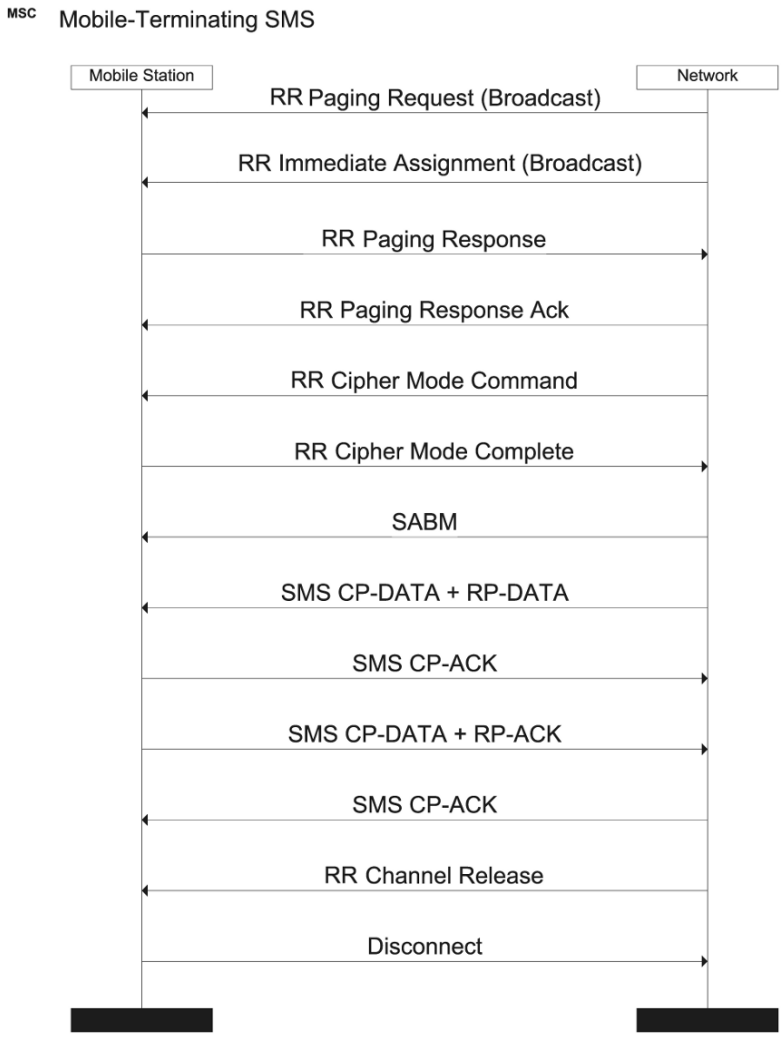

Figure 5.6: Message Sequence Diagram showing the scenario *Mobile-Terminating SMS*.

5.5.2 Decoded Message Examples

Listings 5.1-5.17 show the decoded examples of many of the encountered messages. Appendix H explains the LAPDm frame structure, format of fields and procedures needed to understand these messages. Each listing is built up with the raw hex-values on the top, and the decoded, human-readable interpretation, below.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 25 06 21 00 05 f4 0d 44 - 6d b5 2b 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 25 001001-- Pseudo Length: 9
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 21 00100001 Paging Request Type 1
    3: 00 ------00 Page Mode: Normal paging
    5: f4 -----100 Type of identity: TMSI/P-TMSI
    6: 0d -------- ID(4/even): 0D446DB5
```

Listing 5.1: *RR Paging Request* message sent downlink on the PCH at ARCFN 5. This is a standard paging request, Type 1. There is a total of three paging types. Type 2 and Type 3 are used for paging several MSs at once. The MS is being paged with the TMSI 0D446DB5. The *RR Paging Response* can be found in listing 5.3. Listing 5.2 is an example of another *RR Paging Request* using IMSI to identify the MS.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 31 06 21 00 08 29 24 20 - 65 10 24 00 39 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 31 001100-- Pseudo Length: 12
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 21 00100001 Paging Request Type 1
    3: 00 ------00 Page Mode: Normal paging
    5: 29 -----001 Type of identity: IMSI
    6: 24 -------- ID(7/odd): 24202XXXXXXXXX
```

Listing 5.2: *RR Paging Request* message sent downlink on the PCH at ARCFN 5. The MS being paged with IMSI as identifier. The MSIN of the IMSI has been censored with "X" because it uniquely identifies some mobile subscriber.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 3f 35 06 27 06 03 33 - 19 81 05 f4 0d 44 6d b5
001: 2b 2b 2b 2b 2b 2b 2b 2b
   0: 01 -------1 Extended Address: 1 octet long
   0: 01 ------0- C/R: Response
   0: 01 ---000-- SAPI: RR, MM and CC
   0: 01 -00----- Link Protocol Discriminator: GSM (not Cell Broadcasting)
   1: 3f ------11 Unnumbered Frame
   1: 3f ---1---- P
   1: 3f 011-11-- SABM frame (Set asynchronous balance mode)
   2: 35 -------1 EL, Extended Length: y
   2: 35 ------0- M, segmentation: N
   2: 35 001101-- Length: 13
   3: 06 0------- Direction: From originating site
   3: 06 -000---- 0 TransactionID
   3: 06 ----0110 Radio Resource Management
   4: 27 0-100111 RRpagingResponse
   4: 27 -x------ Send sequence number: 0
   5: 06 -----110 Ciphering key sequence: 6
   5: 06 -000---- Ciphering key sequence: 0
   6: 03 00000011 MS Classmark 2 length: 3
   7: 33 -01----- Revision Level: Phase 2
   7: 33 ---1---- Controlled early classmark sending: Implemented
   7: 33 ----0--- A5/1 available
   7: 33 -----011 RF power class capability: Class 4
   8: 19 -1------ Pseudo Sync Capability: not present
   8: 19 --01---- SS Screening: Phase 2 error handling
   8: 19 ----1--- Mobile Terminated Point to Point SMS: supported
   8: 19 -----0-- VoiceBroadcastService: not supported
   8: 19 ------0- VoiceGroupCallService: not supported
   8: 19 -------1 MS supports E-GSM or R-GSM: supported
   9: 81 1------- CM3 option: supported
   9: 81 --0----- LocationServiceValueAdded Capability: not supported
   9: 81 ----0--- SoLSA Capability: not supported
   9: 81 ------0- A5/3 not available
   9: 81 -------1 A5/2: available
  11: f4 -----100 Type of identity: TMSI/P-TMSI
  12: 0d -------- ID(4/even): 0D446DB5
```

Listing 5.3: *RR Paging Response* message sent uplink on SDCCH. This includes the Ciphering Key Sequence Number, 6 in the example, as well as information on the capabilities of the MS; A5/1 and A5/2 are available. The TMSI is also attached.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 31 06 3f 00 59 70 25 f4 - 45 62 01 01 03 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 31 001100-- Pseudo Length: 12
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 3f 0-111111 RRimmediateAssignment
    2: 3f -x------ Send sequence number: 0
    3: 00 ------00 Page Mode: Normal paging
    3: 00 -0------ No meaning
    3: 00 --0----- Downlink assign to MS: No meaning
    3: 00 ---0---- This messages assigns a dedicated mode resource
    4: 59 -----001 Timeslot number: 1
    4: 59 01011--- Channel Description: SDCCH/8 + SACCH/C8 or CBCH
         (SDCCH/8), SC3
    5: 70 011----- Training seq. code : 3
    5: 70 ---1---- HoppingChannel
    6: 25 ........ Mobile Allocation Index Offset (MAIO) 0
    6: 25 --100101 Hopping Seq. Number: 37
    7: f4 111----- Establishing Cause: Other services req. by user
    7: f4 ---xxxxx Random Reference : 20
    8: 45 xxxxxxxx T1/T2/T3
    9: 62 xxxxxxxx T1/T2/T3
   10: 01 --xxxxxx Timing advance value: 1
   11: 01 00000001 Length of Mobile Allocation: 1
   12: 03 ------1- Mobile Allocation ARFCN #2
   12: 03 -------1 Mobile Allocation ARFCN #1
```

Listing 5.4: *RR Immediate Assignment* message sent downlink on the AGCH at ARFCN 5. This message assigns a dedicated channel to the MS and it includes a specific timeslot, a list of ARFCNs, the HSN and the MAIO.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 3f 35 05 24 51 03 33 - 19 81 05 f4 0d 44 6d b5
001: 2b 2b 2b 2b 2b 2b 2b 2b
   0: 01 -------1 Extended Address: 1 octet long
   0: 01 ------0- C/R: Response
   0: 01 ---000-- SAPI: RR, MM and CC
   0: 01 -00----- Link Protocol Disciminator: GSM (not Cell Broadcasting)
   1: 3f ------11 Unnumbered Frame
   1: 3f ---1---- P
   1: 3f 011-11-- SABM frame (Set asynchonous balance mode)
   2: 35 -------1 EL, Extended Length: y
   2: 35 ------0- M, segmentation: N
   2: 35 001101-- Length: 13
   3: 05 0------- Direction: From originating site
   3: 05 -000---- 0 TransactionID
   3: 05 ----0101 Mobile Management Message (non GPRS)
   4: 24 00------ SendSequenceNumber: 0
   4: 24 --100100 MMcmServiceRequest
   5: 51 -101---- Ciphering key sequence: 5
   5: 51 ----0001 Request Service Type: MS originated call
   6: 03 00000011 MS Classmark 2 length: 3
   7: 33 -01----- Revision Level: Phase 2
   7: 33 ---1---- Controlled early classmark sending: Implemented
   7: 33 ----0--- A5/1 available
   7: 33 -----011 RF power class capability: Class 4
   8: 19 -1------ Pseudo Sync Capability: not present
   8: 19 --01---- SS Screening: Phase 2 error handling
   8: 19 ----1--- Mobile Terminated Point to Point SMS: supported
   8: 19 -----0-- VoiceBroadcastService: not supported
   8: 19 ------0- VoiceGroupCallService: not supported
   8: 19 -------1 MS supports E-GSM or R-GSM: supported
   9: 81 1------- CM3 option: supported
   9: 81 --0----- LocationServiceValueAdded Capability: not supported
   9: 81 ----0--- SoLSA Capability: not supported
   9: 81 ------0- A5/3 not available
   9: 81 -------1 A5/2: available
  11: f4 -----100 Type of identity: TMSI/P-TMSI
  12: 0d -------- ID(4/even): 0D446DB5
```

Listing 5.5: *MM Service Request* message sent uplink on the SDCCH at ARCFN 5. It contains the Ciphering Key Sequence Number, as well as information on which ciphering algorithms that are available. The supported GSM frequency bands are also included.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 03 42 4d 05 12 00 15 e3 - 74 55 08 af 7d 53 cc 70
001: 77 49 39 04 39 ab 2b
    0: 03 -------1 Extended Address: 1 octet long
    0: 03 ------1- C/R: Command
    0: 03 ---000-- SAPI: RR, MM and CC
    0: 03 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 42 -------0 Information Frame
    1: 42 ----001- N(S), Sequence counter: 1
    1: 42 ---0---- P
    1: 42 010----- N(R), Retransmission counter: 2
    2: 4d -------1 EL, Extended Length: y
    2: 4d ------0- M, segmentation: N
    2: 4d 010011-- Length: 19
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 12 00------ SendSequenceNumber: 0
    4: 12 --010010 Authentication Request
    5: 00 -----000 Cipher Key Sequence Number: 0
    6: 15 -------- RAND: 15e3745508af7d53cc707749390439ab
```

Listing 5.6: A downlink *MM Authentication Request* on SDCCH. Important parts of the message are the new CKSN, in this case 0, as well as the 128-bit RAND value. In the *MM Authentication Response* shown in listing 5.7 the MS returns the 32-bit SRES value.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 44 19 05 14 54 58 63 - ca 2b 2b 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 01 -------1 Extended Address: 1 octet long
    0: 01 ------0- C/R: Response
    0: 01 ---000-- SAPI: RR, MM and CC
    0: 01 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 44 -------0 Information Frame
    1: 44 ----010- N(S), Sequence counter: 2
    1: 44 ---0---- P
    1: 44 010----- N(R), Retransmission counter: 2
    2: 19 -------1 EL, Extended Length: y
    2: 19 ------0- M, segmentation: N
    2: 19 000110-- Length: 6
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 14 00------ SendSequenceNumber: 0
    4: 14 --010100 Authentication Response
    5: 54 -------- SRES: 545863ca
```

Listing 5.7: An uplink *MM Authentication Response* message on SDCCH. The message contains the 32-bit SRES value.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 03 22 0d 06 35 11 2b 2b - 2b 2b 2b 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 03 -------1 Extended Address: 1 octet long
    0: 03 ------1- C/R: Command
    0: 03 ---000-- SAPI: RR, MM and CC
    0: 03 -00----- Link Protocol Discriminator: GSM (not Cell
       Broadcasting)
    1: 22 -------0 Information Frame
    1: 22 ----001- N(S), Sequence counter: 1
    1: 22 ---0---- P
    1: 22 001----- N(R), Retransmission counter: 1
    2: 0d -------1 EL, Extended Length: y
    2: 0d ------0- M, segmentation: N
    2: 0d 000011-- Length: 3
    3: 06 0------- Direction: From originating site
    3: 06 -000---- 0 TransactionID
    3: 06 ----0110 Radio Resource Management
    4: 35 00110101 RR Cipher Mode Command
    5: 11 ----000- Cipher: A5/1
    5: 11 -------1 Start ciphering
    5: 11 ---1---- Cipher Response: IMEISV shall be included
```

Listing 5.8: *RR Cipher Mode Command* message sent downlink on SDCCH at ARFCN 5. This message is used to tell the MS to start ciphering with, in this case, the A5/1 cipher. Note also that the BTS asks the MS to return its IMEISV in the *RR Cipher Mode Complete* message, shown in listing 5.9.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 44 35 06 32 17 09 33 - 05 01 13 80 25 59 17 f0
001: 2b 2b 2b 2b 2b 2b 2b
    0: 01 -------1 Extended Address: 1 octet long
    0: 01 ------0- C/R: Response
    0: 01 ---000-- SAPI: RR, MM and CC
    0: 01 -00----- Link Protocol Discriminator: GSM (not Cell
       Broadcasting)
    1: 44 -------0 Information Frame
    1: 44 ----010- N(S), Sequence counter: 2
    1: 44 ---0---- P
    1: 44 010----- N(R), Retransmission counter: 2
    2: 35 -------1 EL, Extended Length: y
    2: 35 ------0- M, segmentation: N
    2: 35 001101-- Length: 13
    3: 06 0------- Direction: From originating site
    3: 06 -000---- 0 TransactionID
    3: 06 ----0110 Radio Resource Management
    4: 32 00110010 RR Cipher Mode Complete
    7: 33 -----011 Type of identity: IMEISV
    8: 05 -------- ID(8/even): 35010310XXXXXX10
```

Listing 5.9: The *RR Cipher Mode Complete* message is sent uplink on SDCCH and contains the IMEISV. The IMEISV has been partly censored with "X" as requested by the owner of the MS because it uniquely identifies the mobile equipment. This message is sent encrypted, but the trace log shows the message before encryption.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 03 88 21 06 2e 0a 63 ed - 05 63 21 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 03 -------1 Extended Address: 1 octet long
    0: 03 ------1- C/R: Command
    0: 03 ---000-- SAPI: RR, MM and CC
    0: 03 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 88 -------0 Information Frame
    1: 88 ----100- N(S), Sequence counter: 4
    1: 88 ---0---- P
    1: 88 100----- N(R), Retransmission counter: 4
    2: 21 -------1 EL, Extended Length: y
    2: 21 ------0- M, segmentation: N
    2: 21 001000-- Length: 8
    3: 06 0------- Direction: From originating site
    3: 06 -000---- 0 TransactionID
    3: 06 ----0110 Radio Resource Management
    4: 2e 00101110 RR Assign Command
    5: 0a -----010 Timeslot number: 2
    5: 0a 00001--- Channel Description: TCH/F + ACCHs
    6: 63 011----- Training seq. code: 3
    7: ed ........ Absolute RF channel number: 1005
    8: 05 ---00101 Power Level: 5
   10: 21 00100001 Channel Mode: TCH/F or TCH/H rev 2
```

Listing 5.10: *RR Assignment Command* message sent downlink on SDCCH at ARFCN 5. It is used to get the MS off from the SDCCH and onto a TCH+FACCH. In this example the MS is told to use timeslot 2 on ARFCN 1005. The MS acknowledges this transaction with an *RR Assignment Complete* message on the FACCH. From this point on, all signaling transactions are performed on that channel.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 3f 3d 05 08 32 42 f2 - 20 01 31 33 05 f4 0a f6
001: 0f 67 2b 2b 2b 2b 2b
    0: 01 -------1 Extended Address: 1 octet long
    0: 01 ------0- C/R: Response
    0: 01 ---000-- SAPI: RR, MM and CC
    0: 01 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 3f ------11 Unnumbered Frame
    1: 3f ---1---- P
    1: 3f 011-11-- SABM frame (Set asynchronous balance mode)
    2: 3d -------1 EL, Extended Length: y
    2: 3d ------0- M, segmentation: N
    2: 3d 001111-- Length: 15
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 08 00------ SendSequenceNumber: 0
    4: 08 --001000 MM Location Update Request
    5: 32 -011---- Cipher Key Sequence Number: 3
    5: 32 ----0--- No follow-on request pending
    5: 32 ------10 Location Update: IMSI attach
    6: 42 242 Mobile Country Code (Norway)
    7: f2 02f Mobile Network Code (Netcom GSM AS)
    9: 01 305 [0x0131] Local Area Code
   11: 33 -01----- Revision Level: Phase 2
   11: 33 ---1---- Controlled early classmark sending: Implemented
   11: 33 ----0--- A5/1 available
   11: 33 -----011 RF power class capability: Class 4
   13: f4 -----100 Type of identity: TMSI/P-TMSI
   14: 0a -------- ID(4/even): 0AF60F67
```

Listing 5.11: *MM Location Updating Request* message sent uplink on the SDCCH. The type of location update in the example is an IMSI attach, which happens initially when a MS wants to associate to a network. The *Cipher Key Sequence Number* is included, and if it matches the CKSN the network remembers from earlier, no authentication needs to be performed. The TMSI 0AF60F67 is in this example not recognized by the network, and thus a *MM Identity Request* message will follow from the network. An example of such a message can be found in listing 5.12.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 03 20 0d 05 18 01 2b 2b - 2b 2b 2b 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 03 -------1 Extended Address: 1 octet long
    0: 03 ------1- C/R: Command
    0: 03 ---000-- SAPI: RR, MM and CC
    0: 03 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 20 -------0 Information Frame
    1: 20 ----000- N(S), Sequence counter: 0
    1: 20 ---0---- P
    1: 20 001----- N(R), Retransmission counter: 1
    2: 0d -------1 EL, Extended Length: y
    2: 0d ------0- M, segmentation: N
    2: 0d 000011-- Length: 3
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 18 00------ SendSequenceNumber: 0
    4: 18 --011000 MMIdentityRequest
    5: 01 -----001 Type of Identity: IMSI
```

Listing 5.12: *MM Identity Request* message sent downlink on the SDCCH. It mainly contains the type of identity to be used - IMSI in this case since the TMSI was unknown by the network.

```
HEX l2_data_out_B:194 Format B DATA (up)
000: 01 22 2d 05 59 08 29 24 - 50 70 20 44 54 90 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 01 -------1 Extended Address: 1 octet long
    0: 01 ------0- C/R: Response
    0: 01 ---000-- SAPI: RR, MM and CC
    0: 01 -00----- Link Protocol Discriminator: GSM (not Cell
       Broadcasting)
    1: 22 -------0 Information Frame
    1: 22 ----001- N(S), Sequence counter: 1
    1: 22 ---0---- P
    1: 22 001----- N(R), Retransmission counter: 1
    2: 2d -------1 EL, Extended Length: y
    2: 2d ------0- M, segmentation: N
    2: 2d 001011-- Length: 11
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 59 01------ SendSequenceNumber: 1
    4: 59 --011001 MMidentityResponse
    6: 29 -----001 Type of identity: IMSI
    7: 24 -------- ID(7/odd): 24205XXXXXXXXX
```

Listing 5.13: The *MM Identity Response* message from the MS on the SDCCH. Most notable here is the IMSI, censored with "X" because it uniquely identifies a mobile subscriber.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 0f 00 53 19 01 1e 01 00 - 06 91 74 74 19 09 00 00
001: 13 00 0a 91 74 39 66
    0: 0f -------1 Extended Address: 1 octet long
    0: 0f ------1- C/R: Command
    0: 0f ---011-- SAPI: SMS and SS
    0: 0f -00----- Link Protocol Discriminator: GSM (not Cell Broadcasting)
    1: 00 -------0 Information Frame
    1: 00 ----000- N(S), Sequence counter: 0
    1: 00 ---0---- P
    1: 00 000----- N(R), Retransmission counter: 0
    2: 53 -------1 EL, Extended Length: y
    2: 53 ------1- M, segmentation: Y
    2: 53 010100-- Length: 20
    3: 19 -------- [SEGMENTED MESSAGE. MORE DATA FOLLOWS...]
```

Listing 5.14: The first segment of a *SMS CP-DATA+RP-DATA* message sent downlink on SDCCH at ARFCN 5. The last segment can be found in listing 5.15, and the reassembled message is shown in listing 5.16.

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 0f 02 35 05 91 00 00 01 - 50 41 41 72 14 80 01 4b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 0f -------1 Extended Address: 1 octet long
    0: 0f ------1- C/R: Command
    0: 0f ---011-- SAPI: SMS and SS
    0: 0f -00----- Link Protocol Discriminator: GSM (not Cell Broadcasting)
    1: 02 -------0 Information Frame
    1: 02 ----001- N(S), Sequence counter: 1
    1: 02 ---0---- P
    1: 02 000----- N(R), Retransmission counter: 0
    2: 35 -------1 EL, Extended Length: y
    2: 35 ------0- M, segmentation: N
    2: 35 001101-- Length: 13
    3: 05 -------- [SEGMENTED MESSAGE. LAST...]
```

Listing 5.15: The last segment of the *SMS CP-DATA+RP-DATA* message sent downlink on SDCCH at ARFCN 5. The message has been reassembled in listing 5.16.

```
HEX l2_data_out_B:294 Format SMS data
000: 19 01 1e 01 00 06 91 74 - 74 19 09 00 00 13 00 0a
001: 91 74 39 66 05 91 00 00 - 01 50 41 41 72 14 80 01
002: 4b
    0: 19 0------- Direction: From originating site
    0: 19 -001---- 1 TransactionID
    0: 19 ----1001 SMS messages
    1: 01 00000001 Type: CP-DATA
    1: 01 00000001 Length: 1
    2: 1e 00011110 Parameter 30
    3: 01 00000001 Parameter 1
    4: 00 00000000 Parameter 0
    5: 06 00000110 SMSC Address Length: 6
    6: 91 1------- Extension
    6: 91 -001---- International Number
    6: 91 ----0001 Numbering plan: ISDN/telephone (E164/E.163)
    7: 74 -------- Number(5): 4747919000
   12: 00 00000000 Message Flags: 0
   13: 13 00010011 Reference Number [continue]
   14: 00 00000000 Reference Number: 4864
   15: 0a 00001010 Destination Address Length: 10
   16: 91 1------- Extension
   16: 91 -001---- International Number
   16: 91 ----0001 Numbering plan: ISDN/telephone (E164/E.163)
   17: 74 -------- Number(9): 47XXXXXXXX00001005
   26: 41 01000001 Protocol Identifier: 0x41
   27: 41 01000001 reserved
   29: 14 XXXXXXXX UNKNOWN DATA (4 bytes)
   29: 14 YYYYYYYY REST OCTETS (4)
```

Listing 5.16: The reassembled *SMS CP-DATA+RP-DATA* message from the segments in listings 5.14 and 5.15. In this example the last byte "4b" represents the actual content of the SMS message: "K". The sender's phone number has been censored with "X".

```
HEX l2_data_out_B:194 Format B DATA (down)
000: 03 86 35 05 1a 42 f2 10 - 3a fd 05 f4 a0 05 b6 38
001: 2b 2b 2b 2b 2b 2b 2b
    0: 03 -------1 Extended Address: 1 octet long
    0: 03 ------1- C/R: Command
    0: 03 ---000-- SAPI: RR, MM and CC
    0: 03 -00----- Link Protocol Discriminator: GSM (not Cell
        Broadcasting)
    1: 86 -------0 Information Frame
    1: 86 ----011- N(S), Sequence counter: 3
    1: 86 ---0---- P
    1: 86 100----- N(R), Retransmission counter: 4
    2: 35 -------1 EL, Extended Length: y
    2: 35 ------0- M, segmentation: N
    2: 35 001101-- Length: 13
    3: 05 0------- Direction: From originating site
    3: 05 -000---- 0 TransactionID
    3: 05 ----0101 Mobile Management Message (non GPRS)
    4: 1a 00------ SendSequenceNumber: 0
    4: 1a --011010 TMSI Reallocation Command
    5: 42 242 Mobile Country Code (Norway)
    6: f2 01f Mobile Network Code (Telenor Mobil AS)
    8: 3a 15101 [0x3afd] Local Area Code
   11: f4 -----100 Type of identity: TMSI/P-TMSI
   12: a0 -------- ID(4/even): A005B638
```

Listing 5.17: *MM TMSI Reallocation Command* sent downlink on SDCCH to assign a new TMSI to the MS. This happens after ciphering is activated.

Chapter 6

Intercepting GSM Traffic

This chapter covers the use of AirProbe to capture GSM signaling traffic. An introduction to the USRP and GNU Radio is also given.

6.1 Laboratory Setup

This section describes the laboratory setup used for conducting the experiment. The equipment was placed in room F-260 in Elektroblokk F, NTNU Gløshaugen, Trondheim, Norway. The following hardware was used in this investigation:

- Acer Aspire 2920Z with Ubuntu[1] 9.04 32-bit, GNU Radio and AirProbe
- Universal Software Radio Peripheral (USRP)

[1] Open-source Linux operating system, `http://www.ubuntu.com/`

6.1.1 USRP

The USRP (Universal Software Radio Peripheral) is an open-source hardware device that allows a computer to function as a software radio. The main idea is to make it possible to implement various radio communication systems simply by replacing the software running on the computer. The USRP was created by Matt Ettus in an effort to solve the problem of getting signaling samples in and out of a computer at a high rate. Picture 6.1 shows how the USRP looks like externally.

Figure 6.1: The USRP

The USRP comes with a built in motherboard featuring four analog-to-digital (ADC), four digital-to-analog converters (DAC), and a field-programmable gate array (FPGA) for high-speed signal processing.

In addition, the motherboard support multiple extension cards, also known as daughterboards, to serve as radio frequency (RF) front end. The daughterboard acts as an interface between the USRP and the radio world as they translate the

signal between the antenna and the ADC/DAC. The block diagram in figure 6.2 shows how the components in the USRP are connected together.

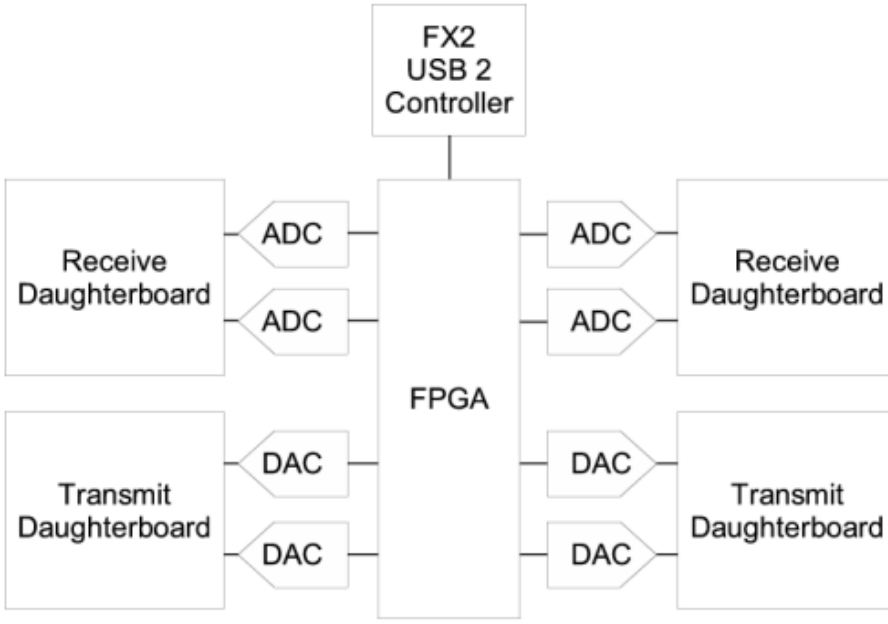

Figure 6.2: Universal Software Radio Peripheral Block Diagram. From [21].

Daughterboards are available as either transmitters, receivers or both, and are designed to operate at specific frequency bands. The USRP daughterboards operating in the GSM frequency range include the DBSRX, RFX900 and RFX1800 cards; they operate at 800-2400 MHz, 800-1000 MHz and 1.5-2.1 GHz respectively. In addition, an antenna must be connected to the USRP in order to transmit or receive signals through the air. Antennas such as the LP0410 (400-1000 MHz), LP0926 (900-2600 MHz) and VERT900 (824-960 MHz, 1710-1990 MHz) are all capable of tuning into the GSM frequency band.

	USRP	**USRP2**
Interface	USB 2.0	Gigabit Ethernet
FPGA	ltera EP1C12	Xilinx Spartan 3 20006
Internal clock	64 MHz	100 MHz
RF Bandwidth to/from host	8 MHz @ 16bits	25 MHz @ 16bits
Cost	700$	1400$
ADC Samples	12-bit, 64 MS/s	14-bit, 100 MS/s
DAC Samples	14-bit, 128 MS/s	16-bit, 400 MS/s
Daughterboard capacity	2 TX, 2 RX	1 TX, 1 RX
SRAM	None	1 Megabyte
Power	6V, 3A	6V, 3A
AirProbe support	Yes	Limited
OpenBTS support	Yes	No

Table 6.1: Comparison between the USRP and USRP2. Extended from [3].

The DBSRX card covers all GSM frequencies, but is a receive only board. A RFX card is needed in order to actively transmit. However, be aware of that most countries require a license to transmit, as GSM operates on regulated frequency bands.

A computer connects to the USRP device over a USB 2.0 cable. A newer version, USRP2, was made available in May 2009 offering a faster FPGA and a Gigabit Ethernet interface. A comparison between the two USRP versions are shown in table 6.1.

The research in this report uses the original USRP with the RFX900 daughterboard and a VERT900 antenna.

6.1.2 GNU Radio

GNU Radio is an open-source software toolkit for building and deploying software radio systems. The framework of GNU Radio provides signal processing runtime and processing blocks for communicating with external hardware (e.g.

a USRP). It is widely used in hobbyist, academic and commercial environments to support wireless communications research as well as to implement real-world radio systems. GNU Radio applications are primarily written using the Python programming language, while the supplied, performance-critical signal processing path is implemented in C++ [3].

GNU Radio has, since it first started in 1998, been a collaborative effort of online community constantly improving and developing new code. The GNU Radio and USRP combined form a system often referred to as software defined radio (SDR).

GNU Radio does not itself contain any GSM sniffing capabilities, although it can be used to locate the beacon frequency of an active BTS [44]. However, GNU Radio can be used in combination with other software packages, like AirProbe, to perform reception and demodulation of GSM signaling traffic.

A tutorial on how to set up GNU Radio on Ubuntu can be found in appendix E.

6.1.3 AirProbe

AirProbe is a software project aimed at developing open-source GSM air interface analysis tools. It is divided into three main subprojects: acquisition, demodulation and analysis [1]:

- The acquisition module is hardware dependent and contains everything that has to do with receiving and digitizing the air interface.
- The demodulation module contains all necessary code to make bits out of the signal captured by acquisition.
- The analysis module contains all the protocol parsing and decoding capabilities. Wireshark can, for instance, be used to handle parts of the

visualization tasks.

A tutorial on how to set up AirProbe can be found in Appendix F.

6.2 Method

The following parts of AirProbe were mainly used in this experiment:

- `gsm-receiver`
- `gsm-tvoid`
- `gsmdecode`

The latest and most popular part of AirProbe is the `gsm-receiver` application. This program implements GSM layer 1, 2, and some layer 3 functionality. It supports currently only downlink decoding of one channel, and will by default capture the first timeslot of a given frequency. An alternative to `gsm-receiver` is `gsm-tvoid`, which is able to capture all timelslots of a given frequency.

`gsmdecode` is another widely used AirProbe application, mainly used to interpret the GSM messages from the Gammu trace log and the outputs from both `gsm-receiver` and `gsm-tvoid`. `gsmdecode` basically converts hexadecimal bytes from GSM layer 2 and higher to human-readable format.

Two combinations of the software were used; `gsm-tvoid`+`gsmdecode` and `gsm-receiver`+`gsmdecode`:

gsm-tvoid+gsmdecode

`gsm-tvoid` with its `gsm_scan.py` is able to capture all timeslots of a given frequency. The following command is used to start intercepting the air interface:

```
./gsm_scan.py -p d -c 5 > scan_results
```

where the parameters have the following meaning:

- -p to choose what to print to the console, (d)ecoded hex for gsmdecode
- -c to choose which channel to capture, 5 in this case

The output is stored in the file `scan_results`. This file can be interpreted by `gsmdecode` with the following command:

```
./gsmdecode -X < scan_results
```

where X can be

- b for Format B (e.g. raw hex SDCCH)
- i for Format Bbis (e.g. raw hex BCCH)

Note that the USRP synchronizes itself to the BTS using the FCCH and is therefore only able to capture the C0 downlink. Recall also from table 5.1 that the observed Network Norway BTS only asked the MS to change timeslot while receiving SMS. This would indicate it should be possible to capture SMSs with `gsm-tvoid`, since `gsm_scan.py` captures all timeslots of the given frequency. However, this was not pursued extensively due to ethical considerations; it would be unwise and illegal to read other people's SMS messages.

gsm-receiver+gsmdecode

`gsm-receiver` provides two shell scripts that implement all necessary functions to capture the signals on a frequency, including interpretation of these signals. By typing

```
capture.sh 936.0M [duration==10] [decim==112] [gain==52]
```

it starts capturing the signals on timeslot 0 at the given frequency, in this case 936.0 MHz, and saving it to a file. It essentially recombines the captured GSM bursts into MAC blocks. The MAC blocks are displayed in 23 byte blocks and use 2B as padding if there is not enough data to fill a single block. The duration, decimation and gain in the above statement are optional arguments with default values. The file `capture_936.0M_112.cfile` contains the captured samples. These samples can then be interpreted by calling:

```
go.sh capture_936.0M_112.cfile [decim==112]
```

The `go.sh` script calls the `gsm-receiver.py` file which filters the information bits out of the samples. The result is a series of hexadecimal values that display the information sent out by the GSM network. By taking these values and run them through `gsm-decode`, it is possible to decode and view the data in a human-readable format. `gsm-receiver` and `gsm-decode` can also be run manually on the `.cfile`:

```
./gsm_receive.py -d 112 -I capture_936.0M_112.cfile -O
    output_file
```

where

- -d specifies the USRP decimation rate to be 112
- -I specifies the input file
- -O specifies the output file

Then `gsmdecode` needs to be run to translate into human-readable text:

```
./gsmdecode -X < output_file
```

where X can be

- b for Format B (e.g. raw hex SDCCH)
- i for Format Bbis (e.g. raw hex BCCH)

6.3 Results

The results from this experiment are mainly the outputs from `gsmdecode`, of which many greatly coincide with the listings in section 5.5.2. They thus also serve as confirmation of the results from the experiment in chapter 5. The main difference between the two experiments is that while the Nokia is only able to view "its own" traffic, the USRP can capture downlink traffic to any MS connected to the specific BTS. Listings 6.1-6.5 show a selection of the messages the USRP captured. Appendix H explains the LAPDm frame structure, format of fields and procedures needed to understand these messages. Each listing is built up with the raw hex-values on the top, and the decoded, human-readable interpretation, below.

The *System Information* messages in listings 6.3-6.5 were also seen by the Nokia, but in that experiment NetMonitor had already provided much of the same information. They are however included here because if someone aims at performing an attack without access to NetMonitor, these SI messages may provide vital information about the networks.

Other messages, such as *Immediate Assignment*, were also encountered, but are not included here because they would be redundant to the listings in section 5.5.2.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 25 06 21 00 05 f4 13 6d - f3 36 2b 2b 2b 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 25 001001-- Pseudo Length: 9
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 21 00100001 Paging Request Type 1
    3: 00 ------00 Page Mode: Normal paging
    5: f4 -----100 Type of identity: TMSI/P-TMSI
    6: 13 -------- ID(4/even): 136DF336
```

Listing 6.1: *RR Paging Request* message similar to listing 5.1. The MS is being paged with the TMSI. Listing 6.2 is an example of another *RR Paging Request* using IMSI to identify the MS.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 31 06 21 00 08 29 24 20 - 09 20 67 97 47 2b 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 31 001100-- Pseudo Length: 12
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 21 00100001 Paging Request Type 1
    3: 00 ------00 Page Mode: Normal paging
    5: 29 -----001 Type of identity: IMSI
    6: 24 -------- ID(7/odd): 24202XXXXXXXXX
```

Listing 6.2: *RR Paging Request* message similar to listing 5.2. The MS is being paged with the IMSI. The MSIN of the IMSI has again been censored with "X" because it uniquely identifies some mobile subscriber.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 59 06 1a 00 08 00 00 00 - 00 00 00 00 00 01 02 92
001: 90 a4 8a ff a5 00 00
    0: 59 010110-- Pseudo Length: 22
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 1a 00011010 RRsystemInfo2
    3: 00 ---x---- BCCH alloc. seq. num: 0
    3: 00 00------ Bitmap 0 format
    4: 08 ----1--- BCCH Allocation : ARFCN 116
   13: 01 -------1 BCCH Allocation : ARFCN 41
   14: 02 ------1- BCCH Allocation : ARFCN 34
   15: 92 1------- BCCH Allocation : ARFCN 32
   15: 92 ---1---- BCCH Allocation : ARFCN 29
   15: 92 ------1- BCCH Allocation : ARFCN 26
   16: 90 1------- BCCH Allocation : ARFCN 24
   16: 90 ---1---- BCCH Allocation : ARFCN 21
   17: a4 1------- BCCH Allocation : ARFCN 16
   17: a4 --1----- BCCH Allocation : ARFCN 14
   17: a4 -----1-- BCCH Allocation : ARFCN 11
   18: 8a 1------- BCCH Allocation : ARFCN 8
   18: 8a ----1--- BCCH Allocation : ARFCN 4
   18: 8a ------1- BCCH Allocation : ARFCN 2
   19: ff 1------- BCCH carrier with NCC = 7 is permitted for monitoring
   19: ff -1------ BCCH carrier with NCC = 6 is permitted for monitoring
   19: ff --1----- BCCH carrier with NCC = 5 is permitted for monitoring
   19: ff ---1---- BCCH carrier with NCC = 4 is permitted for monitoring
   19: ff ----1--- BCCH carrier with NCC = 3 is permitted for monitoring
   19: ff -----1-- BCCH carrier with NCC = 2 is permitted for monitoring
   19: ff ------1- BCCH carrier with NCC = 1 is permitted for monitoring
   19: ff -------1 BCCH carrier with NCC = 0 is permitted for monitoring
   20: a5 10------ Max. of retransmiss : 4
   20: a5 --1001-- slots to spread TX : 12
   20: a5 ------0- The cell is barred : no
   20: a5 -------1 Cell reestabl.i.cell: not allowed
   21: 00 -----0-- Emergency call EC 10: allowed
   21: 00 00000--- Acc ctrl cl 11-15: 0 = permitted, 1 = forbidden
   21: 00 ------00 Acc ctrl cl 8- 9: 0 = permitted, 1 = forbidden
   21: 00 -------0 Ordinary subscribers (8)
   21: 00 ------0- Ordinary subscribers (9)
   21: 00 -----0-- Emergency call (10): Everyone
   21: 00 ----0--- Operator Specific (11)
   21: 00 ---0---- Security service (12)
   21: 00 --0----- Public service (13)
   21: 00 -0------ Emergency service (14)
   21: 00 0------- Network Operator (15)
   22: 00 00000000 Acc ctrl cl 0- 7: 0 = permitted, 1 = forbidden
   22: 00 00000000 Ordinary subscribers (0-7)
```

Listing 6.3: *System Information 2* message containing information about neighboring cells, access rights, and NCCs

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 49 06 1b 41 e9 42 f2 20 - 01 31 c8 02 28 64 85 00
001: a5 00 00 3c bb 2b 2b
    0: 49 010010-- Pseudo Length: 18
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 1b 00011011 RRsystemInfo3C
    3: 41 16873 [0x41e9] Cell identity
    5: 42 242 Mobile Country Code (Norway)
    6: f2 02f Mobile Network Code (Netcom GSM AS)
    8: 01 305 [0x0131] Local Area Code
   10: c8 1------- Spare bit (should be 0)
   10: c8 -1------ MSs in the cell shall apply IMSI attach/detach procedure
   10: c8 --001--- Number of blocks: 1
   10: c8 -----000 1 basic physical channel for CCCH, not combined with SDCCHs
   11: 02 00000--- spare bits (should be 0)
   11: 02 -----010 4 multi frames period for paging request
   12: 28 00101000 T3212 TimeOut value: 40
   13: 64 0------- spare bit (should be 0)
   13: 64 -1------ Power control indicator is set
   13: 64 --10---- MSs shall not use uplink DTX
   13: 64 ----0100 Radio Link Timeout: 20
   14: 85 100----- Cell Reselect Hyst. : 8 db RXLEV
   14: 85 ---xxxxx Max Tx power level: 5
   15: 00 0------- No additional cells in SysInfo 7-8
   15: 00 -0------ New establishm cause: not supported
   15: 00 --xxxxxx RXLEV Access Min permitted = -110 + 0dB
   16: a5 10------ Max. of retransmiss : 4
   16: a5 --1001-- slots to spread TX : 12
   16: a5 ------0- The cell is barred : no
   16: a5 -------1 Cell reestabl.i.cell: not allowed
   17: 00 -----0-- Emergency call EC 10: allowed
   17: 00 00000--- Acc ctrl cl 11-15: 0 = permitted, 1 = forbidden
   17: 00 ------00 Acc ctrl cl 8- 9: 0 = permitted, 1 = forbidden
   17: 00 -------0 Ordinary subscribers (8)
   17: 00 ------0- Ordinary subscribers (9)
   17: 00 -----0-- Emergency call (10): Everyone
   17: 00 ----0--- Operator Specific (11)
   17: 00 ---0---- Security service (12)
   17: 00 --0----- Public service (13)
   17: 00 -0------ Emergency service (14)
   17: 00 0------- Network Operator (15)
   18: 00 00000000 Acc ctrl cl 0- 7: 0 = permitted, 1 = forbidden
   18: 00 00000000 Ordinary subscribers (0-7)
   19: 3c YYYYYYYY REST OCTETS (2)
```

Listing 6.4: *System Information 3* message containing information that identifies the BTS and the location area. The organization of CCCH is also given.

```
HEX l2_data_out_Bbis:462 Format Bbis DATA
000: 31 06 1c 42 f2 20 01 31 - 85 00 a5 00 00 05 2b 2b
001: 2b 2b 2b 2b 2b 2b 2b
    0: 31 001100-- Pseudo Length: 12
    1: 06 0------- Direction: From originating site
    1: 06 -000---- 0 TransactionID
    1: 06 ----0110 Radio Resource Management
    2: 1c 00011100 RRsystemInfo4-C
    3: 42 242 Mobile Country Code (Norway)
    4: f2 02f Mobile Network Code (Netcom GSM AS)
    6: 01 305 [0x0131] Local Area Code
    8: 85 100----- Cell Reselect Hyst. : 8 db RXLEV
    8: 85 ---xxxxx Max Tx power level: 5
    9: 00 0------- No additional cells in SysInfo 7-8
    9: 00 -0------ New establishm cause: not supported
    9: 00 --xxxxxx RXLEV Access Min permitted = -110 + 0dB
   10: a5 10------ Max. of retransmiss : 4
   10: a5 --1001-- slots to spread TX : 12
   10: a5 ------0- The cell is barred : no
   10: a5 -------1 Cell reestabl.i.cell: not allowed
   11: 00 -----0-- Emergency call EC 10: allowed
   11: 00 00000--- Acc ctrl cl 11-15: 0 = permitted, 1 = forbidden
   11: 00 ------00 Acc ctrl cl 8- 9: 0 = permitted, 1 = forbidden
   11: 00 -------0 Ordinary subscribers (8)
   11: 00 ------0- Ordinary subscribers (9)
   11: 00 -----0-- Emergency call (10): Everyone
   11: 00 ----0--- Operator Specific (11)
   11: 00 ---0---- Security service (12)
   11: 00 --0----- Public service (13)
   11: 00 -0------ Emergency service (14)
   11: 00 0------- Network Operator (15)
   12: 00 00000000 Acc ctrl cl 0- 7: 0 = permitted, 1 = forbidden
   12: 00 00000000 Ordinary subscribers (0-7)
```

Listing 6.5: *System Information 4* message serves the purpose of repeating the information sent in previous System Information messages.

Chapter 7

Setting up a Rogue GSM Network

This chapter describes the feasibility experiment of building a (rogue) GSM network with open-source hardware and software.

The authentication process described in section 2.13 does not apply to the network side of GSM, that is, the BTS provides no authentication to the MS it is communicating with. Thus, it is possible for an attacker to set up a man-in-the-middle attack, also known as a rouge BTS, with the same MCC and MNC as the subscriber's network. If the rouge BTS has a higher signal strength than a legitimate BTS, it is able encourage the MSs under its radio coverage to get associated. The attacker can then choose to disable ciphering and frequency hopping, before routing the traffic to the destination site. By acting as a middle point between the MS and the legitimate network, it is relatively easy to intercept calls and SMS messages.

7.1 Laboratory Setup

This section describes the laboratory setup used for conducting the experiment. Permission to set up a BTS was acquired from the Norwegian Post and Telecommunications Authority. The equipment was placed in an anechoic chamber at NTNU Gløshaugen, Trondheim, Norway, consisting of the following hardware:

- HP Compaq 8000 Elite with Ubuntu, GNU Radio, OpenBTS and Asterisk
- A USRP with two RFX900 daughtercards and one VERT900 antenna
- Two Nokia 3310 with SIM cards

For an introduction to the USRP and GNU Radio, see sections 6.1.1 and 6.1.2, respectively.

7.1.1 OpenBTS

OpenBTS is an open-source software access point for GSM, implementing the air interface by utilizing the USRP and the GNU Radio framework. This application aims to replace the network-side in GSM, from the BTS and upwards. OpenBTS has fully integrated RR functions. In addition, it maps CC and MM procedures to SIP operations without the need for any supporting MSCs or VLRs. Adding the capabilities of Smqueue[1], SMS services are also provided. By using Asterisk[2] on the network back-end, OpenBTS allows MSs to be used directly as SIP endpoints.

The diagram in figure 7.1 shows a typical architecture of OpenBTS. The elements in the red box is software that OpenBTS provides. The actual

[1] A RFC-3428 store and forward server
[2] An open-source software implementation of a telephone private branch exchange (PBX).

network interface is Asterisk. MySQL is planned in future version for mobility management.

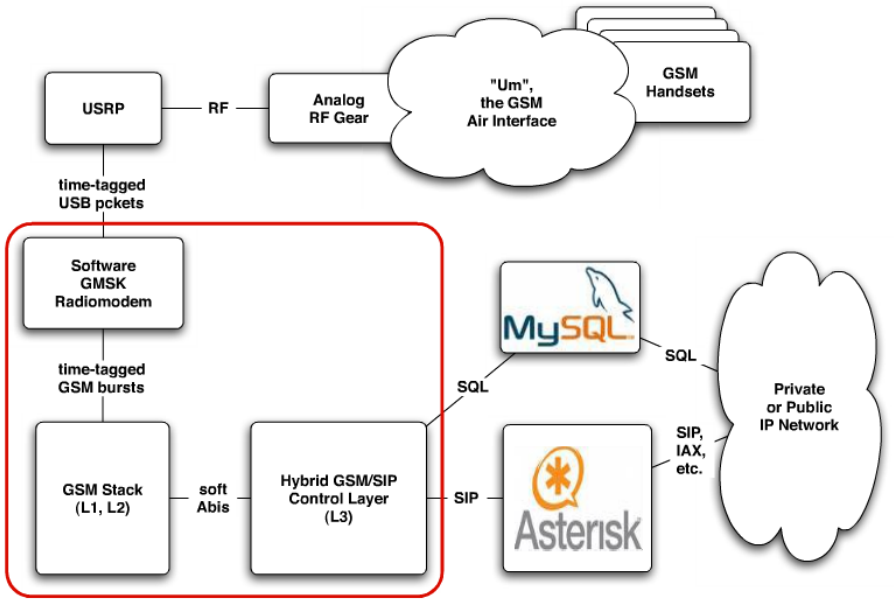

Figure 7.1: OpenBTS Architecture

The OpenBTS project was started by David A. Burgess and Harvind Samra. Their aim was to reduce the cost of GSM service provision to a 1/10 the cost of current technologies. A OpenBTS pilot site is now currently up and running in Niue, an island nation in the South Pacific Ocean. This is the first OpenBTS installation to provide common-carrier service to the general public [26].

7.2 Method

This section only summarizes the main commands and actions taken to set up a working GSM network. A complete installation and configuration guide can be found in appendix G.

USRP was booted up and Asterisk command line interface (CLI) was reached by typing

```
sudo /usr/sbin/asterisk -rvvvv
```

A new shell was opened and the following command was used to start smqueue

```
sudo ./smqueue
```

Wireshark was set to capture traffic on the localhost, and OpenBTS was initialized by calling (in yet another shell):

```
./OpenBTS
```

Picture 7.2 illustrates the typical user interfaces when running a complete (rogue) GSM network. The OpenBTS terminal is shown to the upper left corner, whereas Asterisk is located to the upper right. Smqueue is placed in the bottom left corner and the familiar Wireshark is found to the bottom right.

- OpenBTS was used to view the signaling traffic on the air interface and read SMS messages.
- Asterisk was used to see the authentication process.
- Smqueue was used as a store and forward mechanism for SMS messages.
- Wireshark was used to eavesdrop speech traffic by entering "Statistics -> VoIP calls"

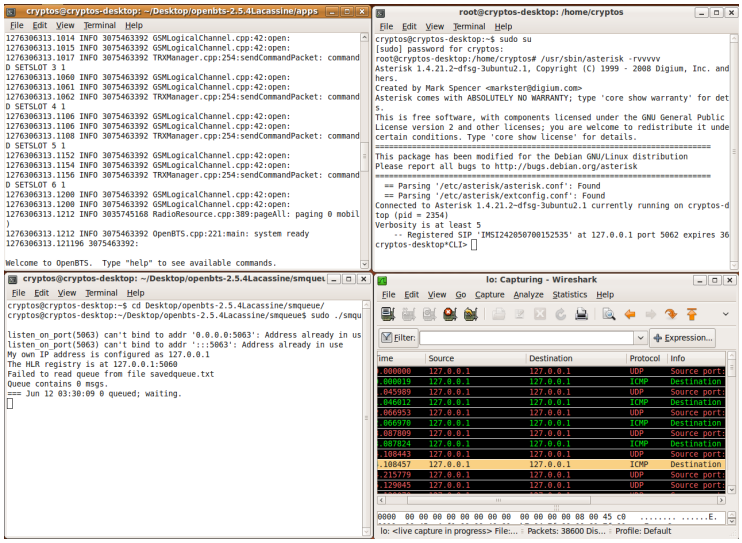

Figure 7.2: Four User Interfaces for Managing a GSM Network

Our preferred testing phones were two Nokia 3310. To ensure that these mobile phones would register to our network, we changed the "select network" setting in the phones' menu from automatic to manual. This provided us with a list of all currently available networks in the area, including our test network (001 01), as shown in picture 7.3. Other GSM phones could also be used. However, some phones had difficulties trying to register to OpenBTS, and this is mainly related to a frequency accuracy problem in the USRP[58].

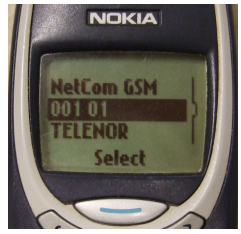

Figure 7.3: Available GSM Networks in the Lab Environment

In order to keep the power level to a minimum, the mobile phones were held at a short distance (of maximum 2 meters) away from the USRP. Once a successful connection was made, the mobile phones could interact and communicate as normal. However, as the experiment took place in a closed environment, the phones were limited to calls and SMS transfer only to other registered users of our BTS.

We explored various scenarios, such as MOC, MTC, MO-SMS and MT-SMS. These scenarios involved active connections between two registered users and a man-in-the-middle performing the actual interception. During the entire experiment, encryption and frequency hopping were turned off.

7.3 Results

The purpose of this experiment was to give a practical demonstration of how a GSM network could be built by applying open-source hardware and software. Having our own GSM network allows us to control and monitor the traffic going in and out of our local host. Our results indicate that this approach is fully working and may be applied to real-world scenarios to perform an active attack.

Even though an attack of this sort has been performed in the past, the equipment used was either extremely expensive or not available for the public. In this experiment, all the necessary equipment could be bought in for less than 1500$ and no more than a few hours of configuration.

By deploying the rogue GSM network close to our intended target, we could intercept the victim's voice and signaling traffic. Because the traffic is routed through our machine, Wireshark's built-in SIP analyzer, as shown in picture 7.4, allows us to play back voice traffic in real-time or to be recorded for later use. Alternatively, the Asterisk's Monitor feature could be activated, which record

entire conversations as audio files.

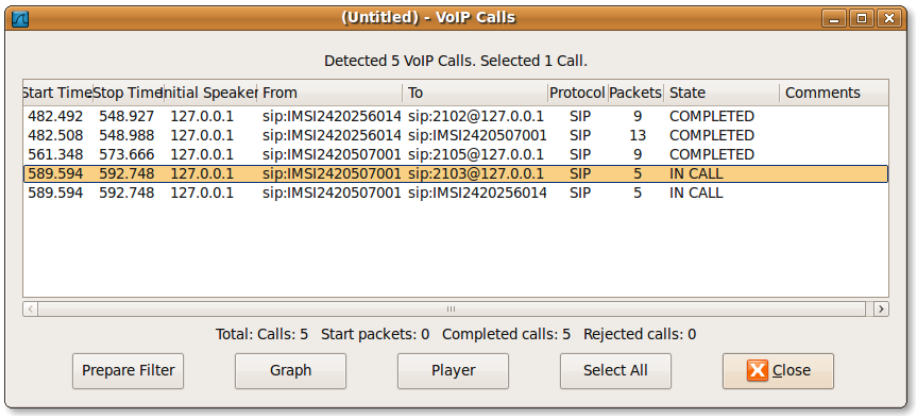

Figure 7.4: Recorded conversations in Wireshark

During MOC and MTC, NetMonitor provided additional information on the current connection. Picture 7.5 confirms that neither ciphering nor any frequency hopping were enabled during calls.

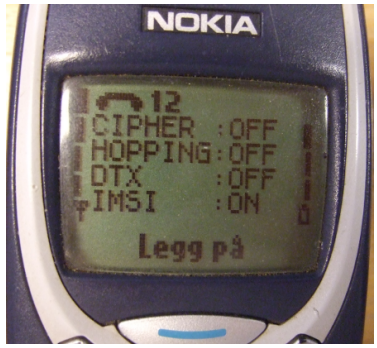

Figure 7.5: Encryption and Frequency Hopping Disabled

In addition, SMS messages were displayed in clear-text in the OpenBTS shell. An example of this is shown in listing 7.4.

The remainder of this section gives an overview of the various interactions between the phones and our rogue GSM network. An understanding of the GSM protocol stack and basic SIP procedures are helpful when reading through the following listings.

```
ChannelDescription=(typeAndOffset=SDCCH/4-0 TN=0 TSC=0 ARFCN=986) RequestReference=(RA=29 T1
      '=17 T2=14 T3=15) TimingAdvance=1
MM Location Updating Request LAI=(MCC=001 MNC=01 LAC=0x29a) MobileIdentity=(TMSI=0x4c0c008c)
MM Location Updating Request LAI=(MCC=001 MNC=01 LAC=0x29a) MobileIdentity=(TMSI=0x4c0c008c)
L3 SAP0 sending MM Identity Request type=IMSI
L3 recv MM Identity Response mobile id=IMSI=242050700152535
registration SUCCESS: IMSI=242050700152535
L3 SAP0 sending MM MM Information short name=(Cryptos)
L3 SAP0 sending MM Location Updating Accept LAI=(MCC=001 MNC=01 LAC=0x29a)ID=(TMSI=0
      x4c0c139b)
L3 SAP0 sending RR Channel Release cause=0x0
```

Listing 7.1: This listing shows a successful IMSI attach on our GSM network. The network sends out an Identity Request, as it does not recognize the TMSI transmitted from the MS. Hence, the MS replies with its IMSI in an Identity Response message. It is also clearly shown that we are a test network in a test country (MCC = 001 and MNC = 01) using channel combination V at ARFCN 986. A MS should never connect to it by default.

By typing 'tmsis' in the OpenBTS shell, an overview of the connected phones is given with the corresponding TMSIs allocated by our network. Picture 7.6 illustrates the mapping between the IMSI and TMSI for our two mobile phones.

```
tmsis
 IMSI             TMSI
242050700152535 0x4c139f6f
242025601404061 0x4c139f70

OpenBTS>
```

Figure 7.6: Mapping between IMSI and TMSI

```
MM CM Service Request serviceType=MOC mobileIdentity=(TMSI=0x4c139f70)
MOC: MM CM Service Request serviceType=MOC mobileIdentity=(TMSI=0x4c139f70)
L3 SAP0 sending MM CM Service Accept
MOC: CC Setup TI=(0,0) CalledPartyBCDNumber=(type=unknown plan=E.164/ISDN digits=2102)
new transaction 1804289388 TI=(0,0) IMSI=242025601404061 MOC to=2102 Q.931State=MOC
SIP send INVITE IMSI242025601404061
creating SIP message FIFO callID 1208560147
write INVITE sip:2102@127.0.0.1 SIP/2.0
sending AssignmentCommand for 0x8288038 on 0xbfc90370
L3 SAP0 sending RR Assignment Command channelDescription=(typeAndOffset=TCH/F TN=2 TSC=0
     ARFCN=986) powerCommand=0 mode1=speech1
MOC: transaction: 1804289388 TI=(0,0) IMSI=242025601404061 MOC to=2102 Q.931State=MOC
     initiated SIPState=Starting
L3 SAP0 sending CC Call Proceeding TI=(1,0)
MOC A: wait for Ringing or OK
L3 SAP0 sending CC 0x3 TI=(1,0) prog_ind=(location=1 progress=0x0)
MOC A: SIP:Ringing, send Alerting and move on
L3 SAP0 sending CC Alerting TI=(1,0)
MOC: wait for SIP OKAY
SIP send Ringing IMSI242050700152535
write SIP/2.0 180 Ringing
SIP send Ack IMSI242025601404061
GSM Connect Acknowledge IMSI=242025601404061
MOC: sending Connect to handset
MOC MTC connected, IMSI=242025601404061 entering callManagementLoop
```

Listing 7.2: An excerpt of a outgoing call viewed from our OpenBTS shell. A TCH+FACCH is assigned immediately to the MS without using a SDCCH. The entire call setup is performed on the FACCH. Knowing that this call is performed, the speech traffic could be listen in on using Wireshark.

```
set up MTC paging for channel=TCH/F
INVITE to IMSI242050700152535
creating SIP message FIFO callID 1c493d7165aba6146e5047850ba5e0cc
new transaction 1804289390 TI=(1,0) IMSI=242050700152535 MTC from=IMSI24202560140 Q.931State
     =MTC paging
IMSI=242050700152535 added to table
paging 1 mobile(s)
RR Paging Response mobileID=(IMSI=242050700152535)
MTC on FACCH transaction: 1804289390 TI=(1,0) IMSI=242050700152535 MTC from=IMSI24202560140
     Q.931State=MTC paging
MTC: sending GSM Setup to call type=national plan=E.164/ISDN digits=IMSI24202560140
L3 SAP0 sending CC Setup TI=(0,0) CallingPartyBCDNumber=(type=national plan=E.164/ISDN
     digits=IMSI24202560140)
SIP send Trying IMSI242050700152535
write SIP/2.0 100 Trying
GSM Connect IMSI=242050700152535
MTC:: allocating port and sending SIP OKAY
SIP send INVITE-OK IMSI242050700152535
write SIP/2.0 200 OK
read ACK sip:IMSI242050700152535@127.0.0.1:5062 SIP/2.0
read SIP/2.0 200 OK
L3 SAP0 sending RR Channel Mode Modify description=(typeAndOffset=TCH/F TN=3 TSC=0 ARFCN
     =986) mode=(speech1)
MTC:: waiting for GSM Alerting and Connect
GSM Alerting IMSI=242050700152535
SIP send Ringing IMSI242050700152535
write SIP/2.0 180 Ringing
received sip_method=INVITE
read SIP/2.0 180 Ringing
L3 SAP0 sending CC Connect TI=(1,0)
L3 SAP0 sending CC Connect Acknowledge TI=(0,0)
MOC MTC connected, IMSI=242050700152535 entering callManagementLoop
```

Listing 7.3: An excerpt of a incoming call viewed from our OpenBTS shell.

```
MM CM Service Request serviceType=SMS mobileIdentity=(TMSI=0x4c139f70)
L3 SAP0 sending MM CM Service Accept
CPData CP-DATA TI=(0,7) RPDU=(00030006917429000100241145048112200000
     a71dcd3c685e1ecbcb74103c3c9fdfdf7232283d0785e5f3b23bcc06)
SMS RP-DATA 0 ref=1 origSMSC=(type=unknown plan=unknown digits=) destSMSC=(type=
     international plan=E.164/ISDN digits=4792001000) TPDU=(primitive=undefined data
     =(1145048112200000a71dcd3c685e1ecbcb74103c3c9fdfdf7232283d0785e5f3b23bcc06))
SMS SMS-SUBMIT 1 RD=0 VPF=2 RP=0 UDHI=0 SRR=0 MR=69 DA=(type=unknown plan=E.164/ISDN digits
     =2102) PI=0 DCS=0 VP=(expiration=(Sun Jun 14 16:59:50 2010)) UD="My secret password is
     arsenal"
from IMSI=242025601404061 mesage: 1 RD=0 VPF=2 RP=0 UDHI=0 SRR=0 MR=69 DA=(type=unknown plan
     =E.164/ISDN digits=2102) PI=0 DCS=0 VP=(expiration=(Sun Jun 14 16:59:50 2010)) UD="My
     secret password is arsenal"
TI=(0,0) IMSI=242025601404061 SMS to=2102 Q.931State=SMS submission
SIP send to 2102@127.0.0.1 MESSAGE My secret password is arsenal
creating SIP message FIFO callID 852050533
write MESSAGE sip:2102@127.0.0.1 SIP/2.0
read SIP/2.0 202 Queued
successful
removing SIP message FIFO callID 852050533
sending RPAck in CPData
CPAck CP-ACK TI=(0,7)
closing
L3 SAP0 sending RR Channel Release cause=0x0
```

Listing 7.4: An excerpt of the SMS message sent to one of our phones - viewed from our OpenBTS shell. The actual SMS is in the RP-DATA message. Notice that our interceptor easily reads the content of the SMS message in clear text.

```
set up MTC paging for channel=SDCCH
MESSAGE to IMSI242050700152535
new transaction 1804289385 TI=(1,0) IMSI=242050700152535 MTSMS Q.931State=MTC paging message
     ="My secret password is arsenal"
paging 1 mobile(s)
RA=0x14 when=0:1869128 age=24 TOA=1.0000
RR Paging Response mobileID=(IMSI=242050700152535)
MTSMS: transaction: 1804289385 TI=(1,0) IMSI=242050700152535 MTSMS Q.931State=MTC paging
     message="My secret password is arsenal"
MTSMS: sending CPAck
MTSMS: closing
L3 SAP0 sending RR Channel Release cause=0x0
```

Listing 7.5: An excerpt of the same SMS message received by the other phone.

Chapter 8

Discussion

Our tables

As pointed out in section 4.3, we have a total coverage of 19.13% on 38 tables. This is must be considered to be a serious breach in the security of A5/1, and should be a warning to both GSMA and the public that A5/1 cannot be considered to be secure anymore.

The obvious way to improve this coverage of the tables is to generate more of them. Given todays hard drive prices, such an option would by no means be a expensive solution. Another way to improve coverage, is by having even more known plaintext. The above coverage percent is when 114 bits of known plaintext is known, but it is possible to get an ever higher coverage if we are in possession of even more known plaintext. The last way to improve coverage is of course by changing the table format or parameters. Given an in depth analysis on this is without the scope of this thesis and will not be done. It would require an extensive amount of theoretical analysis and experimental verification, and is therefor left as future work. However, an interesting approach that should be

researched further is to take keystream samples of less than 64-bits. This would lead to more false positives, but will at the same time give more samples to do lookup on.

Known plaintext needed for the tables

From our decoded signaling traffic in 5.5.2, LAPDm frames occurs at predictable periods of time. Some of these frames are filled with so-called filling bits (or padding), which offers samples for a known-plaintext attack against A5/1. A possible source of known plaintext may be found in the *Cipher Mode Complete* message. This is transmitted in an uplink frame and interleaved over four consecutive bursts. However, our research in chapter 5 suggests that the Norwegian networks include the IMEISV in this message. Taking this into account, the message could be less attractive as known plaintext than previously thought. Regarding networks not including the IMEISV, this message still provides large quantities of filling bits.

As the *Cipher Mode Complete* message is sent uplink, it is difficult to capture, and focus should be directed towards downlink messages. Particularly interesting downlink sources of known plaintext may be the System Information 5 and System Information 6 messages. These frames are sent on the SACCH in a repeating pattern, similarly for MOC and MTC. Further analysis in Wireshark revealed several potential sources of known plaintext, summarized below.

For MTC:

- Acknowledge to the *Assignment Complete* message
- Acknowledge to the *Alerting* message
- The *Connect Acknowledge* message

For MOC:

- Acknowledge to the *Cipher Mode Complete* message

- The *Call Proceeding* message
- The *Alerting* message
- The *Connect* message

Future work should investigate the feasibility of using the mentioned messages as known plaintext, as well as determine other sources.

Improvements to AirProbe

As the rainbow tables approach fully generated, the focus should be turned to capture data from the air interface. The AirProbe project is a likely candidate for this work, but it needs significant improvements before it can perform satisfactory. Firstly, the software should be made better at decoding the downlink bursts, as it still produces unexpected anomalies. This might also require changes in GNU Radio and/or the USRP hardware, depending on where the reception problems actually exist. Secondly, support for intercepting uplink traffic should be added. However, as the USRP requires fairly strong signal strengths to be able to decode correctly, uplink capture remains an unresolved issue. Another big challenge is to passively capture traffic if frequency hopping is employed, which based on figure 5.1 seems to be fairly common in use. Frequency hopping was originally implemented to minimize interference, although its presence proves to be a real issue for both AirProbe and the USRP. In some sense it has evolved to a security feature. Having said that, the frequency hopping sequence may still be transmitted in clear-text by the BTS as shown in listings 5.4 and 5.10. Therefore, if frequency hopping is activated in a cell, the attacker needs to:

1. Capture the entire frequency band of interest and find the hopping sequence pattern later; or
2. Configure the USRP to follow the frequency hopping sequence in real-time.

The first approach seems to be supported by the majority of the AirProbe community, as it provides the highest probability of success. The drawback is that it requires the capture and transfer of huge amount of data. If AirProbe solves the current compatibility issues with the USRP2, it should be feasible to record the entire GSM band in one direction. Currently, no AirProbe software can capture multiple channels.

The second approach is by far the most elegant solution. However, the margin of failure is extremely narrow, as the USRP has to process the hopping sequence relatively fast without loosing synchronization. In some scenarios, the hopping sequences could be exchanged during ciphered mode, meaning that the encryption must be cracked within milliseconds. A moment's thought shows that this is not feasible.

Recall from table 5.1 that not all cells activate frequency hopping, and the first succeeding practical attacks will typically be directed against these.

Having said that, AirProbe must implement speech decoding in order to make a passive attack truly feasible. Given the complex schemes of channel coding and interleaving of speech traffic, this will require significant amount of work.

The applications of OpenBTS

OpenBTS in all its complexity provides us with a working GSM stack fully operational with the USRP. It is an effective learning tool, ideal for both students and hobbyists who wants to learn and understand the fundamentals of the GSM protocols close-up. However, be aware of any legal constraints before running OpenBTS in a GSM band. It might interfere with the services of a local operator or even worse; an unsuspecting user could register on your BTS trying to make an emergency call that you can not connect. Doing so would violate a number of civil and criminal laws. The safest way to run OpenBTS is in a closed environment, with all signals confined in a Faraday cage.

Originally designed for setting up low-cost networks, OpenBTS could also be exploited to act as a rogue GSM network. From our controlled experiment in chapter 7, we eavesdropped conversations, monitored the signaling traffic and intercepted SMS messages. Alternately, our rogue GSM network could have been set up as a relay and route traffic through a local VoIP gateway to connect calls and SMS to the "outside" world. However, this would have been highly illegal, and was thus not tested.

To create and set up a rogue GSM network in a real-world scenario, all an attacker needs to do is broadcast the appropriate MCC and MNC of a local PLMN and emit a stronger signal than any other BTSs in the area. Having said that, the process of building and implementing a successful rogue GSM network is still far from plug-and-play, as appendix G shows.

Advice to users

Users wanting to be as secure as possible while using GSM should choose a network operator that enables frequency hopping on MOCs, MTCs and SMS messages. This project's limited survey on the Norwegian networks in chapter 5 revealed that Network Norway only changes timeslot when delivering SMS messages on the examined BTS. Some AirProbe software is already capable of capturing all timeslots of C0, and it should thus be an achievable task to intercept these SMS messages. Further, NetCom and Network Norway did not hop during calls on all BTSs. Telenor is the only operator utilizing frequency hopping in all the scenarios.

Another interesting question is whether SMS messages are encrypted or not. When the authors of this thesis asked the various operators, Telenor was the only company to admit that they were *not* encrypted. This fact is particularly surprising, given the vast amount of applications and services that today offer secure login through SMS. It seems to be optional for the operator to encrypt SMS, because we have reports of it being encrypted in Germany.

Advice to operators and equipment manufacturers

Even though frequency hopping is "security by obscurity" it should be enabled for the scenarios like MOC, MTC, MO-SMS and MT-SMS. In the longer term the introduction of A5/3 as an alternative ciphering algorithm could mitigate utilization of the produced tables. However, all old equipment would still be vulnerable as it would fall back to A5/1 if not both endpoints support A5/3. Another important factor would be to randomize the filling bits found in the LAPDm frames in order to reduce the amount of known plaintext. Also, if SMS messages are transmitted in clear-text, as stated by Telenor, they are in constant exposure to interception attacks. Thus, it is in the interest of all to implement secure transmission mechanisms of these messages.

In addition, the MS should provide the user with information about the encryption scheme applied when connected to a GSM network. By displaying an open padlock if encryption is disabled, it could prove valuable to prevent or detect rogue networks. According to one of the GSM specifications [33], whenever a connection is in place, which is, or becomes unencrypted, an indication shall be given to the user. However, this alert is unfortunately seldom displayed, as it also states; the encryption indicator feature may be disabled in the SIM by the home network operator.

The issues with hardware and software

An extensive amount of work was put into configuring, installing and troubleshooting both hardware and software. Software installations were often related to compatibility issues with library versions or other configuration problems. The software side is thus supported by a number of tutorials found in the appendix, ranging from Gammu to OpenBTS.

Chapter 9

Conclusion

In collaboration with Frank Stevenson and Karsten Nohl we have generated 1896 gigabyte of rainbow tables that given 114-bits of known plaintext have more than 19% probability of decrypting a GSM conversation.

Several sources of where to find the needed known plaintext were discovered. However, capturing GSM traffic is still troublesome, with frequency hopping currently being the main obstacle. Improvements to AirProbe have been suggested, and the authors expect the process of capturing multiple channels and decoding speech traffic will become feasible in the near future.

From our controlled experiment, OpenBTS was exploited to perform an active attack. Conversations were tapped and signaling traffic, including SMS messages, were viewed in cleartext. It is no longer the case that GSM interception is limited only to government agencies. Today, an average hacker can purchase hardware equipment for less than 1500$ to capture raw GSM traffic or to perform an active attack. An awareness is thus raised by pointing the way to a more secure future for the users, operators and equipment manufacturers.

Bibliography

[1] Airprobe. https://svn.berlin.ccc.de/projects/airprobe/. Last accessed June 11, 2010.

[2] COPACOBANA - A Codebreaker for DES and other Ciphers. http://www.copacobana.org/. Last accessed June 11, 2010.

[3] GNU Radio. http://gnuradio.org/redmine/wiki/gnuradio. Last accessed June 9, 2010.

[4] GNU Radio Ubuntu Build Instructions. http://gnuradio.org/redmine/wiki/gnuradio/UbuntuInstall. Last accessed June 10, 2010.

[5] GSM decoding with Nokia 3310 phone. https://svn.berlin.ccc.de/projects/airprobe/wiki/tracelog. Last accessed May 05, 2010.

[6] Working with the USRP. https://svn.berlin.ccc.de/projects/airprobe/wiki/WorkingWithTheUSRP. Last accessed May 29, 2010.

[7] *Nokia NetMonitor Manual*, 0.95 edition, November 2002. http://www.nobbi.com/download/nmmanual.pdf, last accessed June 8, 2010.

[8] Gammu:Compiling/installing in Linux. `http://www.gammu.org/wiki/index.php?title=Gammu:Compiling/installing_in_Linux`, August 2007. Last accessed May 29, 2010.

[9] The GSM Sniffer Project. `http://web.archive.org/web/20071018030844/http://wiki.thc.org/gsm/`, 2007.

[10] A5/1 Security Project. `http://reflextor.com/trac/a51`, May 2010. Last accessed June 11, 2010.

[11] GSM for Dummies. `http://students.ee.sun.ac.za/~gshmaritz/gsmfordummies/intro.shtml`, February 2010. Mirror for www.gsmfordummies.com.

[12] Index of /torrents. `http://reflextor.com/torrents/`, February 2010.

[13] 3rd Generation Partnership Project. Radio Access Network; Radio transmission and reception. `http://www.3gpp.org/ftp/Specs/html-info/45005.htm`.

[14] GSM Association. GSM World - Home of the GSM Association. `http://www.gsmworld.com/`, June 2010. Last accessed June 11, 2010.

[15] Jan A. Audestad. *Technologies and Systems for Access and Transport Networks*. Artech House, 2008.

[16] S. Babbage. A Space/Time Tradeoff in Exhaustive Search Attacks on Stream Ciphers. In *European Convention on Security and Detection*, number 408 in IEE Conference Pulication, May 1995.

[17] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. 2007.

[18] Eli Biham and Orr Dunkelman. *Cryptanalysis of the A5/1 GSM Stream Cipher*, volume 1977/2000 of *Lecture Notes in Computer Science*, pages 43–51. Springer Berlin / Heidelberg, January 2000. `http://www.springerlink.`

com/content/gugp3fwpg2f2ecyh/fulltext.pdf, last accessed March 08, 2010.

[19] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *ASIACRYPT 2000*, pages 1–13. Springer-Verlag Berlin Heidelberg, 2000.

[20] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. 2001. http://www.springerlink.com/content/hjytym5f7pnx6wna/fulltext.pdf, Last accessed March 02, 2010.

[21] Eric Blossom. Exploring GNU Radio. http://www.gnu.org/software/gnuradio/doc/exploring-gnuradio.html, November 2004. Last accessed May 19, 2010.

[22] Johan Borst, Bart Preneel, and Joos Vandewalle. On the Time-Memory Tradeoff Between Exhaustive Key Search and Table Precomputation. In *Proc. of the 19th Symposium in Information Theory in the Benelux, WIC*, pages 111–118, 1998.

[23] Noureddine Boudriga. *Security of Mobile Communication.* Auerbach Publications, 2009.

[24] Alex Brand and Hamid Aghvami. *Multiple Access Protocols for Mobile Communications: GPRS, UMTS and Beyond.* Wiley, 2002.

[25] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of A5/1. http://scard.org/gsm/a51.html/, 1998. Last accessed June 11, 2010.

[26] David A. Burgess. The OpenBTS Chronicles. http://openbts.blogspot.com/. Last accessed June 3, 2010.

[27] Praphul Chandra. *Bulletproof Wireless Security - GSM, UMTS, 802.11, And Ad Hoc Security.* Elsevier Inc, 2005.

[28] GNU Radio Community. Um interface. `http://en.wikipedia.org/wiki/Um_interface`. Last accessed May 29, 2010.

[29] D.E. Denning. *Cryptography and Data Security*, chapter 2, page 100. ACM Classic Books Series. Addison-Wesley, 1982.

[30] Jörg Eberspächer, Hans-Joerg Vögel, Christian Bettstetter, and Christian Hartmann. *GSM – Architecture, Protocols and Services, 3rd edition.* John Wiley & Sons Ltd, 2009.

[31] ETSI. Digital cellular telecommunications system (Phase 2+); Mobile radio interface layer 3 specification (GSM 04.08 version 8.0.0 Release 1999), July 1999.

[32] ETSI. Digital cellular telecommunications system (Phase 2+); Mobile radio interface signalling layer 3; General aspects (GSM 04.07 version 7.3.0 Release 1998), December 1999.

[33] ETSI. Digital cellular telecommunications system (Phase 2+); Mobile Stations (MS) features (GSM 02.07 version 8.0.0 Release 1999), June 1999.

[34] ETSI. Digital cellular telecommunications system (Phase 2+); Point-to-Point (PP) Short Message Service (SMS) Support on Mobile Radio Interface (GSM 04.11 version 7.0.0 Release 1998), August 1999.

[35] ETSI. Digital cellular telecommunications system (Phase 2); Security related network functions (GSM 03.20 version 8.0.0 Release 1999), November 1999.

[36] ETSI. Digital cellular telecommunications system (Phase 2+); Abbreviations and acronyms (GSM 01.04 version 8.0.0 release 1999), May 2000.

[37] ETSI. Digital cellular telecommunications system (Phase 2+); Data Link (DL) layer; General aspects (GSM 04.05 version 8.0.1 Release 1999), September 2000.

[38] ETSI. Digital cellular telecommunications system (Phase 2+); Mobile Station - Base Stations System (MS - BSS) interface Data Link (DL) layer specification (GSM 04.06 version 8.0.1 Release 1999), September 2000.

[39] ETSI. Digital cellular telecommunications system (Phase 2+); Multiplexing and multiple access on the radio path (GSM 05.02 version 8.5.1 Release 1999), November 2000.

[40] ETSI. Digital cellular telecommunications system (Phase 2+); Physical Layer on the Radio Path (General Description) (GSM 05.01 version 8.4.0 Release 1999), July 2000.

[41] ETSI. Digital cellular telecommunications system (Phase 2+); Radio subsystem synchronization (GSM 05.10 version 8.2.0 Release 1999), July 2000.

[42] ETSI. Digital cellular telecommunications system (Phase 2+); Mobile Station - Base Station System (MS - BSS) interface; Channel structures and access capabilities (GSM 04.03 version 8.0.1 Release 1999), September 2001.

[43] ETSI. Digital cellular telecommunications system (Phase 2+); Security aspects (GSM 02.09 version 8.0.1 Release 1999), June 2001.

[44] Robert Fitzsimons. Find a GSM base station manually using a USRP. `http://273k.net/gsm/find-a-gsm-base-station-manually-using-a-usrp/`, April 2007. Last accessed February 1st, 2010.

[45] El Gambo. *Nokia NetMonitor Manual*. http://www.mob385.com, 2.1a edition. `http://www.mob385.com/download/netmonitor.pdf`, last accessed June 8, 2010.

[46] Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In *Advances in Cryptology - EUROCRYPT ´97*, volume 1233 of *Lecture Notes in Computer Science*, pages 239 – 255. Springer Berlin / Heidelberg, 1997.

[47] Google. Google Wave Overview. `http://wave.google.com/about.html`, 2010. Last accessed May 11, 2010.

[48] Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, IT-26(4):401–406, July 1980.

[49] David Hulton. The A5 Cracking Project. `http://events.ccc.de/camp/2007/Fahrplan/events/2015.en.html`, August 2007. Last accessed June 11, 2010.

[50] ITU-T. Recommendation E.164, The International Public Telecommunication Numbering Plan. `http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-E.164-200502-I!!PDF-E&type=items`, February 2005. Last accessed May 25, 2010.

[51] Jari P. Jokinen. *Field Test Display Specification.* Nokia, 1.0 edition, April 2004. `http://www.bakx.pl/download/netmonitor_guide_official_nokia_6630.zip`, last accessed June 8, 2010.

[52] Inc. Kestrel Signal Processing. The OpenBTS Project. `http://openbts.sourceforge.net/`. Last accessed June 5, 2010.

[53] Ettus Research LLC. The USRP. `http://www.ettus.com`. Last accessed May 1, 2010.

[54] Alexsander Loula. OpenBTS - Installation and Configuration Guide. `http://gnuradio.org/redmine/attachments/139/OpenBTS_Guide_En_v0.1.pdf`, May 2009. Last accessed June 7, 2010.

[55] Philippe Oechslin. Making a Faster Cryptanlytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO 2003*, Lecture Notes in Computer Science, pages 617–630, 2003.

[56] P.Oechslin. Rainbow Cracking: Do you need to fear the Rainbow?, 2006. Last accessed June 8, 2010.

[57] Ian Poole. *Cellular Communications Explained: From Basics to 3G*. Elsevier Ltd, 2006.

[58] GNU Radio. Bts clocks. `http://gnuradio.org/redmine/wiki/1/OpenBTSClocks`.

[59] GNU Radio. Building and running openbts. `http://gnuradio.org/redmine/wiki/1/OpenBTSBuildingAndRunning`. Last accessed June 10, 2010.

[60] GNU Radio. OpenBTS Handset Compatibility. `http://gnuradio.org/redmine/wiki/1/OpenBTSCompatibility`. Last accessed June 9, 2010.

[61] Man Young Rhee. *Mobile Communication Systems and Security*. Wiley, 2009.

[62] Jochen H. Schiller. *Mobile Communication, 2nd edition*. Addison Wesley, 2003.

[63] Raymond Steele and Lajos Hanzo. *Mobile Radio Communications, 2nd Edition*. Wiley-IEEE Press, 1999.

[64] Frank A. Stevenson. A5/1 Security Project Mailing List, October 21, 2009.

[65] Frank A. Stevenson. Announcing "Berlin A5/1 rainbow table set". `http://lists.lists.reflextor.com/pipermail/a51/2010-June/000657.html`, June 2010. Last accessed June 16, 2010.

[66] Fabian van den Broek. Catching and Understanding GSM-Signals. `http://www.ru.nl/publish/pages/578936/fvdbroekscriptie.pdf`, March 2010. Last accessed June 10, 2010.

[67] M vd S. The A5/1 Security Project Mailing List. `http://lists.lists.reflextor.com/pipermail/a51/2010-January/000480.html`, January 9th 2010. Last accessed June 7, 2010.

[68] Marcin Wiaceks. NetMonitor in Nokia DCT1-DCT3 Phones. `http://www.mwiacek.com/www/?q=node/114`, October 2002. Last accessed May 8, 2010.

Appendix A

Gammu Tutorial

This tutorial covers the installation and configuration of Gammu.

Version 1.27.93 of the Gammu source code was downloaded from `http://dl.cihar.com/gammu/releases/gammu-1.27.93.tar.bz2`. To extract the source code:

```
tar xjvf gammu-1.27.93.tar.bz2
```

CMake is an open-source make system and must be installed in order for Gammu to be configured and compiled. Version 2.6.2 of CMake can be installed using the built-in Synaptic Package Manager[1] in Ubuntu. To configure, build and install Gammu [8]:

```
./configure
make
sudo make install
```

[1] System->Administration->Synaptic Package Manager

The package `dialog` displays user-friendly dialog boxes from shell scripts and is needed by Gammu. Install `dialog` by running the following command:

```
sudo apt-get install dialog
```

Gammu then needs to be configured to the state shown i figure A.1 by running:

```
gammu-config
```

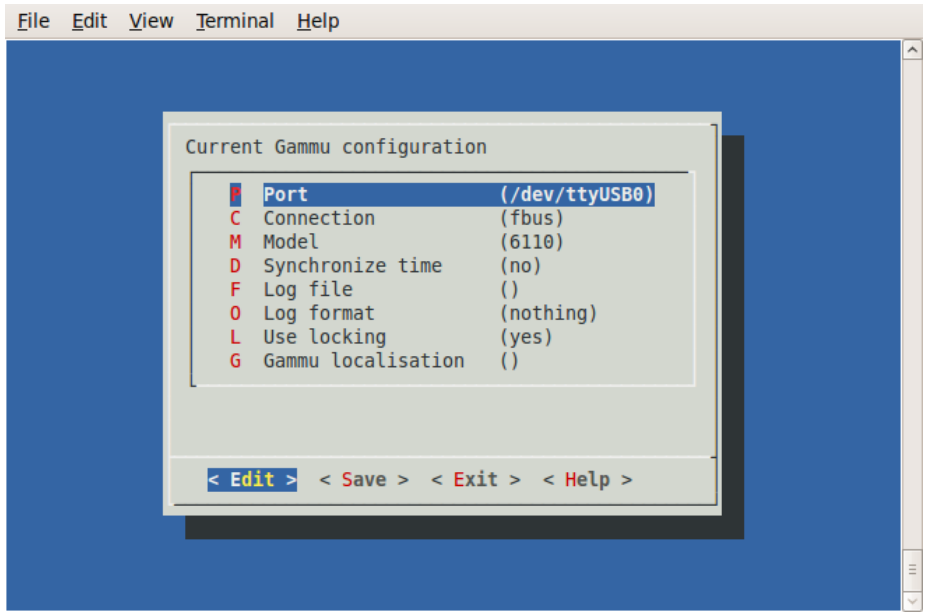

Figure A.1: The configuration of Gammu

Appendix B

Rainbow Table Generation

Appendix B gives a summary of the steps taken in order to set up a computer for rainbow table generation.

Step 1: Installing Ubuntu 9.04

First step was to make a clean install of Ubuntu 9.04 32-bit (Jaunty Jackalope). The iso file was be downloaded from `http://releases.ubuntu.com/9.04/`. After Ubuntu was installed, we ran *Update Manager* in order to have a fully updated system.

Step 2: Installing ATI Catalyst driver

Next step was to install a working driver for our ATI-card. We tested for both the ATI Radeon HD 5870[1] and the ATI Radeon HD 5970[2], and found that

[1] `http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/hd-5870/Pages/ati-radeon-hd-5870-overview.aspx`

[2] `http://www.amd.com/us/products/desktop/graphics/ati-radeon-hd-5000/hd-5970/`

the 5870-card works with the ATI Catalyst 9.10 driver, while the 5970-card works with the ATI Catalyst 9.12 driver. The 9.10-driver can be downloaded from `http://support.amd.com/us/gpudownload/linux/9-9/Pages/radeon_linux.aspx`, wheras the 5970-driver can be downloaded from `http://support.amd.com/us/gpudownload/linux/9-12/Pages/radeon_linux.aspx`. If you're in need of another driver, you can pick the driver of your choice from the ATI website (`http://ati.amd.com/support/driver.HTML`).

The rest of this appendix follows the installation of the 9.10 driver, but instructions are the same for all other driver versions, except that you have to change the file names.

After we had downloaded the 9.10 driver, we opened up a terminal window and typed in:

```
sudo sh ./ati-driver-installer-9-10-x86.x86_64.run
```

We selected *Install Driver* as seen in figure B.1 and clicked continue.

We then press *I Agree* when prompted about the licence agreement, and then select *Automatic* when asked about mode of installation, as seen in figure B.2.

A Window saying that the installation has finished was now displayed, as to where we clicked *Exit*. In the terminal window, we then typed in:

```
sudo /usr/bin/aticonfig --initial
```

A reboot of the system was then performed in order to finish the installation of the driver.

Step 3: Configuring the *xorg.conf* file

In order to get direct rendering to work, a few lines was added to the *xorg.conf*-file. In a terminal window, we typed in:

`pages/ati-radeon-hd-5970-overview.aspx`

```
sudo gedit /etc/X11/xorg.conf
```

Under *Section "Module"*, we then added the following lines:

```
Load "dri"
Load "glx"
```

And then added the following section to the document:

```
Section "dri"
    Mode 0666
EndSection
```

An example of how our *xorg.conf* looked like can be seen under:

```
Section "ServerLayout"
    Identifier "aticonfig Layout"
    Screen 0 "aticonfig-Screen[0]-0" 0 0
EndSection

Section "Files"
EndSection

Section "Module"
    Load "dri"
    Load "glx"
EndSection

Section "ServerFlags"
    Option "Xinerama" "off"
EndSection

Section "Monitor"
    Identifier "Configured Monitor"
EndSection

Section "Monitor"
    Identifier "aticonfig-Monitor[0]-0"
    Option "VendorName" "ATI Proprietary Driver"
    Option "ModelName" "Generic Autodetecting Monitor"
    Option "DPMS" "true"
EndSection
```

```
Section "Device"
    Identifier "Configured Video Device"
EndSection

Section "Device"
    Identifier "aticonfig-Device[0]-0"
    Driver "fglrx"
    BusID "PCI:4:0:0"
EndSection

Section "dri"
    Mode 0666
EndSection

Section "Screen"
    Identifier "Default Screen"
    Device "Configured Video Device"
    Monitor "Configured Monitor"
EndSection

Section "Screen"
    Identifier "aticonfig-Screen[0]-0"
    Device "aticonfig-Device[0]-0"
    Monitor "aticonfig-Monitor[0]-0"
    DefaultDepth 24
    SubSection "Display"
        Viewport 0 0
        Depth 24
    EndSubSection
EndSection
```

We then rebooted our system, in order to finish the configuration of our graphics card. To verify that everything was set up correctly we ran a few tests, and checked that they all passed:

```
glxinfo | grep rendering
glxgears
fgl_glxgears
```

Step 4: Installing ATI Compute Abstraction Layer (CAL)

In order for the rainbow table generation code to work, Compute Abstraction

Layer (CAL) needed to be installed. From the `http://developer.amd.com/gpu/ATIStreamSDK/ATIStreamSDKv1.4Beta/Pages/default.aspx` website, we downloaded the *atistream_1.4.0_beta-lnx32.tar.gzip* file into our home folder. Unfortunately, Ubuntu is not supported as an operating system for ATI Stream SDK v1.4-beta, but there's a few tricks that we did in order to make it work anyway.

First, the *zlib1g-dev*, *alien* and *libstdc++5* packages was installed by typing in the following command in a terminal window:

```
sudo apt-get install zlib1g-dev alien libstdc++5
```

We then typed in:

```
xdpyinfo | less
```

To be sure that *ATIFGLRXDRI* and *XFree86-DRI* showed up under the list of extensions. Had they not, a different driver might have been needed.

We then started the installation of ATI Cal by opening a terminal window and then typing in:

```
sudo tar xvfz atistream-1.4.0_beta-lnx34.tar.gzip
dd if=atistream-cal-1.4.0_beta.i386.run of=atical.tar.gz bs=1 skip=16384
```

This stripped away the run script at the beginning and created a tar-file instead. We then navigated to the */usr/local/* folder in a terminal window, and typed in:

```
sudo mkdir atical
cd atical
sudo tar xvfz /home/cryptos/atical.tar.gz
```

This extracted the tar-file and produced a RPM packet named *atistream-cal-1.4.0_beta-1.i386.rpm*. Since RPM packets are not supported by Ubuntu, we converted the ATI Cal packet into a debian packet by using *alien*. This was done by typing the following in a terminal window:

```
sudo alien atistream-cal-1.4.0_beta-1.i386.rpm
```

At last we did the install of ATI Cal by typing the following command in a terminal window:

```
sudo dpkg -i atistream-cal_1.4.0_beta-2_i386.deb
```

We then rebooted our system and ATI Cal in order to finish the installation of ATI Cal. As a test to check that it was working, we ran *FindNumDevices* from the */usr/local/atical/bin/lnx32* folder:

```
./FindNumDevices
```

Step 5: Compiling the rainbow table code

To check out the code from the project, subversion was needed:

```
sudo apt-get install subversion
```

When this was done, we started downloading the rainbow table code by typing the following in a terminal window:

```
svn co https://svn.reflextor.com/tmto-svn
```

When asked about certificate, we chose *p* in order to accept it permanently. During our timeline, several different revisions have been used, but the latest one checked out was revision 152.

The only thing left now was to compile the code. First the rainbow table generation code was compiled by running the following commands:

```
cd tmto-svn/tinkering/new_ati_code
make
mkdir tables
```

Note that the creation of *tables* directory is needed in order for the script to work. Having done this, we then compiled to code for lookup and other tools by opening a new terminal window and typing in the following:

```
cd tmto-svn/tinkering/A5Util
make
```

The computer was now set up properly and ready to generate rainbow tables.

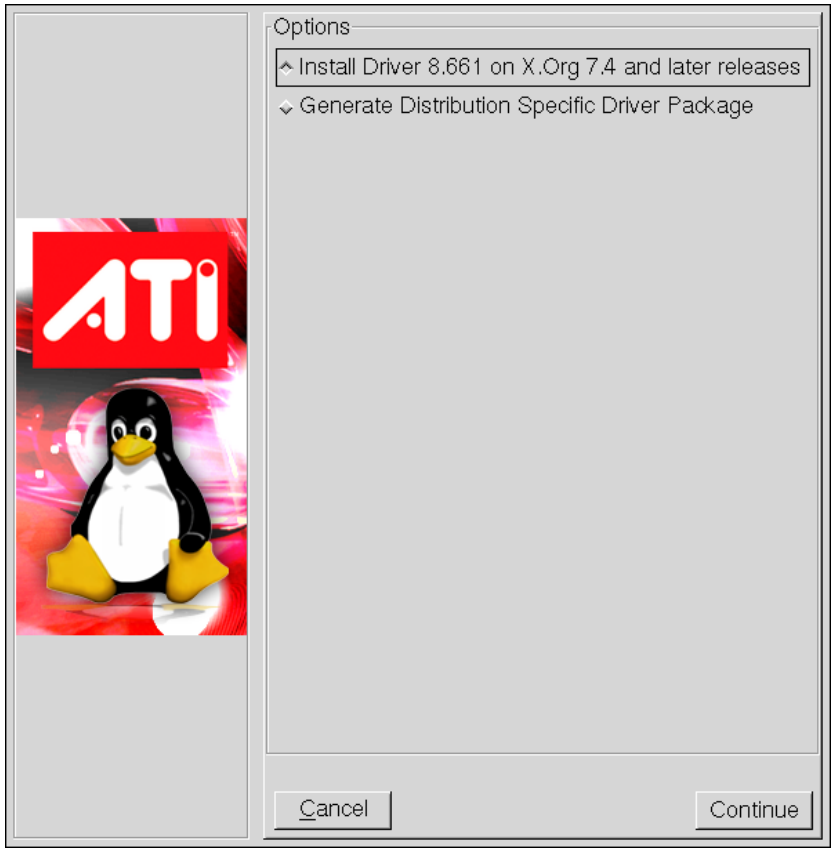

Figure B.1: ATI Catalyst 9.10 Driver Installation Window

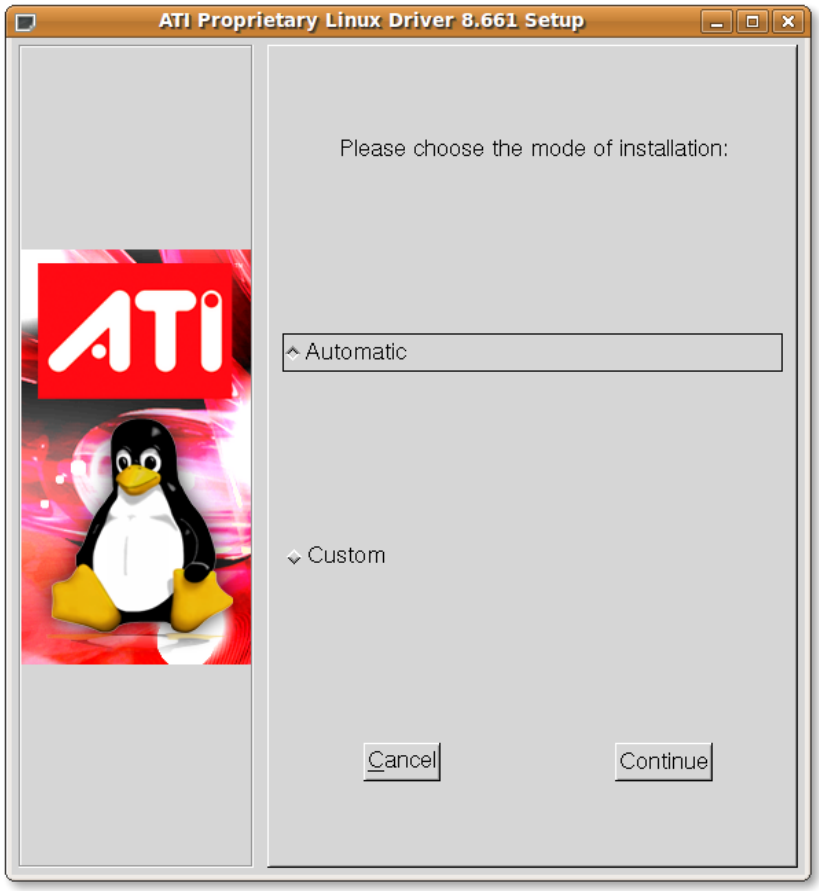

Figure B.2: Choosing mode of installation during ATI driver setup

Appendix C

Processing Crack Results

```
#!/usr/bin/python


f = open("10000keys.txt")
ks = f.readlines()
f.close()

keys = {}
foundFrames = {}
candidates = {}
duplicates = 0
candidateCounter = 0
for k in ks:
    pc = k.find(",")
    pk = k.find("key:")
    f = k[7:pc]
    key = k[pk+4:].strip()
    keys[int(f)] = key

f = open("found.txt","r")


line = f.readline()
found = False
```

```
while line!="":
    line = line.strip()

    if line[:11]=="Candidate: ":
        candidates[candidateCounter] = line[11:]
        candidateCounter += 1
    elif line[:4]=="### ":
        fr = line[13:]
        ps = fr.find(" ")
        frame = int(fr[:ps])
        if keys[frame] in candidates.values():
            candidates = {}
            if frame not in foundFrames:
              foundFrames[int(frame)] = keys[frame]
            else:
              duplicates += 1
            found = True


        if not found:
            if frame not in falseFrames:
              falseFrames[int(frame)] = 1
            else:
    falseFrames[int(frame)] += 1
        found = False
    line = f.readline()
print "------------------------------------------------------------"
print "Total number of false positives is:",sum([i for i in falseFrames.values()])
print "Total percentage of false positives is:", (sum([i for i in falseFrames.values()]) /
      10000.0)*100,""printprint "----------------------------"printtotalFrames =
      len(foundFrames
print "Total number of frames found is:",totalFrames
print "Total number of duplicates is:",duplicates
print "Total coverage is:",(totalFrames/10000.0)*100,""print
      "----------------------------"f.close(
```

Appendix D

Original Script from Frank Stevenson

```
#!/usr/bin/python


f = open("mykeys.txt")
ks = f.readlines()
f.close()

keys = {}
for k in ks:
    pc = k.find(",")
    pk = k.find("key:")
    f = k[7:pc]
    #print pc,pk,k
    key = k[pk+4:].strip()
    keys[int(f)] = key

#print keys

f = open("found.txt","r")
```

```
line = f.readline()
found = False
while line!="":
    line = line.strip()
    if line[:4]=="----":
        fr = line[28:]
        ps = fr.find(" ")
        frame = int(fr[:ps])
    elif line[:11]=="Candidate: ":
        key = line[11:]
        #while len(key)<16:
        # key = "0" + key
        if keys[frame]==key:
            #print "Found: ", frame, key
            found = True
    elif line[:12] == "### Frame is":
        if not found:
            print "Not found for frame: ", frame
        found = False
    line = f.readline()



f.close()
```

Appendix E

GNU Radio Tutorial

This appendix provides a tutorial on how to set up GNU Radio on Ubuntu 9.04 32-bit[4].

The GNU Radio version 3.1.3 source code is downloaded[1]. Version 3.2.2, which was the newest version as of April 27, 2010, was experimented with, but it has some compatibility issues with AirProbe, and thus it was decided to use the older version.

[1] ftp://ftp.gnu.org/gnu/gnuradio/gnuradio-3.1.3.tar.gz

To install packages required for compiling GNU Radio the following command is issued:

```
sudo apt-get -y install \
swig g++ automake1.9 libtool python-dev fftw3-dev \
libcppunit-dev libboost1.35-dev sdcc-nf libusb-dev \
libsdl1.2-dev python-wxgtk2.8 subversion git guile-1.8-dev \
libqt4-dev python-numpy ccache python-opengl libgsl0-dev \
python-cheetah python-lxml doxygen qt4-dev-tools \
libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools
```

After extracting `gnuradio-3.1.3.tar.gz` the following commands are issued to configure, make and install GNU Radio:

```
./configure
make
sudo make install
```

The `PYTHONPATH` has to be updated, and `ldconfig` has to be run:

```
export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.6/site-packages/
sudo ldconfig
```

Finally the user has to be given access to the USRP:

```
sudo addgroup usrp
sudo addgroup <USERNAME> usrp
echo 'ACTION=="add", BUS=="usb", SYSFS{idVendor}=="fffe", SYSFS{idProduct}=="0002", GROUP:="
     usrp", MODE:="0660"' > tmpfile
sudo chown root.root tmpfile
sudo mv tmpfile /etc/udev/rules.d/10-usrp.rules
```

Appendix F

AirProbe Tutorial

This appendix contains a tutorial on setting up AirProbe[6]:

```
sudo apt-get install git-core
git clone git://svn.berlin.ccc.de/airprobe
```

The package `libpcap0.8-dev` has to be installed[1] in order to be able to make `gsm-receiver`.

To build the receiver software:

```
cd airprobe/gsm-receiver
./bootstrap
./configure
make
```

[1] System->Administration->Synaptic Package Manager

To build the acquisition software:

```
cd airprobe/gsm-tvoid:
./bootstrap
./configure
make all
```

To build the frame decoder:

```
cd airprobe/gsmdecode
./bootstrap
./configure
make
```

Appendix G

OpenBTS Tutorial

Appendix G provides a tutorial for setting up OpenBTS 2.5.4 and Asterisk 1.4 on Ubuntu 9.04 with all its dependencies.

The following hardware requirements must be met

- A computer with USB-2.0 port.
- One USRP
- Two RFX900 daughtercards
- One VERT900 antenna
- Two GSM mobile phones
- Two SIM cards

The following software requirements must be met

- GNU Radio 3.1.3
- OpenBTS 2.5.4
- Asterisk 1.4
- Smqueue
- oSIP2 library files
- oRTP library files

Install two RFX900 daughterboards in the USRP. This serves the purpose of minimizing crosstalk between the receive and transmit sections. In the USRP, there is an "A" board (installed on "TXA" and "RXA") and a "B" board (installed on "TXB" and "RXB"). OpenBTS transmits on the "A" board, on the "TX/RX" connector, and receive on the "B" board, on the "RX2" connector.

The VERT900 antenna is mounted to the USRP on the receive section with a SMA connector. Configuring a transmitter antenna is not really necessary, as there is sufficient power leaks for close-range testing. In fact, using a transmitter antenna only causes transmit power to interfere with the receiver antenna, thereby reducing the performance and limiting the uplink range.

OpenBTS uses dependencies from GNU Radio. A detailed instruction of how to install GNU Radio is given as a tutorial in appendix E. In addition, library versions of oSIP and oRTP must be installed. The preferred way to install them in Ubuntu is to use the Synaptic Package Manager. They are found under the name libosip2 and libortp7.

Once all the dependencies are satisfied, download the OpenBTS 2.5.4 source code from

```
http://sourceforge.net/projects/openbts/files/
```

After extracting openbts-2.5.4Lacassine.tar.gz, the following commands are issued to configure, make and install OpenBTS:

```
./configure
make
sudo make install
```

After installation is finished, three files need to be edited: OpenBTS.config.example, sip.conf and extensions.conf.

Locate and open the OpenBTS.config.example in the openbts-2.5.4Lacassine/apps directory.

The following configurations have to be made:

- Set the OpenBTS up as a test network: enter the codes 001 and 01 for the GSM.MCC and GSM.MNC, respectively.
- Define the frequency band you will be operating on: set the GSM.Band to 900
- Find an appropriate ARFCN for your selected frequency band. This can be calculated from `http://www.aubraux.com/design/arfcn-calculator.php`. For the 928 MHz downlink frequency (E-GSM900), the ARFCN is 989.

OpenBTS and Asterisk will be running on localhost, so port assignments must be coordinated. These are the standard port assignments for an OpenBTS host [59]:

- OpenBTS runs SIP on UDP port 5062 and RTP on UDP ports 16484-16583.
- Asterisk runs SIP on UDP port 5060 and RTP on UDP ports 16386-16483.
- The SMS store and forward server smqueue, runs SIP on UDP port 5063.

When finished editing, save the file as OpenBTS.config in the same folder. A complete and working OpenBTS.config is given at the end of this appendix.

OpenBTS uses Asterisk for handling VoIP calls and user authentication. Every MS has to be registered in the sip.conf and extensions.conf with its IMSI. These two configuration files are found in the following folder: openbts-2.5.4Lacassine/AsteriskConfig.

Edit the extensions.conf with the following parameters:

```
...

[sip-local]
; local extensions

This is a simple mapping between extensions and IMSIs.
exten => 2101,1,Macro(dialSIP, Enter your IMSI here, e.g. IMSI242050123456789)
exten => 2102,1,Macro(dialSIP, Enter your IMSI here, e.g. IMSI242059876543210)

...
```

The extensions 2101 and 2102 will be used by the two corresponding mobile phones.

In sip.conf, the following changes must be made:

```
...

; This is a GSM handset entry.
; You need one for each SIM.
; The IMSI is a 15-digit code in the SIM.
; You can see it in the Control log whenever a phone tries to register.
[IMSI242050123456789] ; <- The IMSI is used as a SIP user ID.
canreinvite=no
type=friend
context=sip-external
allow=gsm host=dynamic
host=dynamic

[IMSI242059876543210] ; <- The IMSI is used as a SIP user ID.
canreinvite=no
type=friend
context=sip-external
allow=gsm host=dynamic
host=dynamic

...
```

When finished, copy extensions.conf and sip.conf to the /etc/asterisk directory.

Smqueue provides the SMS service. To build smqueue, go to the openbts-2.5.4Lacassine/smqueue directory and type:

```
make -f Makefile.standalone
```

If changes are made in extensions.conf or sip.conf while running the system, Asterisk has to be restarted by typing the following command:

```
sudo /etc/init.d/asterisk restart
```

Our complete OpenBTS.config is given below:

```
# Sample OpenBTS configuration file.
# Format of each line is. <key><space><value>
# The key name can contain no spaces.
# Everything between the first space and the end of the line becomes the value.
# Comments must start with "#" at the beginning of the line.
# Blank lines are OK.

# As a gerenal rule, non-valid configuration values will crash OpenBTS.

# Logging parameters

# The initial global logging level: ERROR, WARNING, NOTICE, INFO, DEBUG, DEEPDEBUG
LogLevel INFO

# The log file path. If not set, logging goes to stdout.
#LogFileName test.out

# Wireshark support
# The standard IANA for GSMTAP is 4729
# If if this is not defined, we do not generate the GSMTAP dumps.
Wireshark.Port 4729

# Port number for test calls.
# This is where an external program can interact with a handset via UDP.
TestCall.Port 28670

# Transceiver parameters

# Transceiver interface
# This TRX.IP is not really adjustable. Just leave it as 127.0.0.1.
TRX.IP 127.0.0.1
# This value is hard-coded in the transcevier. Just leave it alone.
TRX.Port 5700

# Path to transceiver binary
# YOU MUST HAVE A MATCHING libusrp AS WELL!!
TRX.Path ../Transceiver/transceiver

# TRX logging.
# Logging level.
TRX.LogLevel ERROR
# Logging file. If not defined, logs to stdout.
# TRX.LogFileName test.TRX.out

# SIP, RTP, servers
```

```
# Asterisk PBX
Asterisk.IP 127.0.0.1
Asterisk.Port 5060

# Messaging server
Messenger.IP 127.0.0.1
Messenger.Port 5063

# Local SIP/RTP ports
SIP.Port 5062
RTP.Start 16484
RTP.Range 98

# If Asterisk is 127.0.0.1, this is also 127.0.0.1.
# Otherwise, this should be the local IP address of the interface used to contact asterisk.
SIP.IP 127.0.0.1

# Local SMS port for short code delivery.
SMSLoopback.Port 5064

# Special extensions.

# Routing extension for emergency calls.
# PBX.Emergency 2101

# SIP parameters

# SIP registration period in seconds.
# Ideally, this should be slightly longer than GSM.T3212.
SIP.RegistrationPeriod 3600

# SIP Internal Timers. All timer values are given in millseconds.
# These are from RFC-3261 Table A.

# SIP Timer A, the INVITE retry period, RFC-3261 Section 17.1.1.2
SIP.Timer.A 1000

# SMS parameters

# ISDN address of source SMSC when we fake out a source SMSC.
SMS.FakeSrcSMSC 0000
# ISDN address of destination SMSC when a fake value is needed.
SMS.DefaultDestSMSC 0000

# The SMS HTTP gateway.
```

```
# Comment out if you don't have one or if you want to use smqueue.
#SMS.HTTP.Gateway api.clickatell.com

# IF SMS.HTTP.Gateway IS DEFINED, SMS.HTTP.AccessString MUST ALSO BE DEFINED.
#SMS.HTTP.AccessString sendmsg?user=xxxx&password=xxxx&api_id=xxxx

# Open Registration and Self-Provisioning
# This is a bool and if set to 1, OpenBTS will allow all handsets to register
Control.OpenRegistration 1

# "Welcome" messages sent during IMSI attach attempts.
# ANY WELCOME MESSAGE MUST BE LESS THAN 161 CHARACTERS.
# ANY DEFINED WELCOME MESSAGE MUST ALSO HAVE A DEFINED SHORT CODE.
# Comment out any message you don't want to use.

# The message sent upon full successful registration, all the way through the Asterisk server
      .
 Control.NormalRegistrationWelcomeMessage Welcome to Cryptos!
 Control.NormalRegistrationWelcomeShortCode 0000

# Then message sent to accpeted open registrations.
# IF OPEN REGISTRATION IS ENABLED, THIS MUST ALSO BE DEFINED.
Control.OpenRegistrationWelcomeMessage You're doing it wrong! Welcome anyway!
Control.OpenRegistrationWelcomeShortCode 23

# Then message send to failed registrations.
Control.FailedRegistrationWelcomeMessage It's been fun. See you next year.
Control.FailedRegistrationWelcomeShortCode 666

# GSM

# Network and cell identity.

# Network Color Code, 0-7
GSM.NCC 0
# Basesation Color Code, 0-7
GSM.BCC 0
# Mobile Country Code, 3 digits.
# US is 310
# MCC MUST BE 3 DIGITS. Prefix with 0s if needed.
# Test code is 001.
GSM.MCC 001
# Mobile Network Code, 2 or 3 digits.
# Test code is 01.
GSM.MNC 01
# Location Area Code, 0-65535
```

```
GSM.LAC 666
# Cell ID, 0-65535
GSM.CI 10
# Network "short name" to display on the handset.
# SHORT NAME MUST BE LESS THAN 8 CHARACTERS.
GSM.ShortName Cryptos

# Assignment type for call setup.
# This is defined in an enum AssignmentType in GSMCommon.h.
# 0=Early, 1=VeryEarly.
GSM.AssignmentType 1

# Band and Frequency

# Valid band values are 850, 900, 1800, 1900.
GSM.Band 900
#GSM.Band 850

# Valid ARFCN range depends on the band.
GSM.ARFCN 986
# ARCN 975 is inside the US ISM-900 band and also in the GSM900 band.
#GSM.ARFCN 975
# ARFCN 207 was what we ran at BM2008, I think, in the GSM850 band.
#GSM.ARFCN 207

# Neightbor list
GSM.Neighbors 29

# Downlink tx power level, dB wrt full power
GSM.PowerAttenDB 0

# Channel configuration
# Number of C-VII slots (8xSDCCH)
GSM.NumC7s 1
# Number of C-I slots (1xTCH/F)
GSM.NumC1s 5

# Beacon parameters.

# L1 radio link timeout advertised on BCCH.
# This is the RAW parameter sent on the BCCH.
# See GSM 10.5.2.3 for encoding.
# Value of 15 gives 64-frame timeout, about 30 seconds on the TCH.
# This should be coordinated with T3109.
GSM.RADIO_LINK_TIMEOUT 15
```

```
# Control Channel Description (CCD)

# Attach/detach flag.
# Set to 1 to use attach/detach procedure, 0 otherwise.
# This will make initial registration more prompt.
# It will also cause an un-regstration if the handset powers off.
GSM.CCD.ATT 1

# CCCH_CONF
# See GSM 10.5.2.11 for encoding.
# Value of 1 means we are using a C-V beacon.
GSM.CCD.CCCH_CONF 1

# RACH Parameters

# Maximum RACH retransmission attempts
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
GSM.RACH.MaxRetrans 3

# Parameter to spread RACH busts over time.
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
GSM.RACH.TxInteger 14

# Access class flags.
# This is the RAW parameter sent on the BCCH.
# See GSM 04.08 10.5.2.29 for encoding.
# Set to 0 to allow full access.
GSM.RACH.AC 0

GSM.RACH.CellBarAccess 0

# NCCs Permitted.
# An 8-bit mask of allowed NCCs.
# Unless you are coordinating with another carrier,
# this should probably just select your own NCC.
GSM.NCCsPermitted 1

# Cell Selection Parameters (CS)

GSM.CS.MS_TXPWR_MAX_CCH 0
GSM.CS.RXLEV_ACCESS_MIN 0

# Cell Reselection Hysteresis
# See GSM 04.08 10.5.2.4, Table 10.5.23 for encoding.
```

```
# Encoding is 2N dB, value values of N are 0..7 for 0..14 dB.
GSM.CS.CELL_RESELECT_HYSTERESIS 7

# Reject cause for location updating failures
# Reject causes come from GSM 04.08 10.5.3.6
# Reject cause 0x04, IMSI not in VLR
GSM.LURejectCause 0x04

# Maximum TA for accepted bursts.
# Can be used to control the range of the BTS.
# The unit is GSM symbols of round trips delay, about 550 meters per symbol.
GSM.MaxRACHDelay 20

# GSM Timers. All timer values are given in milliseconds unless stated otherwise.
# These come from GSM 04.08 11.2.

# T3212, registration timer.
# Unlike most timers, this is given in MINUTES.
# Actual period will be rounded down to a multiple of 6 minutes.
# Any value below 6 minutes disables periodic registration, which is probably a bad idea.
# Valid range is 6..1530.
# Ideally, this should be slightly less than the SIP.RegistrationPeriod.
GSM.T3212 6

# T3122, RACH holdoff timer.
# This value can vary internally between the min and max ends of the range.
# When congestion occurs, T3122 grows exponentially.
GSM.T3122Min 2000
# T3211Max MUST BE NO MORE THAN 255 ms.
GSM.T3122Max 255000
```

Appendix H

LAPDm Frames

This appendix describes the frame structure, format of fields and the procedures of the LAPDm in the data link layer.

LAPDm supports two operational modes: acknowledged mode (multiple frame) and unacknowledged mode.

In acknowledged mode, data is transmitted in Numbered Information (I) frames that are acknowledged by the receiving data link layer. Lost messages are retransmitted and flow control procedures are specified and activated. An example of such message is the SMS, which is sent between the MS and BTS. This operation is initiated by using a Set Asynchronous Balanced Mode (SABM) command.

In unacknowledged mode, data is transmitted in Unnumbered Information (UI) frames. No flow control mechanisms nor error recovery mechanisms are defined for these messages. An example of such message is the 'Measurement Result', which is sent periodically by the BTS to the BSC.

Figure H.1 shows an overview of the different types of LAPDm frames. Their respective uses depend on the type of information to be transmitted.

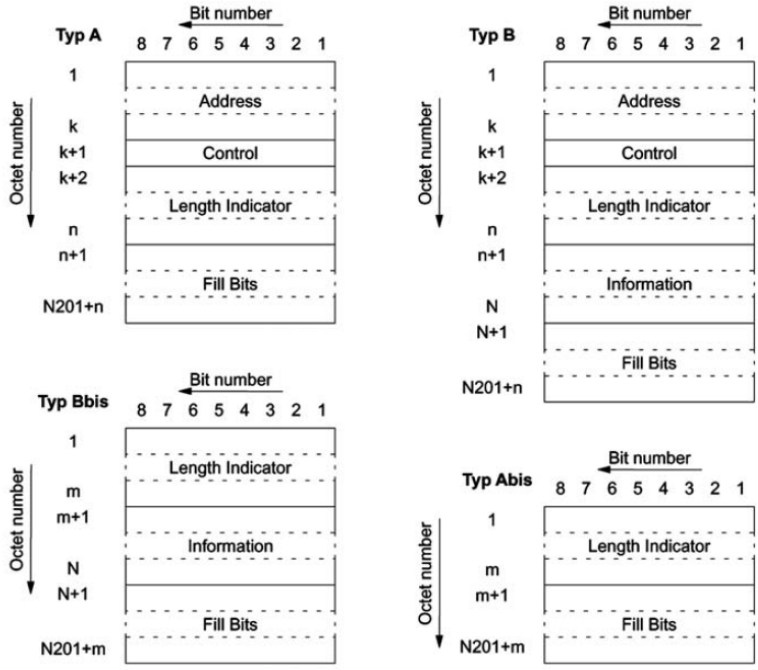

Figure H.1: LAPDm Frame Formats [30].

The type A frame is sent (on DCCH) in acknowledged mode as a fill frame when no payload is available in an active connection. The type B frame is used (on DCCH) for exchanging the actual signaling data. For unacknowledged mode, format types Abis and Bbis are used. They are characterized by the fact that they do not have an address field.

A-format and B-format frames are transmitted in both uplink and downlink, whereas Abis and Bbis frames are only sent on the downlink (BCCH, PCH, AGCH).

The Address field is further divided into five main parts, as shown in figure H.2.

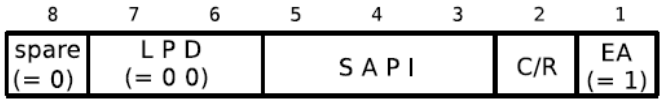

Figure H.2: Address Field

- Spare bit (spare, 1 bit): currently unused and reserved for future use.
- Link Protocol Discriminator (LPD, 2 bits): always set to 00 and serve the purpose to specify the use of LAPDm.
- Service Access Point Identifier (SAPI, 3 bit): set to 000 for RR, MM and CC messages, or 011 for low-priority messages such as SMS.
- Command/Response field bit (C/R, 1 bit): used to specify whether the message is a command or a response.
- Address field extension bit (EA, 1 bit): by default set to 1 (meaning that there is no further extension of the address field).

The control field, as shown in figure H.3, is used to carry the sequence number and to specify the type of frame (command or response). It is composed of the following parts:

- Numbered Information transfer format (I format): used to perform an information transfer between layer 3 entities.
- Supervisory format (S format): used to perform data link administrative control functions, such as: acknowledge I frames, request retransmission of I frames and request a temporary suspension of I frame transmissions.
- Unnumbered format and Control Functions (U format): used to provide additional data link control functions and unacknowledged information transfer.

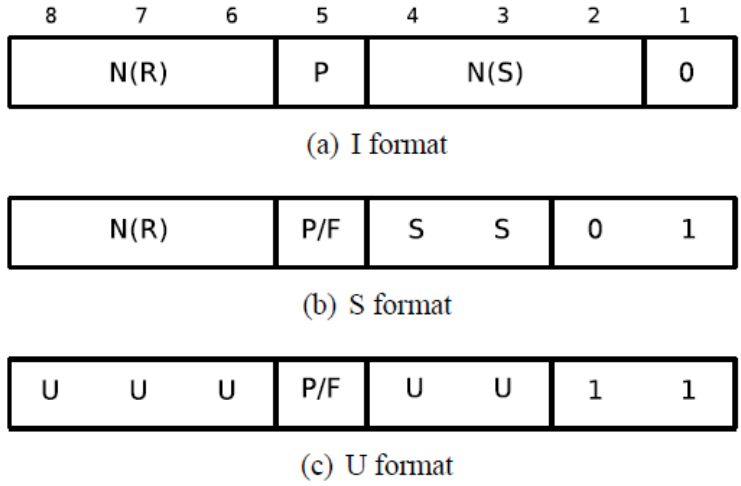

Figure H.3: Control Field

As figure H.3 shows, these formats contain several new fields:

- Send Sequence Number (N(S)) and Receive Sequence Number (N(R)): serve the purpose of acknowledging the transfer and the receipt of I frames. Three bits are used, allowing frame number values between 0 and 7.
- Poll/Final bit (P/F): used to indicate the function of a frame. If it is a Command frame, a one in this bit indicates a poll. If the frame is a 'Response', a one in this bit indicates that the current frame is the final one.
- Supervisory function (S) and Unnumbered function (U): used to encode certain command messages like Receive Ready (RR), Disconnect(DISC) and SABM. A detailed definition of these commands are given in [38]

The length indicator field, as depicted in figure H.4, is used to distinguish the information carrying field from the fill-in bits. It is composed of three parts:

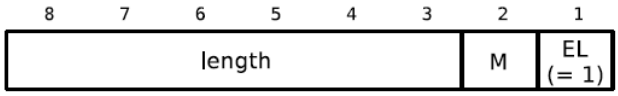

Figure H.4: Length Indicator Field

- Length indicator (L, 6 bits): used to indicate the number of octets contained in the information field.
- More data bit (M, 1 bit): used to indicate segmentation of layer 3 message on the data link layer. If the M bit is set to '1', it indicates that the information field of this message contain only a part of a layer 3 message. If the M bit is set to '0', it indicates:
  - that the information field contains a complete layer 3 message unit provided that the M bit of the previous frame was set to '0'.
  - that the information field contains the last segment of a layer 3 message unit if the M bit of the previous frame was set to '1'.
- Length indicator field extension bit (EL, 1 bit): default set to 1 (meaning that it is the final octet of the field).

The Information Field itself carries the signaling data up to a maximum number of octets (N201), which depends on the type of logical channel being transmitted.

Finally, if a LAPDm frame is less than 184 bits (or 23 octets), so called fill bits are added. An octet of fill bits sent by the network is set to '00101011' or a random value. '00101011' or '11111111' or a random value is used as fill bits when sent by the MS. '00101011' is chosen as default because of its relation with the modulation and interleaving scheme used in GSM [38]. The '00101011' decodes to the hexadecimal value "2B".

Acronyms

2G Second-generation

A3 Authentication Algorithm

A5 Encryption Algorithm

A8 Key Generation Algorithm

AGCH Access Grant Channel

AMR Adaptive Multi-Rate

ARFCN Absolute Radio Frequency Channel Number

AuC Authentication Center

BCCH Broadcast Control Channel

BCC Base station Colour Code

BCH Broadcast Channels

BSIC Base Station Identity Code

BSS Base Station Subsystem

BTS Base Transceiver Station

C/R Command/Response Field

CCCH Common Control Channels

CC Call Control

CC Country Code

CI Cell Identifier

CKSN Cipher Key Sequence Number

CLI Command Line Interface

CM Connection Management

CPU Central Processing Unit

CUDA Compute Unified Device Architecture

DCH Dedicated Channels

EA Address Field Extension

EFR Enhanced Full Rate

EIR Equipment Identity Register

EL Length Indicator Field Extension

FACCH Fast Associated Control Channel

FCCH Frequency Correction Channel

FDMA Frequency-Division Multiple Access

FN Frame Number

FPGA Field Programmable Gate Array

GDM GNOME Display Manager

GMSC Gateway Mobile Switching Center

GMSK Gaussian Minimum-Shift Keying

GPU Graphical Processing Unit

GSM Global System for Mobile Communications

HLR Home Location Register

HSN Hopping Sequence Number

IMEISV International Mobile Equipment Identity Software Version

IMEI International Mobile Equipment Identity

IMSI International Mobile Subscriber Identity

ITU International Telecommunication Union

I Numbered Information

$\mathbf{K_c}$ 128-bit Session Key

$\mathbf{K_i}$ 128-bit Individual Subscriber Authentication Key

LAC Location Area Code

LAI Location Area Identity

LAPD Link Access Procedures on the D-channel

LFSR Linear feedback shift register

LPC Link Protocol Discriminator

lsb least significant bit

L Length Indicator

MAIO Mobile Allocation Index Offset

MA Mobile Allocation

MCC Mobile Country Code

ME Mobile Equipment

MM Mobility Management

MNC Mobile Network Code

MO-SMS Mobile-Originating SMS

MOC Mobile-Originating Call

msb most significant bit

MSC Mobile Switching Center

MSD Message Sequence Diagrams

MSIN Mobile Subscriber Identification Number

MSISDN Mobile Station Integrated Services Digital Network

MSRN Mobile Subscriber Roaming Number

MS Mobile Station

MT-SMS Mobile-Terminating SMS

MTC Mobile-Terminating Call

M More Data

N(R) Receive Sequence Number

N(S) Send Sequence Number

NCC Network Colour Code

NDC National Destination Code

NSS Network Switching Subsystem

OACSU Off Air Call Set-Up

P/F Poll/Final

PBX Private Branch Exchange

PCH Paging Channel

PLMN Public Land Mobile Network

RACH Random Access Channel

RAND 128-bit Random Number

RPC-LPC Regular Pulse Excitation - Linear Predictive Coding

RPDU Relay Protocol Data Unit

RR Radio Resource Management

RX Receive

SABM Set Asynchronous Balanced Mode

SACCH Slow Associated Control Channel

SAPI Service Access Point Identifier

SCH Synchronization Channel

SDCCH Standalone Dedicated Control Channel

SDK Software Development Kit

SDR Software Defined Radio

SFH Slow Frequency Hopping

SIM Subscriber Identity Module

SIP Session Initiation Protocol

SMA SubMiniature version A

SMS-GMSC Short Message Service Gateway Mobile Switching Center

SMS-G Short Message Service Gateway

SMS-IWMSC Short Message Service Inter-Working Mobile Switching Center

SMSC Short Message Service Center

SMS Short Message Service

SNR Serial Number

SN Subscriber Number

SP Spare

SQL Structured Query Language

SRES Signed Response

SS Supplementary Services

TAC Type Allocation Code

TCH Traffic Channels

TDD Time-Division Duplex

TDMA Time-Division Multiple Access

TMSI Temporary Mobile Subscriber Identity

TMTO Time-Memory Trade-Off

TRAU Transcoder and Rate Adaptation Unit

TX Transmit

T Timeslot

UDP User Datagram Protocol

UI Unnumbered Information

USB Universal Serial Bus

USRP Universal Software Radio Peripheral

VLR Visitor Location Register

VoIP Voice over Internet Protocol