

How to Intercepting GSM

by

Grigorakis Ioannis

A THESIS

Submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

September 2015

Approved by:

Supervisors

Ass. Prof. Harry Manifavas

Abstract

Communication security is a great issue, notably in our day's that we live in technologically advanced community. The GSM is a protocol that used for digital communications (mobile phones), used by a lot of people around the world.

The most of people have mobile phones. Through, we are not cognizant the ease of intercepting or tracking a mobile phone. To draw the heed of a wider public to this security issue, we have to do intercepting and tracking achievable without spending a lot of money for the equipment. In this Master thesis we have hence taken a software radio and a laptop, put it together with open source software applications to build an GSM scanner.

How to intercept the GSM protocol is not a new thing. However, up to date this usually requires very expensive equipment which may not be available to everyone. Hence, in this thesis we employ a software radio to capture and to digitize GSM signals in combination with a standard laptop for further processing. This makes intercepting feasible for students.

As a software radio that captures the signals between the base transceiver station and the mobile phone, we have used a Universal Software Radio Peripheral. On the laptop we run GNURadio and Airprobe to decode the digitized signals. The cost of this setup is limited to a few hundred Euros.

We also revise the main weaknesses of the GSM A5 / 1 encryption enable building Rainbow tables and perform widely known plaintext attack to determine the encryption key and decrypt the encrypted message text.

Σύνοψη

Η ασφάλεια είναι ένα σημαντικό ζήτημα, ιδιαίτερα στη σημερινή προηγμένη τεχνολογικά κοινωνία. Το GSM είναι ένα παγκόσμιο πρότυπο για ψηφιακά ασύρματα κινητά τηλέφωνα, που χρησιμοποιείται σήμερα από εκατομμύρια ανθρώπους σε όλο τον πλανήτη.

Οι περισσότεροι από μας φέρουν πάντα κινητά τηλέφωνα. Ωστόσο, δεν έχουμε επίγνωση του πόσο εύκολο είναι να υποκλέπτούν ή να παρακολουθούν ένα κινητό τηλέφωνο. Για να επιστήσουμε την προσοχή ενός ευρύτερου κοινού σε αυτό το θέμα της ασφάλειας, θα πρέπει να δείξουμε πως μπορεί να γίνει η υποκλοπή και η παρακολούθηση εφικτή, με χαμηλό προϋπολογισμό για αγορά εξοπλισμού. Σε αυτή τη διατριβή, ως εκ τούτου έχουμε λάβει ένα Software radio και έναν φορητό υπολογιστή, επίσης εργαλεία ανοικτού κώδικα λογισμικό για να δημιουργήσουμε ένα σαρωτή GSM.

Η υποκλοπή του GSM δεν είναι κάτι καινούργιο. Ωστόσο, μέχρι σήμερα αυτό απαιτεί συνήθως ένα πολύ απαιτητικό εξοπλισμό που μπορεί να μην είναι διαθέσιμος σε όλους. Ως εκ τούτου, στην παρούσα διατριβή θα απασχολεί με ένα φτηνό εξοπλισμό RTL-SDR για να συλλάβει και να ψηφιοποιεί τα σήματα GSM σε συνδυασμό με ένα φορητό υπολογιστή για περαιτέρω επεξεργασία.

Ως ένα Software Radio που συλλαμβάνει τα σήματα μεταξύ του σταθμού βάσης πομποδέκτη και του κινητού τηλεφώνου, έχουμε χρησιμοποιήσει ένα λεγόμενο Software Radio Peripheral. Για το φορητό υπολογιστή που τρέχει GNURadio και Airprobe να αποκωδικοποιήσει τα ψηφιακά σήματα. Το κόστος αυτής της εγκατάστασης είναι περιορισμένο σε μερικές εκατοντάδες ευρώ.

Μπορούμε επίσης να δείξουμε τις βασικές αδυναμίες της A5/1 κρυπτογράφησης GSM που καθιστούν δυνατή την κατασκευή Rainbow Tables και να εκτελέσει την ευρέως γνωστή plaintext επίθεση για να προσδιοριστεί το κλειδί κρυπτογράφησης και να αποκρυπτογραφήσει το κρυπτογραφημένο μήνυμα κειμένου SMS η ομιλίας

Acknowledgements

I would like to express my deep gratitude to Professor Harry Manifavas.

I would like to thank Xari Manifava , my Family , my friends and some hidden heroes because of their confidence and their faith for the completion of this Master Thesis.

Last but not least, All the people of the forums .

Table of Contents

Chapter 1 : GSM.....	7
1.1 GSM Network.....	9
1.1.1 Short Brief About	9
1.1.2 Uplinks/Downlinks & Reverse Forward.....	9
1.1.3 Frequency Division Multiple Access (FDMA)	10
1.1.4 Absolute Radio Frequency Channel Number (ARFCN)	10
1.1.5 Calculating Uplink/Downlink Frequencies.....	11
1.1.6 International Mobile Subscriber Identity (IMSI)	11
1.1.7 Mobile Country Code (MCC).....	12
1.1.8 Mobile Network Code (MNC).....	12
1.1.9 International Mobile Equipment Identity (IMEI).....	12
1.1.10 Type Allocation Code (TAC)	13
1.1.11 Serial Number (SNR)	13
1.2 Network Architecture	14
1.2.1 Mobile Equipment (ME).....	14
1.2.2 Identity Module (SIM).....	14
1.2.3 Base Transceiver Station (BTS)	14
1.2.4 Base Station Controller (BSC).....	15
1.2.4 Mobile Switching Center (MSC).....	15
1.2.5 Gateway Mobile Switching Center (GMSC).....	15
1.2.6 Home Location Register (HLR).....	16
1.2.7 Visitor Location Register (VLR).....	16
1.2.8 Location Area Code (LAC).....	16
1.2.9 Location Area Identity (LAI)	16
1.2.10 Cell Global Identification (CGI).....	16
1.2.11 Cell Global Identity	17
1.2.12 Equipment Identity Register (EIR)\	17
1.2.13 Authentication Center (AuC).....	17
Chapter 2: Spectrum Analysis Hardware.....	19
2.1 Software Defined Radio (SDR)	19
2.1 HackRF One.....	19
2.2 USRP	21
2.2.1 USRP1.....	21
2.2.2 USRP2.....	22
2.2.3 USRP Daughter boards	22
2.3 RTL2832U Chipset - USB TV receiver.....	23
Chapter 3. Spectrum Analysis Software	25
3.1 Gqrx	26
3.2 AirProbe.....	27
3.3 GNU Radio	28
3.4 ArfcnCalc - GSM frequency calculation tool V.1.0	29
3.5 Kalibrate Tool.....	31
3.5 Wireshark	33
Chapter 4. Analyzing GSM With Airprobe And Wireshark.....	34
4.1 Install GNU Radio.....	35
4.2 Install Libosmocore	36
4.3 Install Airprobe.....	36

Chapter 5 : A5/1	38
5.1 About A5/1.....	38
5.2 Passive GSM interception.....	43
Chapter 6 : Practical exercise on the GSM Encryption A5/1	49
6.1 Analyze The Capture File.....	49
6.2 Finding Information to Crack the Key	53
6.3 Decoding the channel.....	55
Chapter 7: How to Decode Our Gsm Traffic	59
7.1 Kc key and TMSI number	59
7.2 Active Read Kc And TMSI From SIM.....	60
7.2.1 BlackBerry Engineering Screen	60
7.2 Receive Live Channel	65
7.3 Capturing a cfile with the RTL-SDR In Our Kali Linux VM	69
7.4 Use airprobe to send the decoded information to wireshark and analyze the frames.....	73
Chapter 8. Projects & Tools For Analyzing Intercepting Gsm Signals.....	80
8.1 The gr-gsm project.....	80
8.2 The Rtl -tool-kit Project	84
8.3 The Pytacle-Alpha2 Project.....	86

List of Figures

Figure 1 . Downlink And Uplink	10
Figure 2 . GSM BANDS	11
Figure 3 . IMSI Bits	12
Figure 4 . Gsm Arhitecture 1st part 1	15
Figure 5 . Gsm Architecture.....	18
Figure 8. USRP 2	22
Figure 9. RTL2832U Chipset - USB TV	23
Figure 10 . Radio FM spectrum using SDR 1	24
Figure 11. GSM Spectrum using SDR.....	25
Figure 12 . Gqrx.....	26
Figure 13 . Gnu-radio develop Environment	28
Figure 14. ArfcnCalc Help Envirment.....	31
Figure 15 . Kalibrate Band Scan Command	32
Figure 16. Kalibrate Channel Command	32
Figure 17. Wireshark Sniffing Enviroment	33
Figure 18 . Sniffing Packets from UM Air And Analyzed in Wireshark	34
Figure 20 . Packets On Wireshark	37
Figure 21 . Linear feedback shift registers.....	39
Figure 22 . A5/1 stream cipher uses 3 LFSRs	39
Figure 23 . Copying Rainbow Tables on Sda3 Partition	46
Figure 24 . Rainbow Tables Copied	47
Figure 25 . Testing Kraken	48
Figure 26 . Decode the Capture File	49
Figure 27 . Decoded Packets in Wireshark.....	50
Figure 28 . Decoder now only decodes TimeSlot 1	51
Figure 29 . Encrypted Bursts in vf_call6	51
Figure 30 . System Information 5 & 6 packets	52
Figure 31 . Informations Of System Information Type 5	53
Figure 32 . Decode encrypted part of the assigned “SDCCH/8”	55
Figure 33 . Decode encrypted part of the assigned “SDCCH/8” Wireshark	56
Figure 34 . Toast Tool.....	57
Figure 35 . Vlc Playing the speech.au.....	58
Figure 36 . OMNIKEY® 5321 SIM-card reader.....	61
Figure 37 . Test Minicom With Samsung Device.....	64
Figure 38 . Read TMSI and Kc.....	64
Figure 39 . Sniffing Packets With Rtl-sdr.....	65
Figure 40 . Captured System Information Type 4 –General Information.....	66
Figure 41 . Bin to Cfile .grc File.....	70
Figure 42 . Gsm.grc	71
Figure 43 . Gsm.grc Generated with Gnu-radio	72
Figure 44 . Dumb.cfile	72
Figure 45 . Available configurations Supported By Airprobe.....	74
Figure 46 . Flowchart of the whole process Decoding Sms	76
Figure 47 . Flowchart of the whole process Decoding Voice Call	78
Figure 48 . Linux Mint 17 VMachine & Gr-gsm project application files.....	80
Figure 49 . Test of Gnu-Radio	81
Figure 50 . Commands to run Airprobe_rtlsdr.py.....	82
Figure 51 . Airpobe Rtlsdr While running and Capturing GSM packets.....	82
Figure 52 Airpobe Rtlsdr & GSM packets on Wireshark.....	84

Figure 53. Rtl-Tool Kit While Running.....85

Chapter 1 : GSM

1.1 GSM Network

1.1.1 Short Brief About

GSM is an acronym that stands for Global System for Mobile Communications. The original french acronym stands for Groupe Spécial Mobile. It was originally developed in 1984 as a standard for a mobile telephone system that could be used across Europe.

GSM is now an international standard for mobile service. It offers high mobility. Subscribers can easily roam worldwide and access any GSM network.

GSM is a digital cellular network. At the time the standard was developed it offered much higher capacity than the current analog systems. It also allowed for a more optimal allocation of the radio spectrum, which therefore allows for a larger number of subscribers.

GSM offers a number of services including voice communications, Short Message Service (SMS), fax, voice mail, and other supplemental services such as call forwarding and caller ID.

Currently there are several bands in use in GSM. 450 MHz, 850 MHz, 900 MHz, 1800 MHz, and 1900 MHz are the most common ones.

Some bands also have Extended GSM (EGSM) bands added to them, increasing the amount of spectrum available for each band.

GSM makes use of Frequency Division Multiple Access (FDMA) and Time Division Multiple Access (TDMA).

*TDMA will be discussed later

1.1.2 Uplinks/Downlinks & Reverse Forward

GSM allows for use of duplex operation. Each band has a frequency range for the uplink (cell phone to tower) and a separate range for the downlink (tower to the cell phone). The uplink is also known as the Reverse and the downlink is also known as the Forward. In this tutorial, I will use the terms uplink and downlink.



Figure 1 . Downlink And Uplink

1.1.3 Frequency Division Multiple Access (FDMA)

GSM divides the allocated spectrum for each band up into individual carrier frequencies. Carrier separation is 200 khz. This is the FDMA aspect of GSM.

1.1.4 Absolute Radio Frequency Channel Number (ARFCN)

The ARFCN is a number that describes a pair of frequencies, one uplink and one downlink. The uplink and downlink frequencies each have a bandwidth of 200 kHz. The uplink and downlink have a specific offset that varies for each band. The offset is the frequency separation of the uplink from the downlink. Every time the ARFCN increases, the uplink will increase by 200 and the downlink also increases by 200 khz.

*Note: Although GSM operates in duplex (separate frequencies for transmit and receive), the mobile station does not transmit and receive at the same time. A switch is used to toggle the antenna between the transmitter and receiver.

The following table summarizes the frequency ranges, offsets, and ARFCNs for several popular bands

System	Band	Uplink	Downlink	Channel Number
GSM 400	450	450.4 - 457.6	460.4 - 467.6	259 - 293
GSM 400	480	478.8 - 486.0	488.8 - 496.0	306 - 340
GSM 850	850	824.0 - 849.0	869.0 - 894.0	128 - 251
GSM 900 (P-GSM)	900	890.0 - 915.0	935.0 - 960.0	1 - 124
GSM 900 (E-GSM)	900	880.0 - 915.0	925.0 - 960.0	975 - 1023, (0, 1-124)
GSM-R (R-GSM)	900	876.0 - 915.0	921.0 - 960.0	955 - 973, (0, 1-124, 975 - 1023)
DCS 1800	1800	1710.0 - 1785.0	1805.0 - 1880.0	512 - 885
PCS 1900	1900	1850.0 - 1910.0	1930.0 - 1990.0	512 - 810

Figure 2 . GSM BANDS

1.1.5 Calculating Uplink/Downlink Frequencies

The following is a way to calculate the uplink and downlink frequencies for some of the bands, given the band, the ARFCN, and the offset.

GSM 900

$$\text{Up} = 890.0 + (\text{ARFCN} * .2)$$

$$\text{Down} = \text{Up} + 45.0$$

Example :

Given the ARFCN 72, and we know the offset is 45MHz for the GSM900 band:

$$\text{Up} = 890.0 + (72 * .2)$$

$$\text{Up} = 890.0 + (14.4)$$

$$\text{Up} = 904.40 \text{ MHz}$$

$$\text{Down} = \text{Up} + \text{Offset}$$

$$\text{Down} = 904.40 + 45.0$$

$$\text{Down} = 949.40 \text{ MHz}$$

The uplink/downlink pair for GSM900 ARFCN72 is 904.40/949.40 (MHz)

1.1.6 International Mobile Subscriber Identity (IMSI)

The IMSI is how the subscriber is identified to the network. It uniquely identifies the subscriber within the GSM global network. The IMSI is burned into the SIM card when the subscriber registers with PLMN service provider. The IMSI is composed of three parts:

Mobile Country Code (MCC)

Mobile Network Code (MNC)

Mobile Subscriber Identification Number (MSIN)

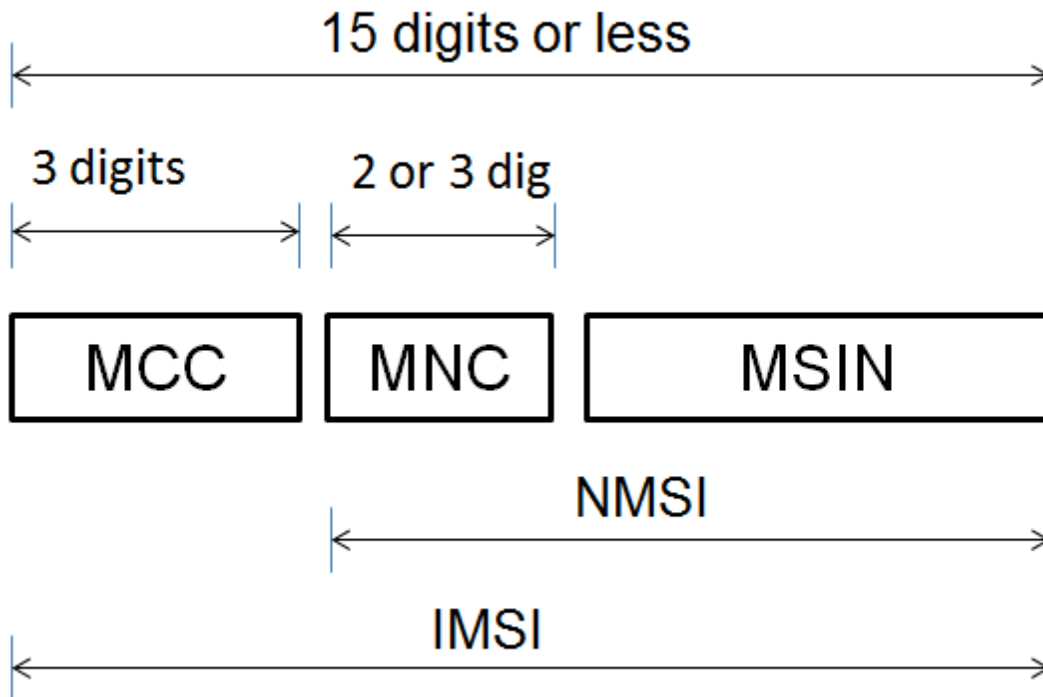


Figure 3 . IMSI Bits

1.1.7 Mobile Country Code (MCC)

This number identifies which country the subscriber's network is in. It has 3 digits.

1.1.8 Mobile Network Code (MNC)

This number identifies the home GSM PLMN of the subscriber (Wind, Vodafone, etc.). It has 2 or 3 digits. Some networks may have more than one MNC allocated to it.

Mobile Subscriber Identification Number (MSIN) - This number uniquely identifies a user within the home GSM network.

1.1.9 International Mobile Equipment Identity (IMEI)

The IMEI uniquely identifies the Mobile Equipment itself. It is essentially a serial number that is burned into the phone by the manufacturer. The IMEI is composed of three parts:

Type Allocation Code (TAC) - 8 digits

Serial Number (SNR) - 6 digits

Spare (SP) - 1 digit

1.1.10 Type Allocation Code (TAC)

This number uniquely identifies the model of a wireless device. It is composed of 8 digits. Under the new system (as of April 2004), the first two digits of a TAC are *the Reporting Body* Identifier of the GSMA approved group that allocated this model type.

1.1.11 Serial Number (SNR)

This number is a manufacturer defined serial number for the model of wireless device.

1.1.12 Spare (SP)- This number is a check digit known as a Luhn Check Digit. It is omitted during transmission within the GSM network.

1.2 Network Architecture

1.2.1 Mobile Equipment (ME)

This refers to the physical phone itself. The phone must be able to operate on a GSM network. Older phones operated on a single band only. Newer phones are dual-band, triple-band, and even quad-band capable. A quad-band phone has the technical capability to operate on any GSM network worldwide. Each phone is uniquely identified by the International Mobile Equipment Identity (IMEI) numbers. The average user does not have the technical ability to change a phone's IMEI Subscriber

1.2.2 Identity Module (SIM)

The SIM is a small smart card that is inserted into the phone and carries information specific to the subscriber, such as IMSI, TMSI, Ki (used for encryption), Service Provider Name (SPN), and Local Area Identity (LAI). The SIM can also store phone numbers (MSISDN) dialed and received, the Kc (used for encryption), phone books, and data for other applications. A SIM card can be removed from one phone, inserted into another GSM capable phone and the subscriber will get the same service as always. Each SIM card is protected by a 4-digit Personal Identification Number (PIN). In order to unlock a card, the user must enter the PIN. If a PIN is entered incorrectly three times in a row, the card blocks itself and can not be used. It can only be unblocked with an 8-digit Personal Unblocking Key (PUK), which is also stored on the SIM card.

1.2.3 Base Transceiver Station (BTS)

The BTS is the Mobile Station's access point to the network. It is responsible for carrying out radio communications between the network and the MS. It handles speech encoding, encryption, multiplexing (TDMA), and modulation/demodulation of the radio signals. It is also capable of frequency hopping. A BTS will have between 1 and 16 Transceivers (TRX), depending on the geography and user demand of an area. Each TRX represents one ARFCN. One BTS usually covers a single 120 degree sector of an area. Usually a tower with 3 BTSs will accommodate all 360 degrees around the tower. However, depending on geography and user demand of an area, a cell may be divided up into one or two sectors, or a cell may be serviced by several BTSs with redundant sector coverage. A BTS is assigned a Cell Identity. The cell identity is 16-bit number (double octet) that identifies that cell in a particular Location Area. The cell identity is part of the Cell Global Identification (CGI), which is discussed in the section about the Visitor Location Register (VLR). 120 ° Sector The interface between the MS and the BTS is known as the Um Interface or the Air Interface.

1.2.4 Base Station Controller (BSC)

The BSC controls multiple BTSs. It handles allocation of radio channels, frequency administration, power and signal measurements from the MS, and handovers from one BTS to another (if both BTSs are controlled by the same BSC). A BSC also functions as a "funneler". It reduces the number of connections to the Mobile Switching Center (MSC) and allows for higher capacity connections to the MSC

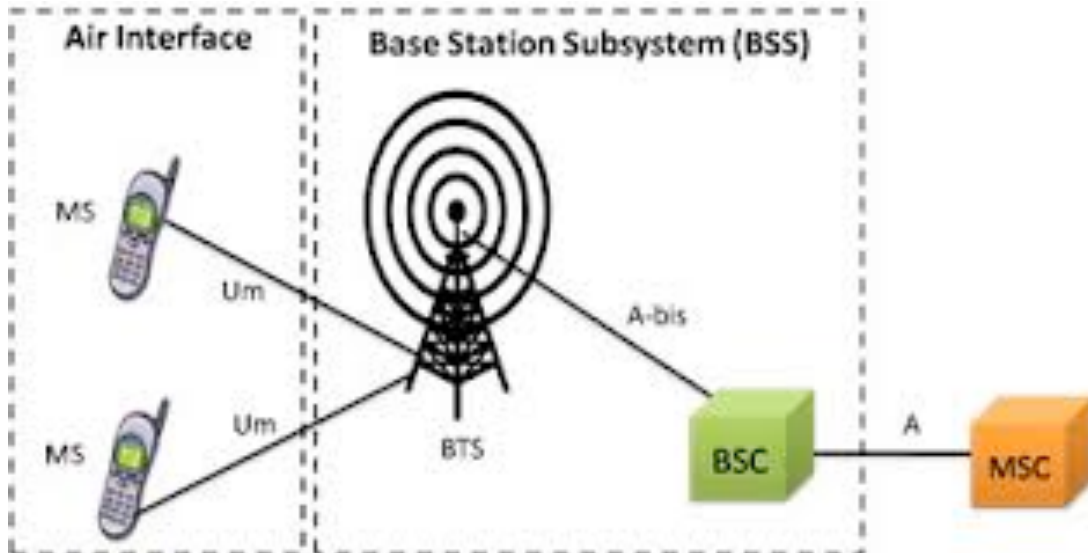


Figure 4 . Gsm Arhitecture 1st part 1

The interface between the BTS and the BSC is known as the Abis Interface. The Base Transceiver Station (BTS) and the Base Station Controller (BSC) together make up the Base Station System (BSS)

1.2.4 Mobile Switching Center (MSC)

Mobile Switching Center (MSC) - The MSC is the heart of the GSM network. It handles call routing, call setup, and basic switching functions. An MSC handles multiple BSCs and also interfaces with other MSC's and registers. It also handles in-BSC handoffs as well as coordinates with other MSC's for inter-MSC handoffs. The interface between the BSC and the MSC is known as the A Interface.

1.2.5 Gateway Mobile Switching Center (GMSC)

There is another important type of MSC, called a Gateway Mobile Switching Center (GMSC). The GMSC functions as a gateway between two networks. If a mobile subscriber wants to place a call to a regular landline, then the call would have to go through a GMSC in order to switch to the Public Switched Telephone Network

(PSTN). The interface between two Mobile Switching Centers (MSC) is called the E Interface

1.2.6 Home Location Register (HLR)

The HLR is a large database that permanently stores data about subscribers. The HLR maintains subscriber-specific information such as the MSISDN, IMSI, current location of the MS, roaming restrictions, and subscriber supplemental features. There is logically only one HLR in any given network, but generally speaking each network has multiple physical HLRs spread out across its network.

1.2.7 Visitor Location Register (VLR)

The VLR is a database that contains a subset of the information located on the HLR. It contains similar information as the HLR, but only for subscribers currently in its Location Area. There is a VLR for every Location Area. The VLR reduces the overall number of queries to the HLR and thus reduces network traffic. VLRs are often identified by the Location Area Code (LAC) for the area they service.

1.2.8 Location Area Code (LAC)

A LAC is a fixed-length code (two octets) that identifies a location area within the network. Each Location Area is serviced by a VLR, so we can think of a Location Area Code (LAC) being assigned to a VLR.

1.2.9 Location Area Identity (LAI)

An LAI is a globally unique number that identifies the country, network provider, and LAC of any given Location Area, which coincides with a VLR. It is composed of the Mobile Country Code (MCC), the Mobile Network Code (MNC), and the Location Area Code (LAC). The MCC and the MNC are the same numbers used when forming the IMSI.

1.2.10 Cell Global Identification (CGI)

The CGI is a number that uniquely identifies a specific cell within its location area, Network, and country. The CGI is composed of the MCC, MNC, LAI, and Cell Identity (CI)

1.2.11 Cell Global Identity

The VLR also has one other very important function: the assignment of a Temporary Mobile Subscriber Identity (TMSI). TMSIs are assigned by the VLR to a MS as it comes into its Location Area. TMSIs are unique to a VLR. TMSIs are only allocated when in cipher mode. The interface between the MSC and the VLR is known as the B Interface and the interface between the VLR and the HLR is known as the D Interface. The interface between two VLRs is called the G Interface

1.2.12 Equipment Identity Register (EIR)

The EIR is a database that keeps tracks of handsets on the network using the IMEI. There is only one EIR per network. It is composed of three lists. The white list, the gray list, and the black list. The black list is a list of IMEIs that are to be denied service by the network for some reason. Reasons include the IMEI being listed as stolen or cloned or if the handset is malfunctioning or doesn't have the technical capabilities to operate on the network. The gray list is a list of IMEIs that are to be monitored for suspicious activity. This could include handsets that are behaving oddly or not performing as the network expects it to. The white list is an unpopulated list. That means if an IMEI is not on the black list or on the gray list, then it is considered good and is "on the white list". The interface between the MSC and the EIR is called the F Authentication Center (Auc)

1.2.13 Authentication Center (AuC)

The AuC handles the authentication and encryption tasks for the network. The Auc stores the Ki for each IMSI on the network. It also generates crypto variables such as the RAND, SRES, and Kc. Although it is not required, the Auc is normally physically collocated with the HLR. There is one last interface that we haven't discussed. The interface between the HLR and a GMSC is called the C Interface. You will see it in the full network diagram below. This completes the introduction to the network architecture of a GSM network. Below you will find a network diagram with all of the components as well as the names of all of the interfaces. [1]

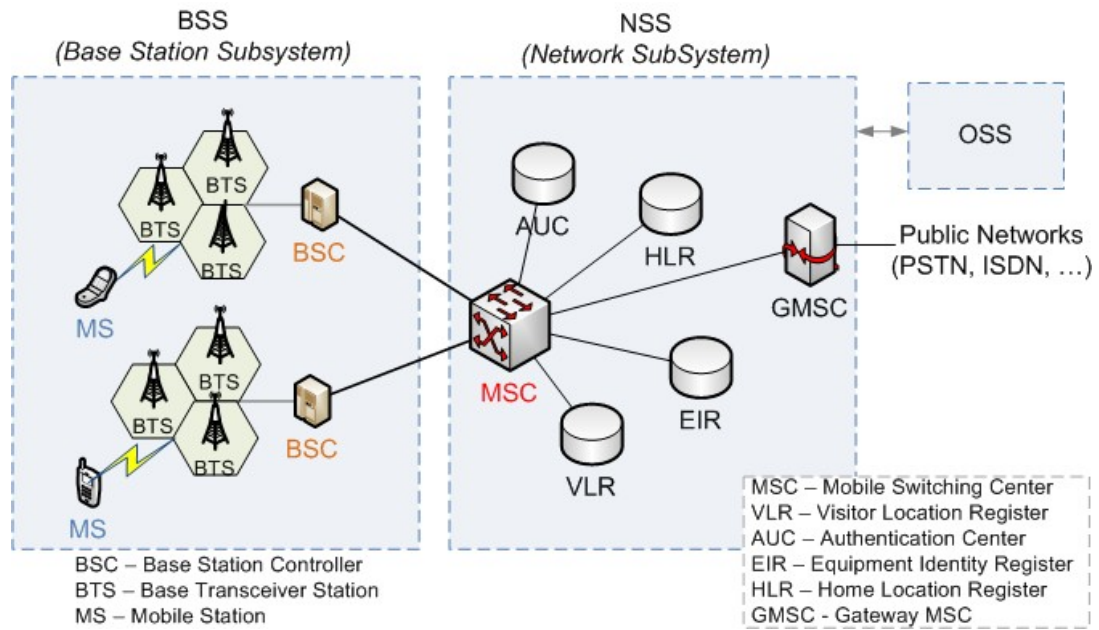


Figure 5. Gsm Architecture

Chapter 2: Spectrum Analysis Hardware

2.1 Software Defined Radio (SDR)

Traditionally radios were a hardware matter. They are often very cheap, but also very rigid. A radio created for specific transmit and receive frequencies and modulation schemes will never divert from these, unless it's hardware is modified. Please note that the word radio here is used as a generic transceiver using electro-magnetic waves for transmissions, not specifically as the device known for the reception of programmed broadcasts made by radio stations. The main idea behind Software Defined Radio (SDR), is to create very versatile transceivers by moving a lot of them, traditionally, hardware functions into the software domain. However a radio can never be purely software, because you need a way to capture and create the radio waves. Analog radio waves can be converted to digital samples using a Analog to Digital Converter (ADC) and vice versa using a Digital to Analog Converter (DAC). The ideal SDR scheme involves an antenna connected to a computer via an ADC for receiving and via a DAC for transmitting. All the processing on the signals, like (de)modulation, is then done in software, but the actual transceiving is done in the hardware subsystem. This makes for a much more adaptable system, able to for instance receive GSM signals as well as GPS and also television broadcasts by only changing something in the software. This ideal scheme however is not practically viable, because in practice ADCs and DACs are not fast enough to process a large portion of the spectrum and antennas are designed for specific frequency bands. This has led to the creation of more extended hardware subsystems for SDRs. Typically such a hardware subsystem consists of a wide band receiver that shifts a frequency band to a standard intermediate frequency, which can be sampled by ADCs and the resulting digital signal can be sent to a computer. Often other common equipment like amplifiers and band-pass filters are also a part of the hardware subsystem. One of the most versatile and widely used SDR systems is GNU Radio, mostly combined with a USRP as the hardware subsystem. [2]

2.1.1 HackRF One

One from Great Scott Gadgets is a Software Defined Radio peripheral capable of transmission or reception of radio signals from 1 MHz to 6 GHz. Designed to enable test and development of modern and next generation radio technologies, HackRF One is an open source hardware platform that can be used as a USB peripheral or programmed for stand-alone operation.



Figure 6. HackRF ONE

- 1 MHz to 6 GHz operating frequency
- Half-duplex transceiver
- Up to 20 million samples per second
- 8-bit quadrature samples (8-bit I and 8-bit Q)
- Compatible with GNU Radio, SDR#, and more
- Software-configurable RX and TX gain and baseband filter
- Software-controlled antenna port power (50 mA at 3.3 V)
- SMA female antenna connector
- SMA female clock input and output for synchronization
- Convenient buttons for programming
- Internal pin headers for expansion
- Hi-Speed USB 2.0
- USB-powered
- Open source hardware

HackRF One has an injection molded plastic enclosure and ships with a micro USB cable. An antenna is not included. ANT500 is recommended as a starter antenna for HackRF One.

HackRF One is test equipment for RF systems. It has not been tested for compliance with regulations governing transmission of radio signals. You are responsible for using your HackRF One legally. [3]

2.2 USRP

The Universal Software Radio Peripheral (USRP) is designed as a general purpose hardware subsystem for software defined radio. It is an open-hardware device developed by Matt Ettus and which can be ordered through his company Ettus Research [4]. There are currently two types: the USRP1 and the USRP2. Both consist of a motherboard which contains a Field Programmable Gate Array (FPGA), Programmable Gain Amplifier (PGA), ADC(s), DAC(s) and a communication port to connect it to the computer. Daughter boards can be plugged into the USRP motherboard according to the specific frequency bands needed. These daughter boards can be hooked up to appropriate antennas. On the receiving path (RX), a daughterboard captures the required frequency range and sends it through the PGA, possibly amplifying the signal, towards the ADC. The resulting digital signal is passed on to the FPGA, where it is transformed into 16 bit I and Q samples. These are complex samples, with the real part (Q) describing the cosine of the signal, and the imaginary part describing the sine of the signal plus 90 degrees. One sample is thus 32 bit long and can be sent to the host computer through the communication port, for further processing. The FPGA and the host CPU both do some processing on the signal, and though the exact division of labor can be changed, standard the high speed general purpose processing, like down and up conversion, decimation, and interpolation are performed in the FPGA, while waveform-specific processing, such as modulation and demodulation, are performed at the host CPU.

2.2.1 USRP1



Figure 7. USRP 1

The USRP1 has four daughterboard slots, two for receiving and two for transmitting. It contains four 12 bit ADCs (two for every receive board), that have a sampling rate of 64 Msamples per second. Nyquist's theorem states that you need a sampling rate of at least 2 times the frequency you wish to capture in order to be able to reconstruct the signal. Therefore the USRP1 can capture a bandwidth of 32 MHz at once, for every receive daughterboard. There are also four 14 bit DACs with a sampling rate of

128Msamples per second making the maximum transmit frequency band 64 MHz wide. At the heart of the USRP1 lies its FPGA, an Altera Cyclone EP1C12. This FPGA can be programmed using the Verilog hardware description language. The compiler for this can be downloaded for free from the Altera website. The communications port is a USB 2.0 chip, with a practical maximum data throughput of 32 Mbyte/s. Since the analog signals are processed into 16 bit I and Q channels, this limits the data throughput to 8 Msamples per second.

2.2.2 USRP2

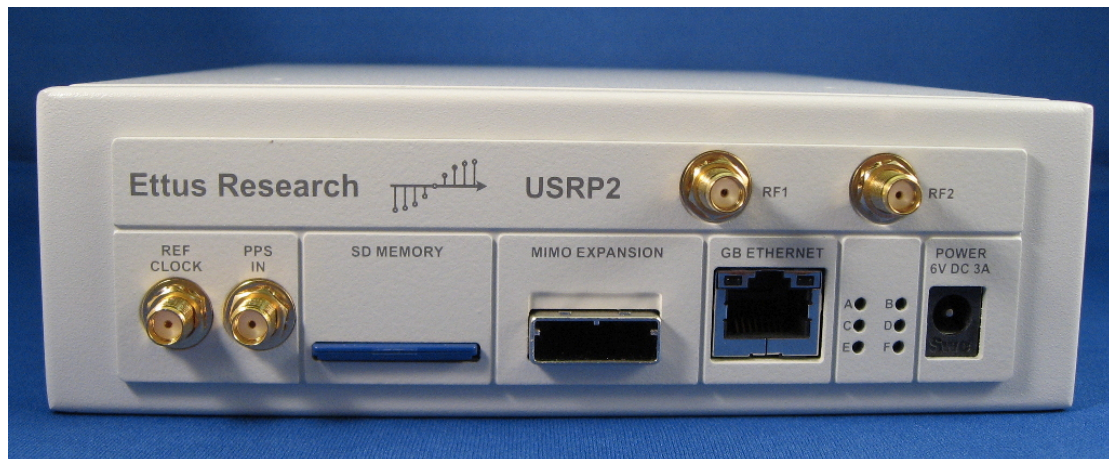


Figure 8. USRP 2

The USRP2 contains only two daughterboard slots, one transmitter and one receiver side. It does contain faster and higher resolution ADCs and DACs. Two 14 bit 100 Msamples per second ADCs and two 16-bits 400 Msamples per second DACs. So it can capture a bandwidth of 50 MHz and transmit on a bandwidth of 200 MHz wide. With respect to the USRP1, the USRP2 also contains a much faster FPGA (Xilinx Spartan 3-2000) and an ethernet port instead of the USB connection. The gigabit ethernet port allows for over 3 times higher bandwidth throughput. The USRP2 is much more costly however; double the price of an USRP1.

2.2.3 USRP Daughter boards

Different frequencies require different antennas and sometimes-different signal processing, like amplifiers or filtering, to receive or transmit correctly. So in order to keep the USRPs as general as possible the actual receiving and transmissions are handled by daughter boards that can be plugged into the USRP motherboard. These daughter boards are specifically meant for certain frequency bands. Currently there are thirteen daughter boards available, of which three are interesting in Relation to GSM signals:

- DBSRX, a 800 MHz to 2.4 GHz Receiver.
- RFX900, 800-1000MHz Transceiver, 200+mW output.

- RFX1800, 1.5-2.1 GHz Transceiver, 100+mW output.

ISSN 2348-1196 (print)

International Journal of Computer Science and Information Technology Research

ISSN 2348-120X (online)

The most used GSM frequencies are GSM900 (890.2-959.8 MHz) and GSM1800 (1710.2-1879.8 MHz) in Europe, and GSM850 (824.0-894.0 MHz) and GSM1900 (1850.0-1990.0 MHz) in America and Canada. The DBSRX board covers all these frequencies, but is only a receiver board. In order to actively transmit a RFX board is needed.[5][6].

2.3 RTL2832U Chipset - USB TV receiver

RTL2832U DVB-T SDR. This reuses very cheap USB TV receivers as SDR receivers. It is this ultra-low-cost SDR that we are going to have a description here.



Figure 9. RTL2832U Chipset - USB TV

Two key points make ultra-low-cost SDR possible.

Firstly it was discovered that certain chipsets widely used in USB TV receivers had a much wider tuning range than was needed for TV. The chipsets can also send the intermediate frequency I and Q samples directly over USB to the host computer.

Secondly it was found that the processing power on normal PCs was sufficient to perform, in real time, SDR functions on the I and Q samples coming from the USB receiver.

We bought a dongle based on the RTL2832U+R820T chipset from Ebay for 12.50 Euro.. So what does a cheap USB TV receiver like this provide in terms of performance? According to this very useful stream of consciousness this chipset can tune from 24 – 1700 MHz. That covers FM, ham radio and GSM. It uses a 3.57 MHz intermediate frequency and has a tuning error of perhaps 30 ppm, which is relatively stable for a particular dongle when it is warm.

The intermediate frequency sampling is 8-bit and around 2MS/s is an achievable sampling rate. The dynamic range is about 45 dB. My experience is that the biggest problem is various spurious signals appearing that seem to be primarily due to interference at the intermediate frequency. There are various homebrew solutions to improve screening described on the web but I haven't tried these.

My 1st experiment was using SDR running on Windows. This was easy to install using lot of information that we found on the web. We just hooked up a few meters of wire to use as an antenna. I was easily able to scan the FM and air bands to receive various stations. Once the "Correct IQ" option was checked I was able to receive sounds. I also looked around 950 MHz and saw what I thought were probably GSM cells.

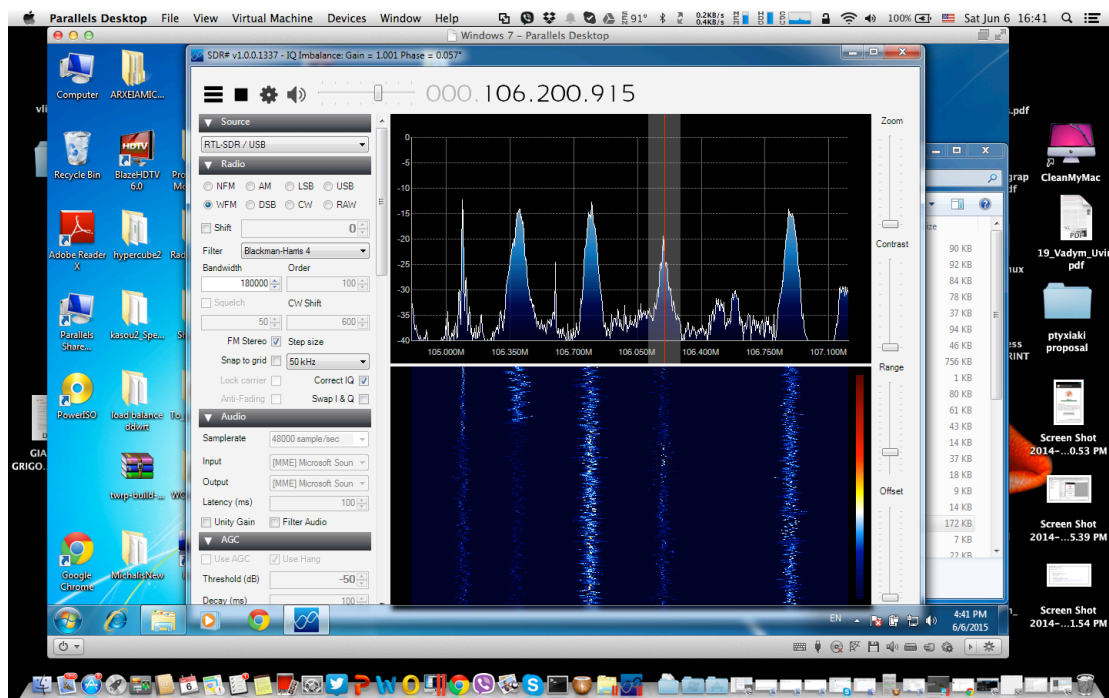


Figure 10 . Radio FM spectrum using SDR 1

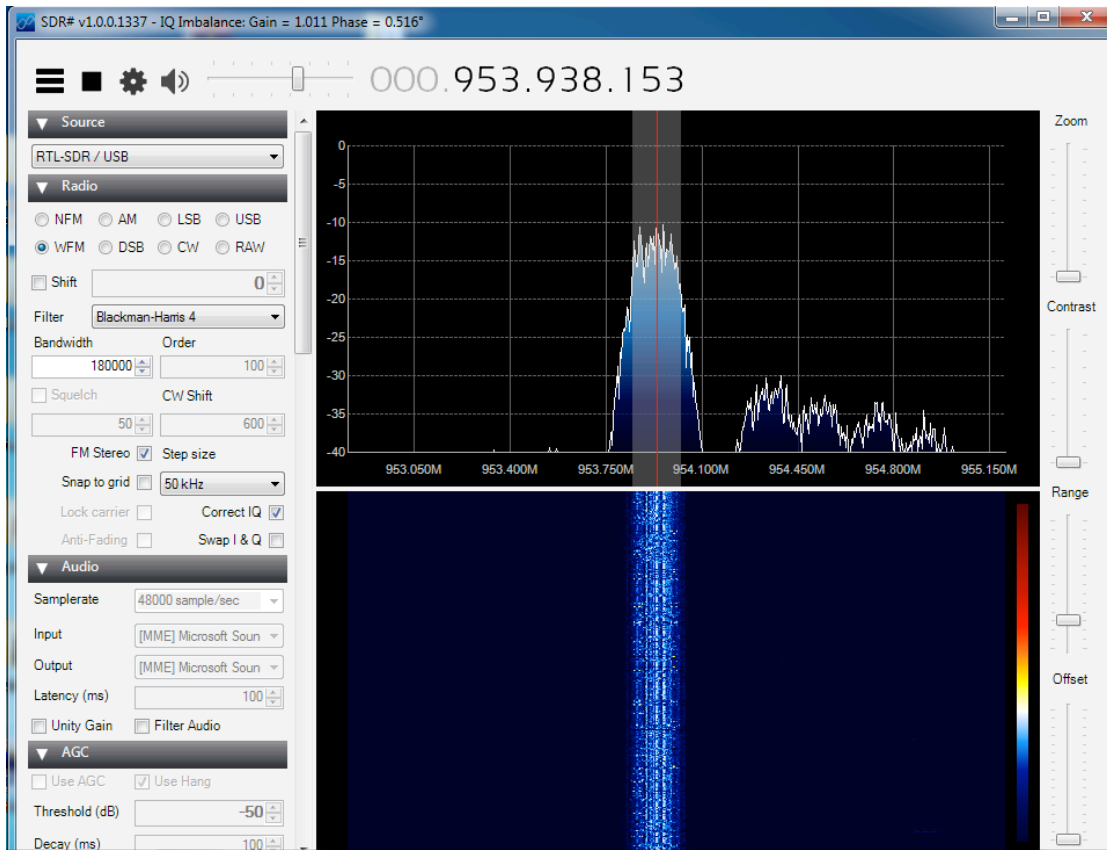


Figure 11. GSM Spectrum using SDR

Though SDR# is easy to use I think the Windows environment is fairly limited for SDR [7]

Chapter 3. Spectrum Analysis Software

3.1 Gqrx

[8] Gqrx is a software defined radio receiver powered by the GNU Radio SDR framework and the Qt graphical toolkit.

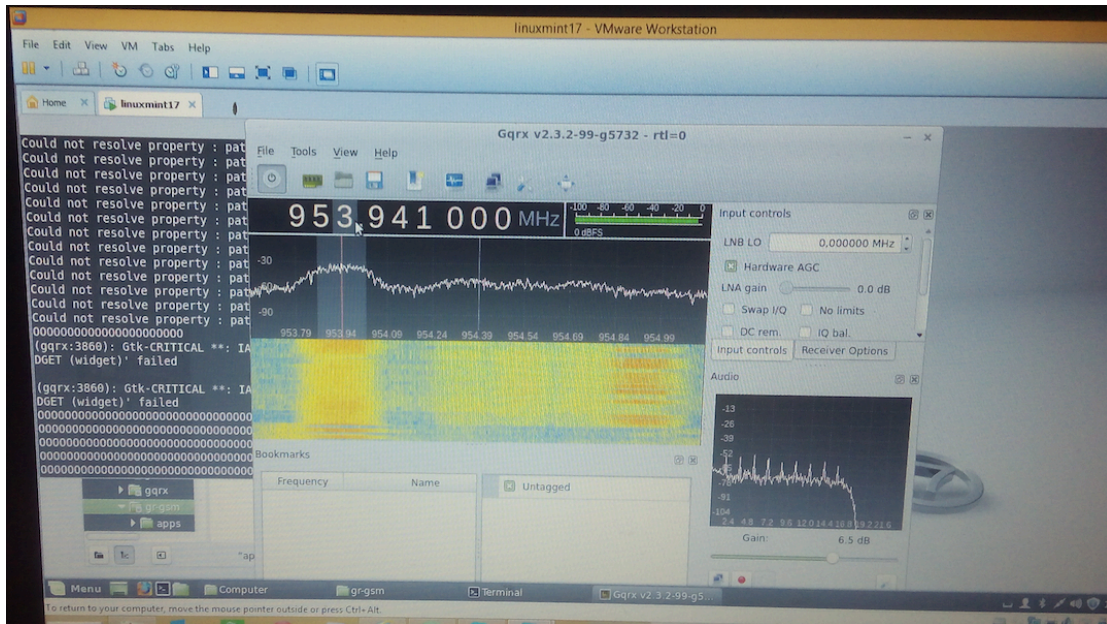


Figure 12 . Gqrx

Gqrx supports many of the SDR hardware available, including Funcube Dongles, Rtl-sdr, HackRF and USRP devices.

Gqrx is free and hacker friendly software. It comes with source code licensed under the GNU General Public license allowing anyone to fix and modify it for whatever use.

The latest stable version of Gqrx is 2.3, it is available for Linux, FreeBSD and Mac and it offers the following features:

- Discover devices attached to the computer.
- Process I/Q data from the supported devices.
- Change frequency, gain and apply various corrections (frequency, I/Q balance).
- AM, SSB, FM-N and FM-W (mono and stereo) demodulators.
- Special FM mode for NOAA APT.
- Variable band pass filter.
- AGC, squelch and noise blankers.
- FFT plot and waterfall.
- Record and playback audio to / from WAV file.
- Spectrum analyzer mode where all signal processing is disabled.
- Basic remote control through TCP connection.
- Streaming audio output over UDP

3.2 AirProbe

AirProbe [9] is the new home of the former GSM-Sniffer project. The goal is to build an air-interface analysis tool for the GSM (and possible later 3G) mobile phone standard. The prime motivation is to learn the details of the technology, help people who develop other open GSM technology (like OpenBTS, OpenMoko BS11/OpenBSC and others) and demonstrate the insecurity of the current standard. General information about the project can be found in the Wiki. Source code is in the git.

Get it using

```
$ git clone https://github.com/ksnieck/airprobe
```

Take a look at the Roadmap for current Milestones that need your contribution. Feel free to generate tickets for these Milestones and work on them.

Structure

AirProbe is divided into three main subprojects: Acquisition, Demodulation and Analysis.

Acquisition

The Acquisition module is hardware dependent and contains everything that has to do with receiving and digitizing the air interface. This is the part that needs to be rewritten for different receiver hardware, so it should be kept small and limited to the necessary functions. Most parts should be inherited from GNURadio, to keep workload limited.

DeModulation

The Demodulation module contains all necessary code to make bits out of the signal captured by Acquisition. It is in principle hardware independent, but should be open to use DSPs is desired.

Analysis

This module contains all the protocol parsing and decoding. Wireshark can be used to handle parts of the visualisation and UI tasks. An important part of the Analysis module is non-realtime A5DeCryption based on a generic fast CPU. Realtime or near-realtime A5 decrytion is not a goal of the project. For purposes of protocol analysis and demonstration of insecurities, non-realtime decryption is sufficient.

3.3 GNU Radio

3.3.1 Introduction

GNU Radio [10] is a free & open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

GNU Radio is licensed under the GNU General Public License (GPL) version 3. All of the code is copyright of the Free Software Foundation.

3.3.2 Using GNU

Radio Companion GNU Radio Companion (GRC) is a graphical user interface that allows you to build GNU Radio flow graphs

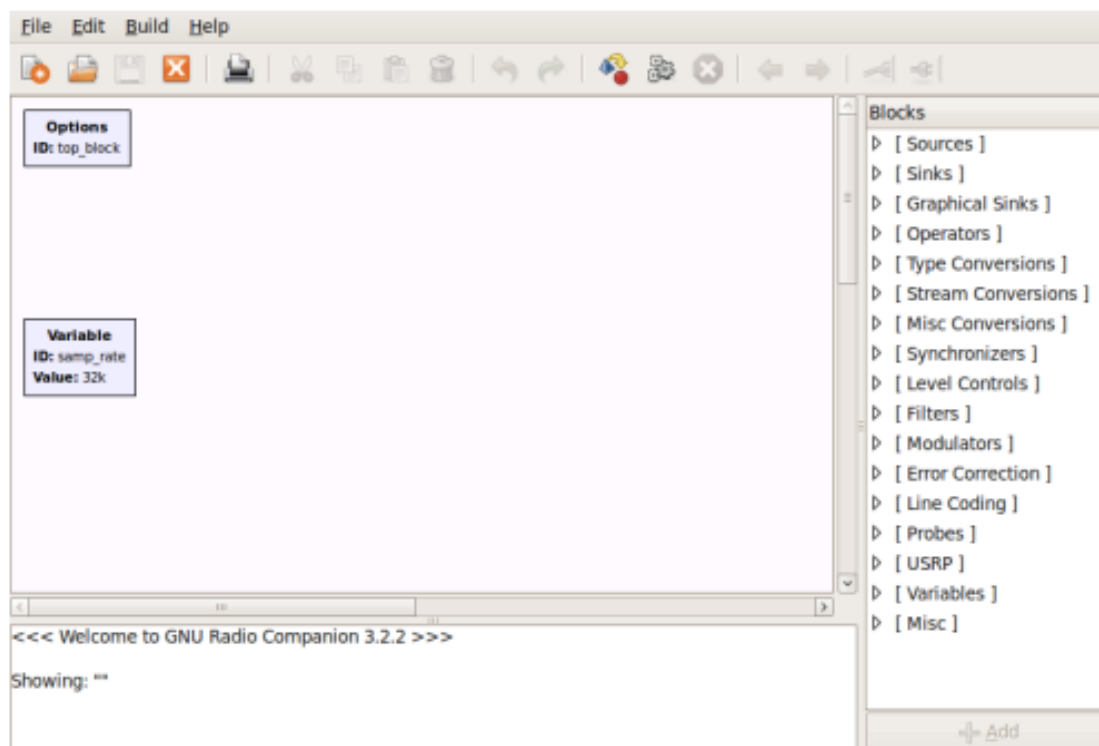


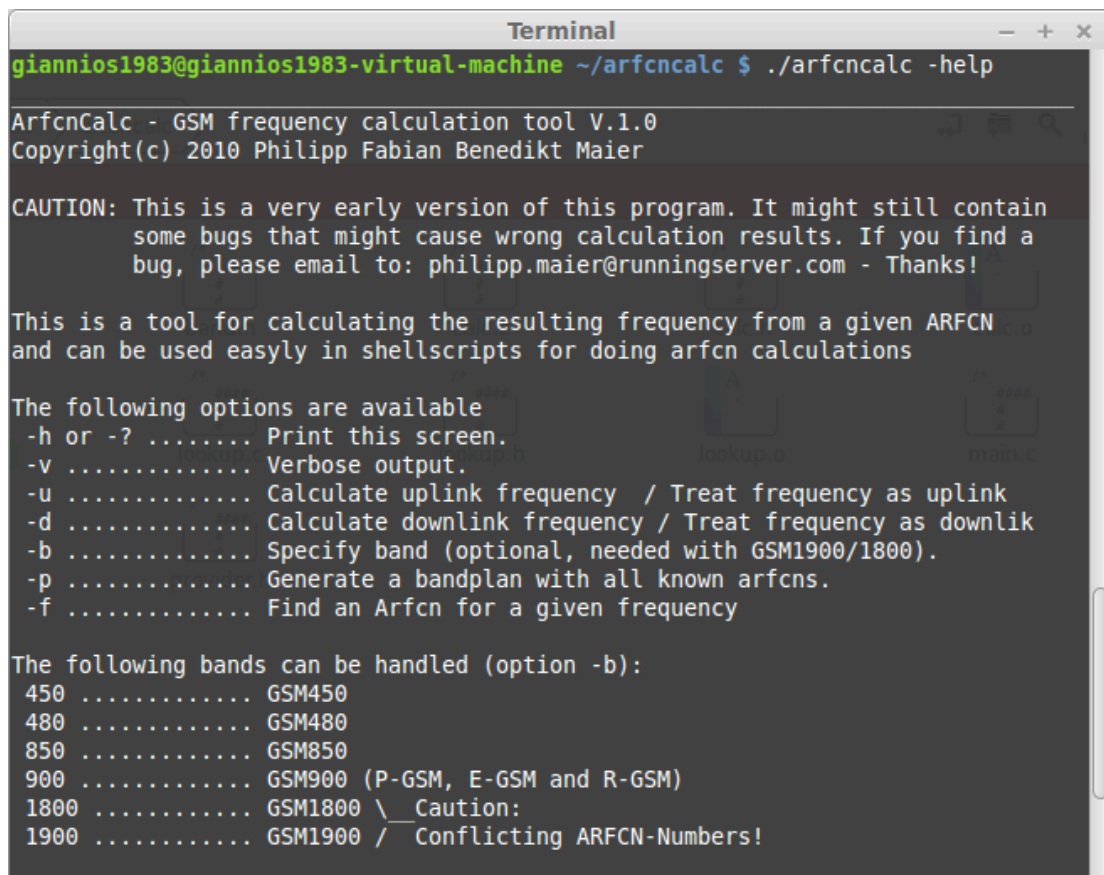
Figure 13 . Gnu-radio develop Environment

3.4 ArfcnCalc - GSM frequency calculation tool V.1.0

Arfncalc is a program for the calculation of frequencies and channels in GSM-based mobile networks. It can be calculated the basis of channel numbers uplink and downlink frequency. It is also possible to determine from a known frequency channel number a very handy feature when you're sitting on the spectrum analyzer and just want to know at what channel the observed frequency belongs. Installed in the Program is also a function with which it is possible to spend a full band plan and any additional information. Arfncalc is suitable for the calculation of frequencies and channel numbers in the bands: GSM450, GSM480, GSM850 GSM900 (P-GSM, GSM and E-R-GSM) and GSM1800 GSM1900.

To install the tool

- 1) download and extract <http://www.runningserver.com/software/arfncalc.tar>
- 2) tar -xf arfncalc.tar
- 3) cd arfncalc
- 4) rm *.o
- 5) make the install like a linux program



```
Terminal
giannios1983@giannios1983-virtual-machine ~/arfncalc $ ./arfncalc -help
ArfcnCalc - GSM frequency calculation tool V.1.0
Copyright(c) 2010 Philipp Fabian Benedikt Maier

CAUTION: This is a very early version of this program. It might still contain
         some bugs that might cause wrong calculation results. If you find a
         bug, please email to: philipp.maier@runningserver.com - Thanks!

This is a tool for calculating the resulting frequency from a given ARFCN
and can be used easily in shellscripts for doing arfcn calculations

The following options are available
-h or -? ..... Print this screen.
-v ..... Verbose output.
-u ..... Calculate uplink frequency / Treat frequency as uplink
-d ..... Calculate downlink frequency / Treat frequency as downlik
-b ..... Specify band (optional, needed with GSM1900/1800).
-p ..... Generate a bandplan with all known arfcns.
-f ..... Find an Arfcn for a given frequency

The following bands can be handled (option -b):
450 ..... GSM450
480 ..... GSM480
850 ..... GSM850
900 ..... GSM900 (P-GSM, E-GSM and R-GSM)
1800 ..... GSM1800 \ Caution:
1900 ..... GSM1900 / Conflicting ARFCN-Numbers!
```

```

giannios1983@giannios1983-virtual-machine ~/arfcnCalc $ arfcnCalc -a 100 -b 900
-d
955000000
giannios1983@giannios1983-virtual-machine ~/arfcnCalc $ arfcnCalc -a 100 -v

ArfcnCalc - GSM frequency calculation tool V.1.0
Copyright(c) 2010 Philipp Fabian Benedikt Maier

CAUTION: This is a very early version of this program. It might still contain
          some bugs that might cause wrong calculation results. If you find a
          bug, please email to: philipp.maier@runningserver.com - Thanks!

* Arfn: 100
* Downlink frequency is: 955000000 Hz
* Uplink frequency is: 910000000 Hz
* Distance: 45000000 Hz
* Offset: 0
* Licensed to T-Mobile (262 01) in Germany
* Licensed to T-Mobile Austria (232 03) in Austria
* Licensed to Sunrise (228 02) in Switzerland
* Licensed to o2 (234 10) in England
* Band: GSM900 (P-GSM, E-GSM or R-GSM)
* Country(s): EUROPE, BRAZIL, GUATEMALA, EL SALVADOR

giannios1983@giannios1983-virtual-machine ~/arfcnCalc $ arfcnCalc -f 955000000 -
d
100
giannios1983@giannios1983-virtual-machine ~/arfcnCalc $ █

```

Figure 13. ArfcnCalc Commands

%arfcnCalc -a 100 -b 900 -d

On output we can see the frequency of our channel which is 955000000

%arfcnCalc -a 100 -v

On output we can see and the Uplink frequency

And if we have the frequency and we want to translate to Arfn

% arfcnCalc -f 955000000 -d

3.5 Kalibrate Tool

Kalibrate [11] (kal) can scan for GSM base stations in a given frequency band and can use those GSM base stations to calculate the local oscillator frequency offset. My dongles drift about 8ppm from cold start to warm up after around 40 minutes.

```
giannios1983@giannios1983-virtual-machine ~/Desktop/ptyxiaki/kalibrate-rtl/src $
sudo kal -h
[sudo] password for giannios1983:
kalibrate v0.4.1-rtl, Copyright (c) 2010, Joshua Lackey
modified for use with rtl-sdr devices, Copyright (c) 2012, Steve Markgraf
Usage:
  GSM Base Station Scan:
    kal <-s band indicator> [options]
  Clock Offset Calculation:
    kal <-f frequency | -c channel> [options]
Where options are:
  -s  band to scan (GSM850, GSM-R, GSM900, EGSM, DCS, PCS)
  -f  frequency of nearby GSM base station
  -c  channel of nearby GSM base station
  -b  band indicator (GSM850, GSM-R, GSM900, EGSM, DCS, PCS)
  -g  gain in dB
  -d  rtl-sdr device index
  -e  initial frequency error in ppm
  -v  verbose
  -D  enable debug messages
```

Figure 14. ArfcnCalc Help Envirment

See the list of options below.

We have the 'Kalibrate' files on my virtual machine Linux Mint 16 in the folder named 'kalibrate-rtl')

That will give you the:

Frequency correction (ppm)

Where options are:

- s band to scan (GSM850, GSM900, EGSM, DCS, PCS)
- f frequency of nearby GSM base station
- c channel of nearby GSM base station
- b band indicator (GSM850, GSM900, EGSM, DCS, PCS)
- R side A (0) or B (1), defaults to B
- A antenna TX/RX (0) or RX2 (1), defaults to RX2

- g gain as % of range, defaults to 45%
- F FPGA master clock frequency, defaults to 52MHz
- v verbose
- D enable debug messages
- h help

Use this command to find a GSM900 signal in your area.

```

giannios1983@giannios1983-virtual-machine ~/Desktop/ptyxiaki/kalibrate-rtl/src $
sudo kal -s 900
Found 1 device(s):
 0: Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
kal: Scanning for GSM-900 base stations.
GSM-900:
chan: 37 (942.4MHz + 20.068kHz) power: 42846.17
chan: 54 (945.8MHz + 19.301kHz) power: 59409.65
chan: 75 (950.0MHz + 18.714kHz) power: 30452.17
chan: 82 (951.4MHz + 19.380kHz) power: 91401.32
chan: 91 (953.2MHz + 19.273kHz) power: 34185.35
chan: 100 (955.0MHz + 19.135kHz) power: 126798.84

```

Figure 15 . Kalibrate Band Scan Command

The search found channel 136 to be the strongest on that antenna. Then, once you have identified a GSM signal in your area, run calibrate using the command below.

```

giannios1983@giannios1983-virtual-machine ~/Desktop/ptyxiaki/kalibrate-rtl/src $
sudo kal -c 100
Found 1 device(s):
 0: Generic RTL2832U OEM

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Exact sample rate is: 270833.002142 Hz
kal: Calculating clock frequency offset.
Using GSM-900 channel 100 (955.0MHz)
average [min, max] (range, stddev)
+ 19.113kHz [19102, 19128] (26, 6.180073)
overruns: 0
not found: 0
average absolute error: -20.014 ppm

```

Figure 16. Kalibrate Channel Command

Using channel 100, the 'Frequency correction offset' rounds up to 20 ppm.

For the above command:

GSM Channel 100 (-c 100l)

3.5 Wireshark

Wireshark [12] is a free and open-source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.[4]Wireshark is cross-platform, using the GTK+ widget toolkit in current releases, and Qt in the development version, to implement its user interface, and using pcap to capture packets; it runs on Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License. Wireshark is very similar to tcpdump, but has a graphical front-end, plus some integrated sorting and filtering options. Wireshark lets the user put network interface controllers that support promiscuous mode into that mode, so they can see all traffic visible on that interface, not just traffic addressed to one of the interface's configured addresses and broadcast/multicast traffic. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic. Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering[citation needed].

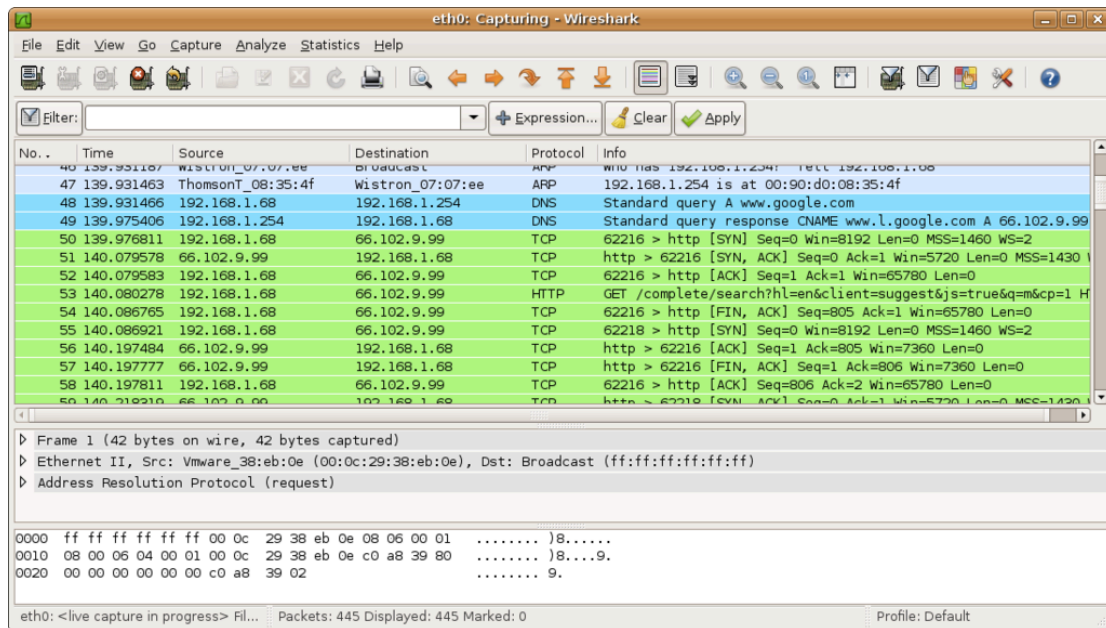


Figure 17. Wireshark Sniffing Environment

If a remote machine captures packets and sends the captured packets to a machine running Wireshark using the TZSP protocol or the protocol used by OmniPeek, Wireshark dissects those packets, so it can analyze packets captured on a remote machine at the time that they are captured.

Chapter 4. ANALYZING GSM WITH AIRPROBE AND WIRESHARK

The RTL-SDR software defined radio is been used to analyze cellular phone GSM spectrum , using Linux based tools Airprobe and Wireshark . This tutorial shows how I set up these tools for use with the RTL-SDR.

Here is a screenshot showing an example of the type of data you can receive. You can see the encrypted GSM packet information. You will not be able to see any sensitive information like voice or text message data since that part is encrypted. Decryption is not covered in this tutorial.

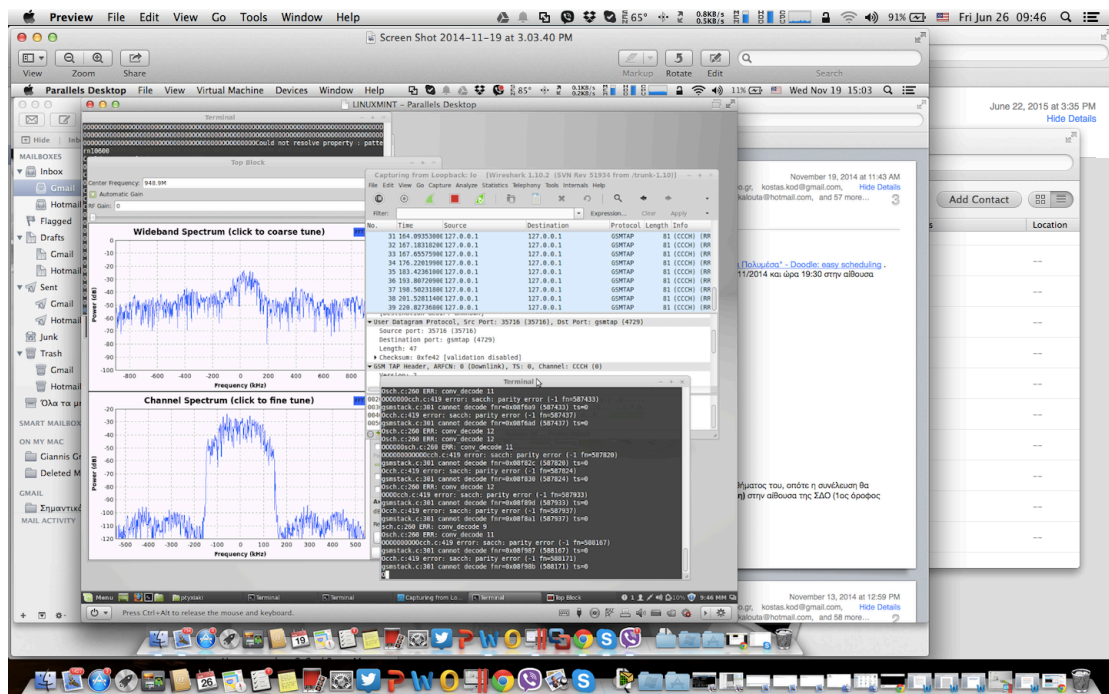


Figure 18 . Sniffing Packets from UM Air And Analyzed in Wireshark

First, you will need to find out at what frequencies you have GSM signals in your area. For most of the world, the basic GSM band is 900 MHz, in America it begin from 850 MHz. If we have an E4000 RTL-SDR, you may also find GSM signals in 1800 MHz band for most of the entire world, and 1900 MHz band for the America Open up SDRSharp, and scan around the 900 MHz or 850 MHz band for signals that looks like the waterfall . This is a non-hopping GSM downlink signal. Using NFM, it will heard something like noise. Note down the strongest GSM frequencies we can find.

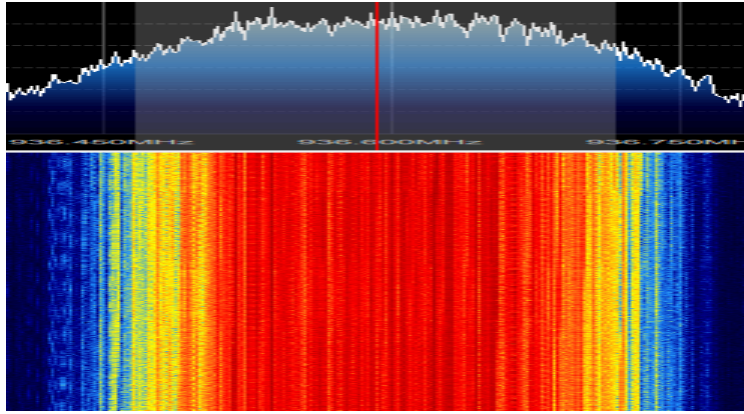


Figure 19 . Gsm frequency Spectrum

The rest of the tutorial is performed in Linux and we suppose that we have basic Linux knowledge to use the terminal window. For this Master Thesis we used Kali Linux in a VMWare session ,Linux Mint 16 64 bit and a third virtual machine running Linux Mint 17 and the most of the tools are set it up with Pypombs. Note that virtual box is reported not to work well with the RTL-SDR, as its USB bandwidth capabilities are poor, so VMWare player should be used.

4.1 Install GNU Radio

There are two ways to install GNU Radio: either by using pre-compiled binary packages, or manually compiling it from source. The recommended way to install GNU Radio is via your distribution's package manager, as described in the **standard installation guide**. If, however, you need the latest version or are planning to make changes to the GNU Radio core itself, you will want to install from source.

Note: We have recently begun releasing the 3.7 versions of GNU Radio. This marks a significant change in both the capabilities and structure of the GNU Radio code. Most out-of-tree (OOT) projects have been built around the older style API (3.6 or earlier). Most projects are updating themselves to the new 3.7 to be compatible with the most current and recommended releases. Some, however, are not. If you are looking to use an existing OOT project, check to see if it has been updated to the 3.7 API to determine which version you require.

We are moving to use PyBOMBBS as our installation tool for GNU Radio. This will take care of dependencies and allow you to easily install out-of-tree projects. It is similar to Python's 'pip' or PHP's 'pear' programs.

This is a new system that is mostly tested on Redhat/Fedora and Debian/Ubuntu. We welcome bug reports, patches, and help with improving it's capabilities for other operating systems.

The **build-gnuradio** is an install script for recent Fedora and Ubuntu systems provided by Marcus Leech.

The build-gnuradio script has a number of options that we can use to install different versions of GNU Radio. Just running build-gnuradio with no additional options, it fetches/builds the latest released version from the 3.7 series

4.2 Install Libosmocore

This tool is necessary to install it before install AirProbe

4.3 Install Airprobe

Airprobe is the tool that will decode the GSM signal. We used multiple tutorials to get airprobe to install.

Note: The new version 3.7> GNU Radio can not work with AirProbe. we will need to install GNU Radio 3.6

Install gsmdecode

Install gsm-receiver

Navigate in to to the airprobe/gsm-receiver/src/python directory. First of all we have to test Airprobe on a sample GSM cfile. Get the sample cfile which we found from the hyperlik .

https://svn.berlin.ccc.de/projects/airprobe/raw-attachment/wiki/DeModulation/capture_941.8M_112.cfile

Note: The cfile link is sometimes is dead. We have mirrored the cfile on my ftp server: <ftp://gianniosnet.no-ip.com>. Place the cfile in the airprobe/gsm-receiver/src/python folder.

Now we open wireshark, by typing wireshark into a second terminal window. Wireshark is already pre-installed in Kali Linux .To other Linux distros like Linux Mint 16 and Linux Mint 17, we have to install it manually. Since Airprobe capture data to a UDP port, we must set Wireshark to capture to this. Under Start in Wireshark , first set the capture interface to lo (loopback), and then press Start. Then in the filter box, type in gsmtap. This will ensure only airprobe GSM data is been displayed .

4.4 Run Capture Data

Back in the first terminal that is in the python directory, type in

```
./go.sh capture_941.8M_112.cfile
```

If everything installed correctly, you should now be able to see the sample GSM data in wireshark.[13]

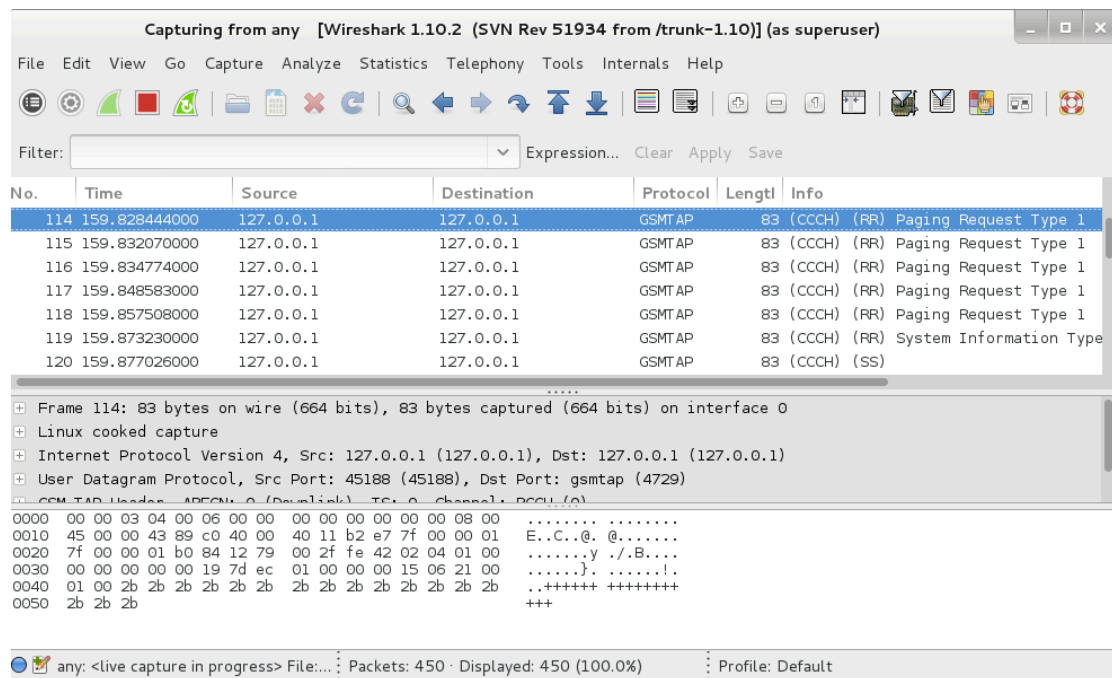


Figure 20 . Packets On Wireshark

Chapter 5 : A5/1

5.1 About A5/1

A5/1 is a stream cipher used to provide over-the-air communication privacy in the GSM cellular telephone standard. It is one of seven algorithms, which were specified for GSM use. It was initially kept secret, but became public knowledge through leaks and reverse engineering. A number of serious weaknesses in the cipher have been identified.

5.1.1 History and usage

A5/1 is used in Europe and the United States. A5/2 was a deliberate weakening of the algorithm for certain export regions. A5/1 was developed in 1987, when GSM was not yet considered for use outside Europe, and A5/2 was developed in 1989. Though both were initially kept secret, the general design was leaked in 1994 and the algorithms were entirely reverse engineered in 1999 by Marc Briceno from a GSM telephone. In 2000, around 130 million GSM customers relied on A5/1 to protect the confidentiality of their voice communications; by 2011, it was 4 billion.

Security researcher Ross Anderson reported in 1994 that "there was a terrific row between the NATO signal intelligence agencies in the mid-1980s over whether GSM encryption should be strong or not. The Germans said it should be, as they shared a long border with the Warsaw Pact; but the other countries didn't feel this way, and the algorithm as now fielded is a French design."

5.1.2 Description

The A5/1 stream cipher uses three LFSRs. A register is clocked if its clocking bit (orange) agrees with one or both of the clocking bits of the other two registers.

A GSM transmission is organised as sequences of bursts. In a typical channel and in one direction, one burst is sent every 4.615 milliseconds and contains 114 bits available for information. A5/1 is used to produce for each burst a 114 bit sequence of keystream which is XORed with the 114 bits prior to modulation. A5/1 is initialized using a 64-bit key together with a publicly known 22-bit frame number. Older fielded GSM implementations using Comp128v1 for key generation, had 10 of the key bits fixed at zero, resulting in an effective key length of 54 bits. This weakness was rectified with the introduction of Comp 128v2 which yields proper 64 bits keys. When operating in GPRS / EDGE mode, higher bandwidth radio modulation allows for larger 348 bits frames, and A5/3 is then used in a stream cipher mode to maintain confidentiality.

A5/1 is based around a combination of three linear feedback shift registers (LFSRs) with irregular clocking. The three shift registers are specified as follows:

LFSR number	Length in bits	Feedback polynomial	Clocking bit	Tapped bits
1	19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$	8	13, 16, 17, 18
2	22	$x^{22} + x^{21} + 1$	10	20, 21
3	23	$x^{23} + x^{22} + x^{21} + x^8 + 1$	10	7, 20, 21, 22

Figure 21 . Linear feedback shift registers

The bits are indexed with the least significant bit (LSB) as 0.

The registers are clocked in a stop/go fashion using a majority rule. Each register has an associated clocking bit. At each cycle, the clocking bit of all three registers is examined and the majority bit is determined. A register is clocked if the clocking bit

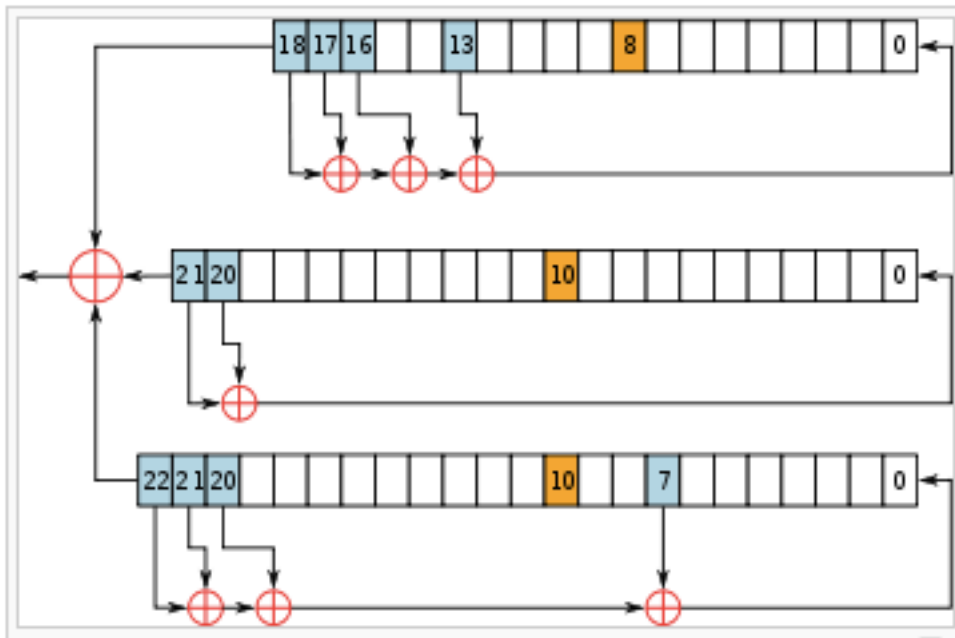


Figure 22 . A5/1 stream cipher uses 3 LFSRs

Agrees with the majority bit. Hence at each step at least two or three registers are clocked, and each register steps with probability 3/4.

Initially, the registers are set to zero. Then for 64 cycles, the 64-bit secret key is mixed in according to the following scheme: in cycle $0 < i < 64$, the i th key bit is added to the least significant bit of each register using XOR —

$$R[i] = R[i] \oplus K[i]$$

Each register is then clocked.

Similarly, the 22-bits of the frame number are added in 22 cycles. Then the entire system is clocked using the normal majority clocking mechanism for 100 cycles, with the output discarded. After this is completed, the cipher is ready to produce two 114 bit sequences of output keystream, first 114 for downlink, last 114 for uplink.

5.1.3 Security

The message on the screen of a mobile phone with the warning about lack of ciphering

A number of attacks on A5/1 have been published, and the American National Security Agency is able to routinely decrypt A5/1 messages according to released internal documents.

Some attacks require an expensive preprocessing stage after which the cipher can be broken in minutes or seconds. Until recently, the weaknesses have been passive attacks using the known plaintext assumption. In 2003, more serious weaknesses were identified which can be exploited in the ciphertext-only scenario, or by an active attacker. In 2006 Elad Barkan, Eli Biham and Nathan Keller demonstrated attacks against A5/1, A5/3, or even GPRS that allow attackers to tap GSM mobile phone conversations and decrypt them either in real-time, or at any later time.

According to professor Jan Arild Audestad, at the standardization process which started in 1982, A5/1 was originally proposed to have a key length of 128 bits. At that time, 128 bits was projected to be secure for at least 15 years. It is now estimated that 128 bits would in fact also still be secure as of 2014. Audestad, Peter van der Arend, and Thomas Haug says that the British insisted on weaker encryption, with Haug saying he was told by the British delegate that this was to allow the British secret service to eavesdrop more easily. The British proposed a key length of 48 bits, while the West Germans wanted stronger encryption to protect against East German spying, so the compromise became a key length of 56 bits.

5.1.4 Known-plaintext attacks

The first attack on the A5/1 was proposed by Ross Anderson in 1994. Anderson's basic idea was to guess the complete content of the registers R1 and R2 and about half of the register R3. In this way the clocking of all three registers is determined and the second half of R3 can be computed.

In 1997, Golic presented an attack based on solving sets of linear equations which has a time complexity of 2^{40} (the units are in terms of number of solutions of a system of linear equations which are required).

In 2000, Alex Biryukov, Adi Shamir and David Wagner showed that A5/1 can be cryptanalysed in real time using a time-memory tradeoff attack,[6] based on earlier work by Jovan Golic. One tradeoff allows an attacker to reconstruct the key in one second from two minutes of known plaintext or in several minutes from two seconds of known plain text, but he must first complete an expensive preprocessing stage which requires 248 steps to compute around 300 GB of data. Several tradeoffs

between preprocessing, data requirements, attack time and memory complexity are possible.

The same year, Eli Biham and Orr Dunkelman also published an attack on A5/1 with a total work complexity of 239.91 A5/1 clockings given 220.8 bits of known plaintext. The attack requires 32 GB of data storage after a precomputation stage of 238.

Ekdahl and Johannson published an attack on the initialisation procedure which breaks A5/1 in a few minutes using two to five minutes of conversation plaintext. This attack does not require a preprocessing stage. In 2004, Maximov et al. improved this result to an attack requiring "less than one minute of computations, and a few seconds of known conversation". The attack was further improved by Elad Barkan and Eli Biham in 2005.

5.1.5 Attacks on A5/1 as used in GSM

In 2003, Barkan et al. published several attacks on GSM encryption. The first is an active attack. GSM phones can be convinced to use the much weaker A5/2 cipher briefly. A5/2 can be broken easily, and the phone uses the same key as for the stronger A5/1 algorithm. A second attack on A5/1 is outlined, a ciphertext-only time-memory tradeoff attack which requires a large amount of precomputation.

In 2006, Elad Barkan, Eli Biham, Nathan Keller published the full version of their 2003 paper, with attacks against A5/X Ciphers. The authors claim:

We present a very practical ciphertext-only cryptanalysis of GSM encrypted communication, and various active attacks on the GSM protocols. These attacks can even break into GSM networks that use "unbreakable" ciphers. We first describe a ciphertext-only attack on A5/2 that requires a few dozen milliseconds of encrypted off-the-air cellular conversation and finds the correct key in less than a second on a personal computer. We extend this attack to a (more complex) ciphertext-only attack on A5/1. We then describe new (active) attacks on the protocols of networks that use A5/1, A5/3, or even GPRS. These attacks exploit flaws in the GSM protocols, and they work whenever the mobile phone supports a weak cipher such as A5/2. We emphasize that these attacks are on the protocols, and are thus applicable whenever the cellular phone supports a weak cipher, for example, they are also applicable for attacking A5/3 networks using the cryptanalysis of A5/1. Unlike previous attacks on GSM that require unrealistic information, like long known plaintext periods, our attacks are very practical and do not require any knowledge of the content of the conversation. Furthermore, we describe how to fortify the attacks to withstand reception errors. As a result, our attacks allow attackers to tap conversations and decrypt them either in real-time, or at any later time.

In 2007 Universities of Bochum and Kiel started a research project to create a massively parallel FPGA-based cryptographic accelerator COPACOBANA. COPACOBANA was the first commercially available solution using fast time-memory trade-off techniques that could be used to attack the popular A5/1 and A5/2 algorithms, used in GSM voice encryption, as well as the Data Encryption Standard

(DES). It also enables brute force attacks against GSM eliminating the need of large precomputed lookup tables.

In 2008, the group The Hackers Choice launched a project to develop a practical attack on A5/1. The attack requires the construction of a large look-up table of approximately 3 terabytes. Together with the scanning capabilities developed as part of the sister project, the group expected to be able to record any GSM call or SMS encrypted with A5/1, and within about 3–5 minutes derive the encryption key and hence listen to the call and read the SMS in clear. But the tables weren't released.[14]

A similar effort, the A5/1 Cracking Project, was announced at the 2009 Black Hat security conference by cryptographers Karsten Nohl and Sascha Krißler. It created the look-up tables using Nvidia GPGPUs via a peer-to-peer distributed computing architecture. Starting in the middle of September 2009, the project ran the equivalent of 12 Nvidia GeForce GTX 260. According to the authors, the approach can be used on any cipher with key size up to 64-bits.

In December 2009, the A5/1 Cracking Project attack tables for A5/1 were announced by Chris Paget and Karsten Nohl. The tables use a combination of compression techniques, including rainbow tables and distinguished point chains. These tables constituted only parts of the 2TB completed table, and had been computed during three months using 40 distributed CUDA nodes, and then published over BitTorrent.. More recently the project has announced a switch to faster ATI Evergreen code, together with a change in the format of the tables and Frank A. Stevenson announced breaks of A5/1 using the ATI generated tables.

Documents leaked by Edward Snowden in 2013 states that NSA "can process encrypted A5/1".[14].

5.2 Passive GSM interception

Global System for Mobile Communications (GSM for short) is a suite of protocols designed for second generation (2G) mobile phone communication, put simply it's used extensively around the world for making calls on your mobile. GSM comes with various security features to authenticate users on the network, it uses the SIM of the caller to authenticate the mobile device with the base station using a pre-shared key and challenge/response, once authenticated the calls are encrypted with a stream-cipher typically A5/1 for America and most of Europe, you can read more about A5/1

In previous chapter of our master thesis we show how to get this Kc of our mobile station (Samsung mobile acting like modem) or to get it , if we have the sim card of the victim SIM with a Sim card Reader and program SpySimII .This pre share key is responsible to encrypt our communication on 2G networks

While our thesis aims to show how the interceptors work for decrypt the communication , it is still no trivial task setting up the tools, like most security work on GSM the documentation seems to be lacking and help is hard to find. Because of this we've written the following guide to show how the Kraken work ready and compiled.

5.2.1 Kraken

Karsten's latest project (The A5/1 Security Project) announced this month on the 16th of July the release of 'Kraken'. Kraken is a software toolkit, which uses new encryption cracking tables to break the cipher used to secure mobile phone communication. Kraken has the potential to de-cipher a phone call in a matter of seconds. The Kraken software has been designed to run on inexpensive desktop computer equipment which brings phone snooping into the hands of the home computer geek. Kraken has been especially designed to de-cipher the A5/1 cryptographic algorithm. The A5/1 stream cipher was developed in 1987 to encrypt both voice and signalling data from a mobile telephone. A5/1 in its day was considered a strong method of keeping mobile phone calls private using 64-bit encryption, and even a watered down version of the algorithm 'A5/2' was developed to be exported outside of Europe.

Frank Stevenson, a developer within the A5/1 Security Project made the announcement of the first release of Kraken: "I have named this beast Kraken, after a Norse mythological creature capable of eating many things for breakfast. Kraken feeds of an exclusive diet of A5/1 encrypted data". He also pointed out the following hardware prerequisites needed to set up Kraken.

When Kraken was in the early stages of development, the GSM Alliance said that the research is a long way from being a practical attack on GSM. The GSMA said that they welcomed research, but continued by highlighting that "the theoretical compromise of GSM network requires the construction of a large look-up table of approximately 2 Terabytes, which is equivalent to the amount of data contained in a 20 kilometre high pile of books".

The software is regarded as a key step towards eavesdropping on mobile phone conversations over GSM networks. Since GSM networks are the backbone of 3G (or 3rd Generation of standards for mobile phones and mobile telecommunications service), even 3G phones can be compromised since when they roll back to GSM mode when a 3G network is not available.

The A5/1 Security Project have stressed that their main aim is to show how easily the A5/1 encryption can be cracked. It is anticipated that A5/1 Security Project leader Karsten Nohl will discuss the hardware and software setup during this years Black Hat Security Conference.

5.2.2 Minimum Requirements

- 1) Install Linux Machine multicore min 3GB RAM
- 2) 2TB HD partitions with the The Berlin A5/1 Rainbow table set
- 3) 2TB HD Without File System
- 4) GPU support will be added for ATI Radeon HD

5.2.3 Setup

We can compile & Install kraken like other linux programs setup but is not a easy. Have a lot patience for doing this

5.2.4 Prepare Internal 2 TB HD

This next step is to create MTHESIS by writing the rainbow tables into a partition on the INTERNAL drive, they cannot simply be copied on to the drive, they're inserted inside a partition using a special tool, this is why you need an additional 2Tb of space on top of the 2Tb INTERNAL drive. First you need to identify which mount name your 2Tb drive has, this step is crucial because if you get this wrong you'll delete the partitions on the wrong drive and if you accidentally delete the rainbow tables you'll be extremely mad.

Open terminal as root and type

```
umount /dev/sdz
```

```
parted /dev/sdz mkpart primary
```

```
partprobe /dev/sdz
```

5.2.5 Preparing tools for use

The tool to insert the tables on to the drive is called TableConvert it's not actually compiled into the binaries so we need to do that first. Open a terminal window and browse to the TableConvert

Make TableConvert

Next we need to set up the config file which will point TableConvert to the right partition to insert the tables into. You need to make a copy of the sample conf file, in your terminal window type:

```
cp tables.conf.sample tables.conf
```

Now open this new tables.conf file for editing and remove the list of example devices and replace them with your drive and partition, the partition number will be 1,

```
#Devices: dev/node max_tables  
Device: /dev/sdz1 40  
#Tables: dev id(advance) offset
```

The max tables value is set at 40 because there's 40 individual tables and we want them all on the same drive. Make sure to save changes to this file.

5.2.5 Inserting the rainbow tables into the partition

Now to start the copy, make sure if you're using a laptop the power cord is in and nothing will interrupt the copy process, this step takes a long time. In a terminal

window type:

```
cd kraken_folder/indexes
```

Now type the following replacing The "*folder/Berlin A5/1 Rainbow table set*" with the directory you've stored the rainbow tables.

```
sudo python Behemoth.py folder/Berlin A5/1 Rainbow table set
```

info pop up on the screen to tell you what table is currently being copied, you have a 10 hour wait from an internal drive

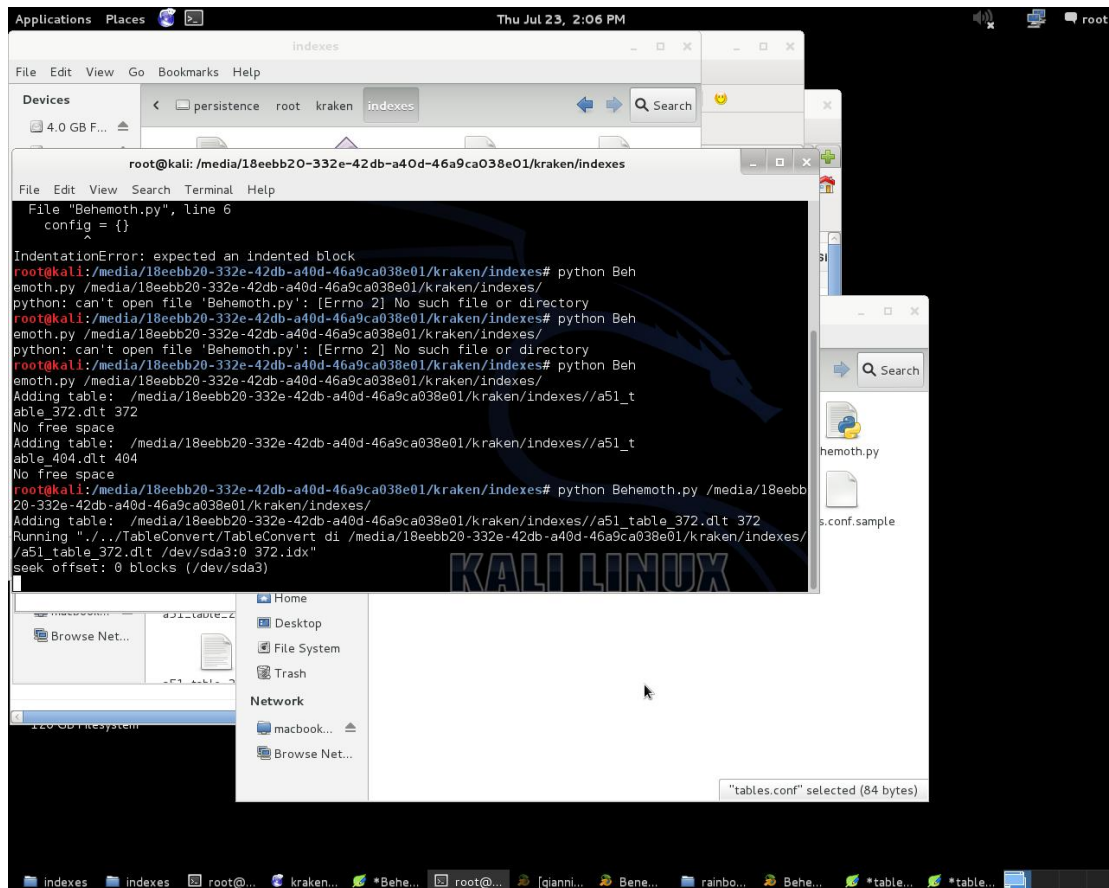


Figure 23 . Copying Rainbow Tables on Sda3 Partition

5.2.5 Testing

Lastly we test Kraken, switch to the kraken directory

```
cd /kraken_folder/kraken/Kraken
```

Run Kraken and give it the directory where the indexes are stored

```
./kraken /kraken_folder/kraken/indexes
```

Now run the test command

```
Kraken>test
```

It should run a test string of binary, a reasonable speed from an external drive is about 160 seconds on average, we get about 30 seconds but it depends on your hardware.[15]

Chapter 6 : Practical exercise on the GSM Encryption A5/1

6.1 Analyze The Capture File

Step 1: We have a captured file *.cfile is taken from srlabs because other air captured traffic from air is illegal and might have problems with the law.

Download this capture file:

```
ftp://gianniosnet.no-ip.com/  
vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile.gz
```

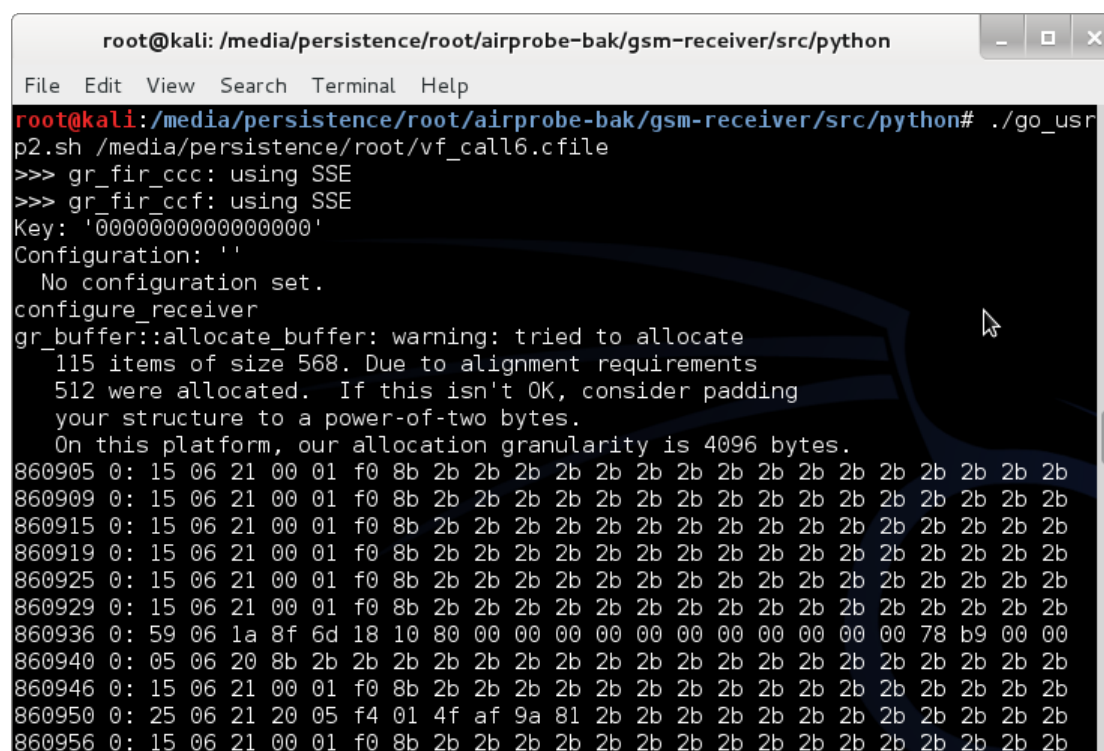
because the link from the original site it doesn't work

http://reflexor.com/vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile.gz

Extract the capture file

Step 2: Start and we listen to your network interface .Use the following capture filter “port 4729” (Hint :Ctrl + I ,then click Option Lo and enter the capture filter.)

Decode the capture file with the script cd airprobe/gsm-receiver/src/python/./go_usrp2.sh



```
root@kali: /media/persistence/root/airprobe-bak/gsm-receiver/src/python  
File Edit View Search Terminal Help  
root@kali:/media/persistence/root/airprobe-bak/gsm-receiver/src/python# ./go_usrp2.sh /media/persistence/root/vf_call6.cfile  
>>> gr_fir_ccc: using SSE  
>>> gr_fir_ccf: using SSE  
Key: '0000000000000000'  
Configuration: ''  
No configuration set.  
configure_receiver  
gr_buffer::allocate_buffer: warning: tried to allocate  
115 items of size 568. Due to alignment requirements  
512 were allocated. If this isn't OK, consider padding  
your structure to a power-of-two bytes.  
On this platform, our allocation granularity is 4096 bytes.  
860905 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860909 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860915 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860919 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860925 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860929 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860936 0: 59 06 1a 8f 6d 18 10 80 00 00 00 00 00 00 00 00 00 00 78 b9 00 00  
860940 0: 05 06 20 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860946 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860950 0: 25 06 21 20 05 f4 01 4f af 9a 81 2b 2b 2b 2b 2b 2b 2b 2b 2b  
860956 0: 15 06 21 00 01 f0 8b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b
```

Figure 26 . Decode the Capture File

We should see a lot of decoded packets in wireshark

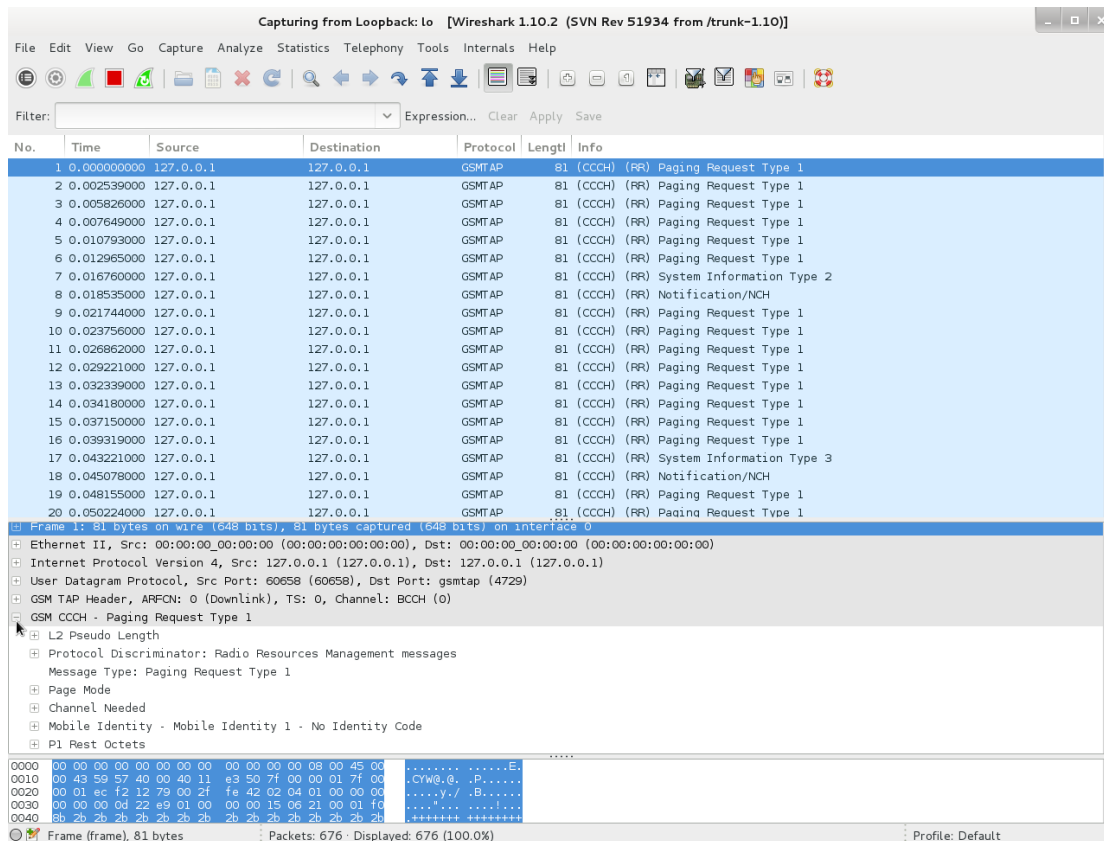


Figure 27 . Decoded Packets in Wireshark

Step 3: Search for an “Immediate Assignment” packet and take a closer look at the packet. You can see that a SDCCH/8 channel on TimeSlot 1 is assigned to a subscriber.

Step 4: Keep wireshark running and keep listening. To decode TimeSlot 1 as “SDCCH/8” pass the parameter “1S” to the decoder

```
./go_usrp2.sh vf_call6.cfile 174 1S
```

The decoder now only decodes TimeSlot 1 and ignores TimeSlot 0. Note: Because this is a USRP2 capture file, “174” has to be used as the decimation rate.

“Cx” are the encrypted burst bits, “Px” are the decrypted burst bits and “Sx” are the keystream bits (encrypted bits XOR decrypted bits). We do not decrypt right now so the decrypted burst bits are the same as the encrypted burst bits. If x is 1 then this is the first burst of a frame. The number after Cx, Px or Sx is the GSM frame number, the second number is the modified frame number as required by the A5/1 algorithm.

Step 6: Now we have to find an encrypted burst where the content of the burst is known. This step is explained in the next task.

In order to find the Kc we are using the Kraken tool. This tool uses a 2 TB rainbow table to calculate the key in a reasonable time. But before you can crack the key you have to find an encrypted burst which you also have in clear text. This step is not as easy as it might sound because normally you don't know the clear text. But with some special information it is possible to guess the clear text.

1. In the last task you decoded timeslot 1 as SDCCH/8. The unencrypted bursts are shown in wireshark. We are now going to take a closer look at the “System Information Type 5” message. This message is repeated in given time intervals. Therefore we know nearly the whole burst as cleartext. Now we only need the framecount where the same message is transmitted after activation of the encryption.

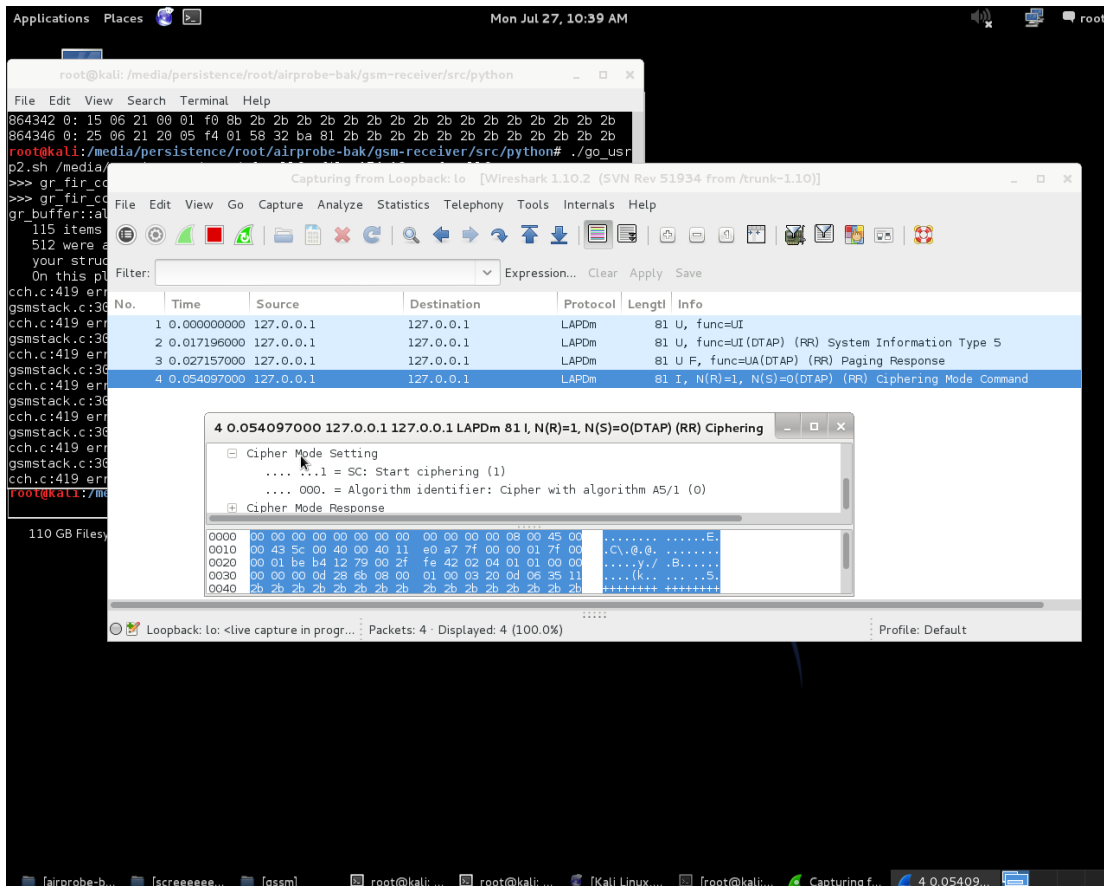


Figure 30 . System Information 5 & 6 packets

6.2 Finding Information to Crack the Key

You can use this System Information Type 5 message to crack the key. First, you need to xor the unencrypted “Px” bits for this message with the corresponding encrypted “Cx” bits for another System Information Type 5 message.

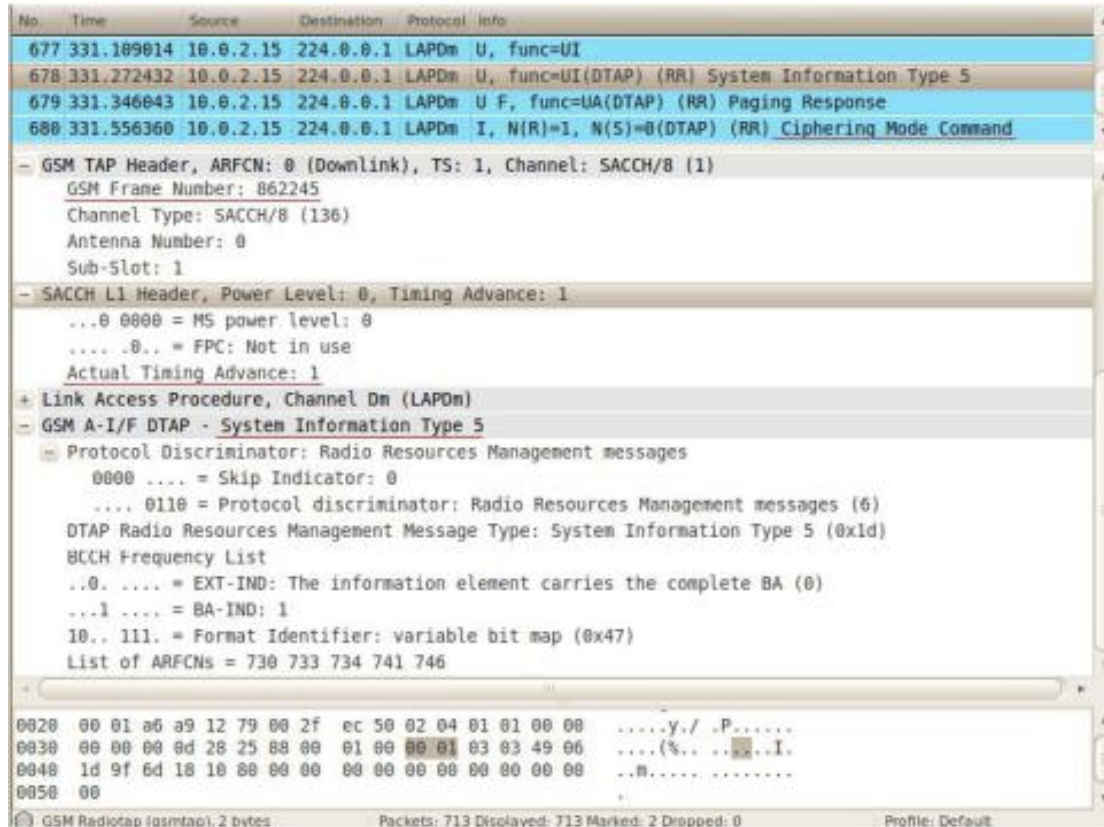


Figure 31 . Information's Of System Information Type 5

To do so we need to find the unencrypted System Information Type 5 message, which is displayed in wireshark (Hint: search for the GSM Frame Number). With in the console, look for the Frame Number. Look at the whole block that contains your frame number. Your result should consist of four “Cx / Px / Sx” lines (one C1/P1/S1 and three C0/P0/S0);

To make it easier for us , we copy the block into a textfile.(Figure 27.)

You will need the four lines that start with “Cx”.

Unfortunately the “Timing Advance” parameter from the unencrypted message has changed on the encrypted message from 1 to 0. Therefore you need to correct this before you can xor the bursts. Johann Betz has created a nice tool to do this. The tool is called

gsmframecoder1.

To change the Timing Advance parameter you need to change the bit from 1 to 0, e.g. if you want to change the Timing Advance parameter on

00 01 03 03 49 06 1d 9f 6d 18 10 80 00 00 00 00 ... from 1 to 0, you have to do it

like this `./gsmframecoder 00 00 03 03 49 06 1d 9f 6d 18 10 80 00 00 00 00 ...`
As result you will get four bit streams, denoted as Burst1 to Burst4, which are needed for the next step.

Then you need to find the encrypted System Information Type 5 message. This message can be found by adding 204 to the Frame Number. Once you have found the corresponding---gsmframecoder:

`http://www.ks.unifreiburg.de/download/misc/gsmframecoder.tar.gz`
message you need to xor the Burst1 bits from the gsmframecoder with the C1 bits from the encrypted message, Burst2 with the first C0 to Burst4 with the last C0 bits.

You can use

the xor.py script (kraken/Utilities) as follows:

```
./xor.py first_bits second_bits
```

(e.g. `./xor.py 00100000000101000.... 01001010001011111...`
Result: `01101010001110111...`)

3. The next step is to crack one of the xor streams with the kraken tool. Paste the results to kraken-server to crack the burst:

Just enter the command “crack” followed by the xor’ed key stream (Each crack takes 30 seconds to complete!):

```
Kraken>Crack  
011111011111110100111100110001000000010011100110001010011...
```

Note: You need to repeat this step with the next xor’ed “BurstX” / “Cx” combination until you find the correct position on the rainbow table.

A successful output looks like this: Found `d5eb21665d2b8f25 @ 13`. This means `ad5eb21665d2b8f25` is the key that produces the output at position 37.

4. These numbers can then be fed into the find_kc tool (kraken/Utilities). It needs a second GSM frame together with the burst data as input to eliminate wrong candidates for Kc.

```
./find_kc d5eb21665d2b8f25 13 1332451 1332352 0110101000111011111110...
```

Here the first number is the key and the second number the position on the rainbow table found by the last “crack”-step. The third number is the “modified frame number” from the encrypted burst. The fourth number is the “modified frame number” from a second encrypted burst (Cx – see output of `go_usrp2.sh`). The last number is the xor of this Cx and the corresponding bitstream from `gsmframedecode`.

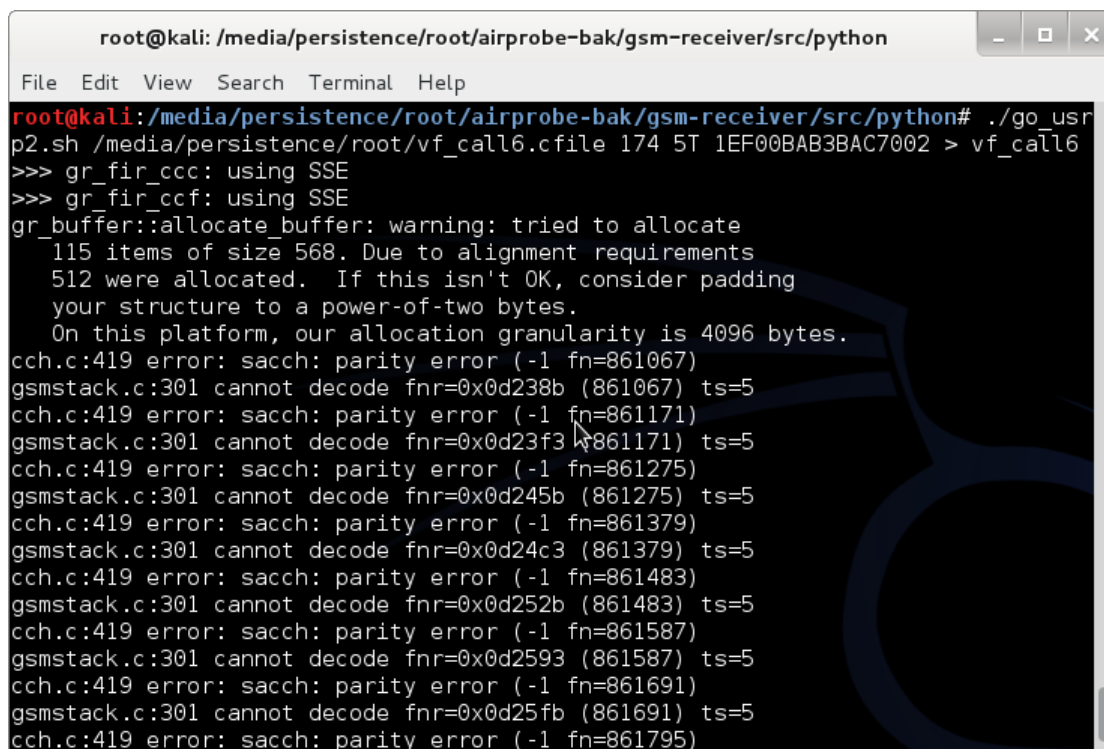
The result should look like this: KC(2): 1e f0 0b ab 3b ac 70 02 *** MATCHED ***

6.3 Decoding the channel

Now you are able to decode the encrypted part of the assigned “SDCCH/8” channel. Pass the key to the decoder like this:

Step 1 :Open terminal and run

```
./go_usrp2.sh vf_call6.cfile 174 5T 1EF00BAB3BAC7002
```



```
root@kali: /media/persistence/root/airprobe-bak/gsm-receiver/src/python
File Edit View Search Terminal Help
root@kali:/media/persistence/root/airprobe-bak/gsm-receiver/src/python# ./go_usrp
p2.sh /media/persistence/root/vf_call6.cfile 174 5T 1EF00BAB3BAC7002 > vf_call6
>>> gr_fir_ccc: using SSE
>>> gr_fir_ccf: using SSE
gr_buffer::allocate_buffer: warning: tried to allocate
 115 items of size 568. Due to alignment requirements
 512 were allocated. If this isn't OK, consider padding
 your structure to a power-of-two bytes.
 On this platform, our allocation granularity is 4096 bytes.
cch.c:419 error: sacch: parity error (-1 fn=861067)
gsmstack.c:301 cannot decode fnr=0x0d238b (861067) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861171)
gsmstack.c:301 cannot decode fnr=0x0d23f3 (861171) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861275)
gsmstack.c:301 cannot decode fnr=0x0d245b (861275) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861379)
gsmstack.c:301 cannot decode fnr=0x0d24c3 (861379) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861483)
gsmstack.c:301 cannot decode fnr=0x0d252b (861483) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861587)
gsmstack.c:301 cannot decode fnr=0x0d2593 (861587) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861691)
gsmstack.c:301 cannot decode fnr=0x0d25fb (861691) ts=5
cch.c:419 error: sacch: parity error (-1 fn=861795)
```

Figure 32 . Decode encrypted part of the assigned “SDCCH/8”

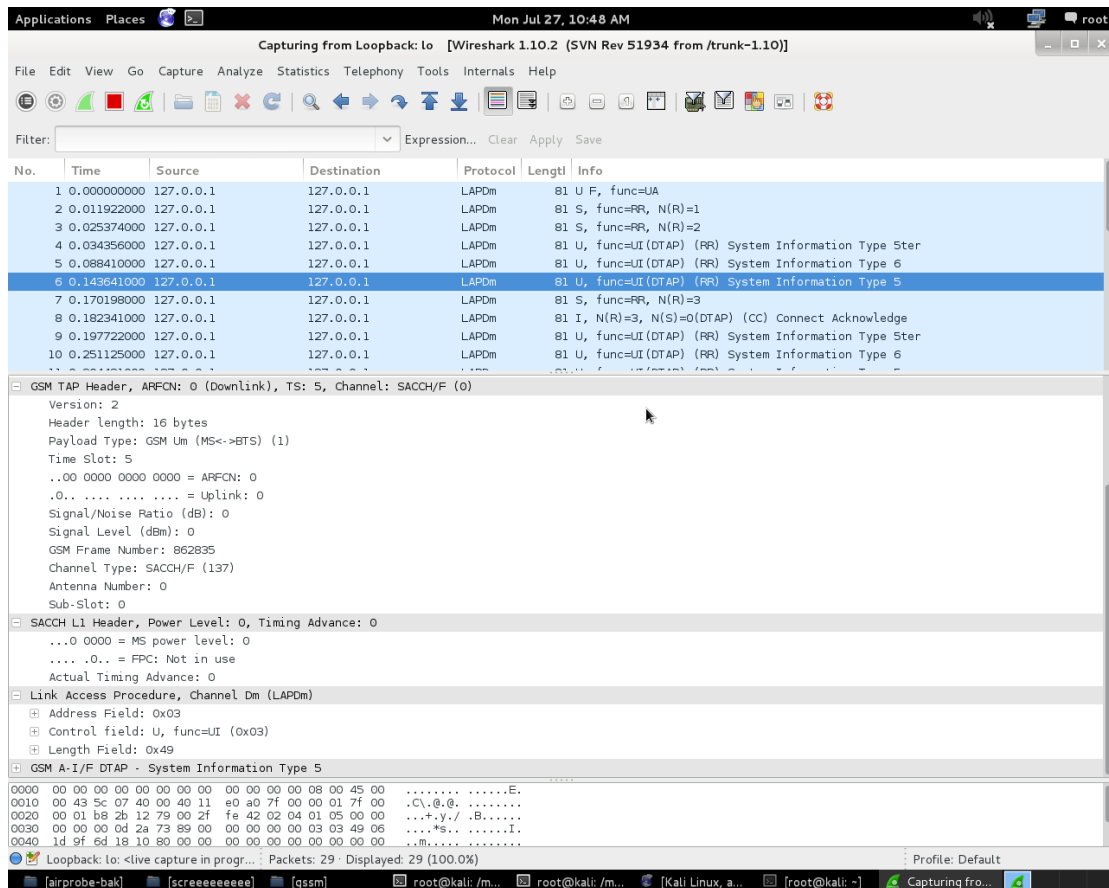


Figure 33 . Decode encrypted part of the assigned “SDCCH/8” Wireshark

Step 2: Now you have a file named speech.au.gsm which contains the audio stream of the channel

Step 3: Convert the speech.au.gsm audio file with toast2:

The toast tool is a software that converts the GSM files that produce Airprobe to audio files that can be played with all of audio players such as VLC. Toast is a project that was developed at the Technical University of Berlin. The research group was needed a speech compression algorithm to support its multimedia conferencing experiments. They found what they were looking for in the ETSI specifications of the Global System for Mobile telecommunication (GSM), Europe's currently most popular protocol suite for digital cellular phones. (John Scourias' overview of GSM does a good job introducing the overall architecture; hire him. Another, more recent, overview of the GSM system (with a list of Web links) comes from Javier Gozávez Sempere.)

The low-level speech compression algorithm of the GSM suite is called GSM 06.10 RPE-LTP (Regular-Pulse Excitation Long-Term Predictor). My colleague Dr. Carsten Bormann and I have implemented a GSM 06.10 RPE-LTP coder and decoder in C. Its source code is freely available, and we encourage you to use it, play with it, and invent new real-time media protocols and algorithms. This tool will be used at the final steps of the GSM cracking produce to get the final file to listen the decoded and sniffed voice call.

Page to download it by clicking at the top of the page the link named: “- free sourcecode“:

<http://www.quut.com/gsm/>

There are some explanations about what type of audio files are the GSM files produced by Airprobe and in what type we must convert them to be listened with VLC player.

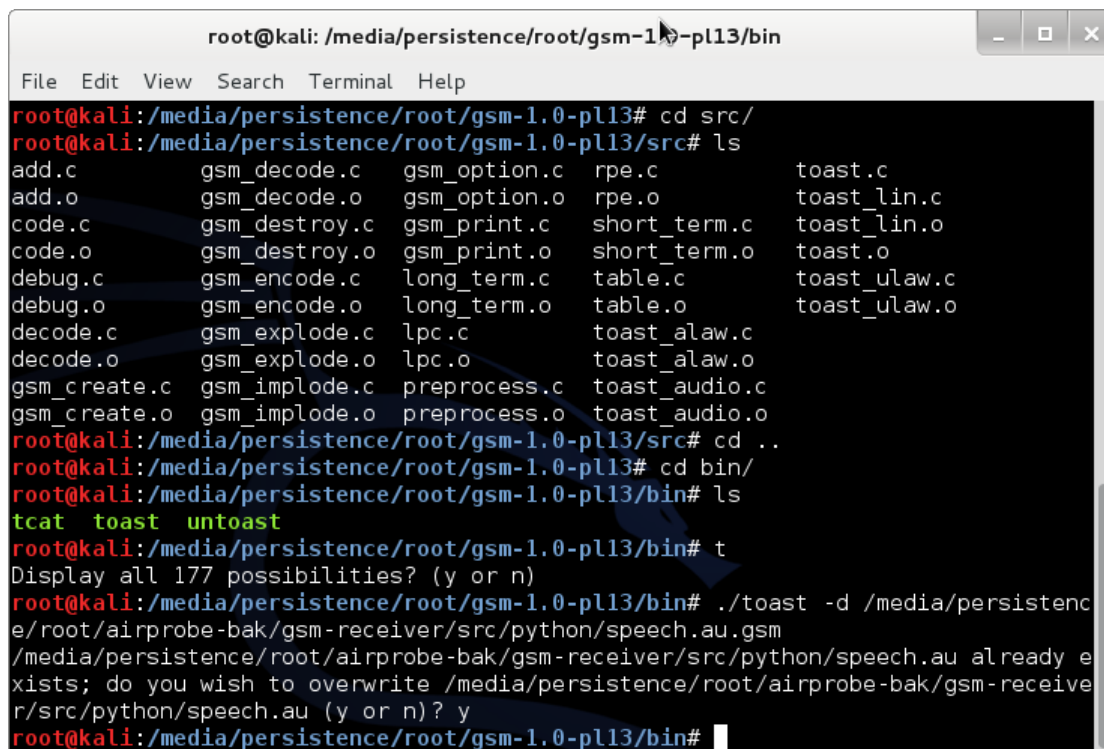
I recommend you to read the information in this page for further explanations. Now we will proceed with the install steps.

Steps to get Toast tool running:

wget <http://www.quut.com/gsm/gsm-1.0.13.tar.gz>

And build and Install like other linux programs

toast -d speech.au.gsm



```
root@kali: /media/persistence/root/gsm-1.0-pl13/bin
File Edit View Search Terminal Help
root@kali:/media/persistence/root/gsm-1.0-pl13# cd src/
root@kali:/media/persistence/root/gsm-1.0-pl13/src# ls
add.c          gsm_decode.c  gsm_option.c  rpe.c          toast.c
add.o          gsm_decode.o  gsm_option.o  rpe.o          toast_lin.c
code.c         gsm_destroy.c gsm_print.c   short_term.c   toast_lin.o
code.o         gsm_destroy.o gsm_print.o   short_term.o   toast.o
debug.c        gsm_encode.c  long_term.c   table.c        toast_ulaw.c
debug.o        gsm_encode.o  long_term.o   table.o        toast_ulaw.o
decode.c       gsm_explode.c lpc.c         toast_alaw.c
decode.o       gsm_explode.o lpc.o         toast_alaw.o
gsm_create.c  gsm_implode.c preprocess.c   toast_audio.c
gsm_create.o  gsm_implode.o preprocess.o   toast_audio.o
root@kali:/media/persistence/root/gsm-1.0-pl13/src# cd ..
root@kali:/media/persistence/root/gsm-1.0-pl13# cd bin/
root@kali:/media/persistence/root/gsm-1.0-pl13/bin# ls
tcat toast untoast
root@kali:/media/persistence/root/gsm-1.0-pl13/bin# t
Display all 177 possibilities? (y or n)
root@kali:/media/persistence/root/gsm-1.0-pl13/bin# ./toast -d /media/persistence/
/media/persistence/root/airprobe-bak/gsm-receiver/src/python/speech.au.gsm
/media/persistence/root/airprobe-bak/gsm-receiver/src/python/speech.au already e
xists; do you wish to overwrite /media/persistence/root/airprobe-bak/gsm-receiv
e/src/python/speech.au (y or n)? y
root@kali:/media/persistence/root/gsm-1.0-pl13/bin#
```

Figure 34 . Toast Tool

Step 4: Finally we open the Vlc player and if every set it up correctly we are able to hear the voice [16][17]

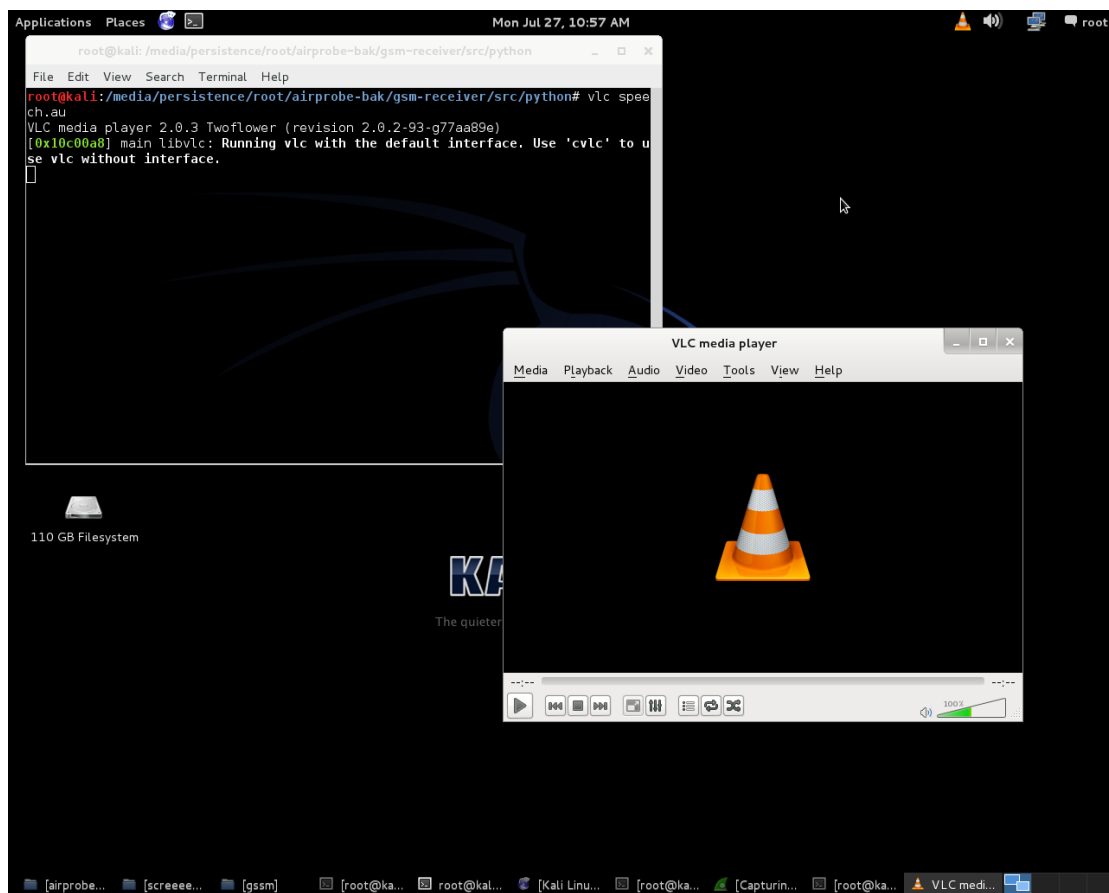


Figure 35 . Vlc Playing the speech.au

Chapter 7: How to Decode Our Gsm Traffic

7.1 Kc key and TMSI number

7.1.1 The TMSI number

Temporary mobile station identities (TMSI) number is allocated within a VLR for enhancing the system security. A TMSI corresponds to an IMSI uniquely within a VLR.

The structure of TMSI can be codetermined by the carrier and the equipment provider.

The principles for TMSI allocation are as follows:

- A TMSI is composed of 4 bytes, which can be 8 hexadecimals.
- The 32 bits of a TMSI cannot be all ones, because an all-one TMSI in a SIM card indicates an invalid TMSI.

A typical example of TMSI is 60340039.

The TMSI number is the identification that the BTS tower give to every MS (Mobile Station), which is your mobile device, to identify it and distinguish it from the other terminals that are connected with the BTS tower.

This TMSI number will be assigned to your device when entering in the area of coverage of the BTS and the BTS can assign this number for one communication between your MS and the BTS only or can reuse this number for your MS if you don't move from the area of coverage of this BTS. I mean if you make a voice call and then make another one, the same TMSI number will be used by the BTS tower for your MS. Sometimes the BTS tower ask the MS for the IMSI number which is unique for each MS. But the BTS tower try to reduce the transfer of this number by the UM Interface or Air Interface (Interface between the BTS and the MS) because of avoiding sniffing the identification number of the MS. Because of this reason is why it uses the TMSI which is a number which is not always the same and don't really identifies the MS at other area coverages or at other connections with the BTS.

7.1.2 Kc key (or cyphering key):

The second is the Kc Key, which is a number extracted with the GSM algorithm and is different for every communication between the BTS and the MS. This number is the output from the GSM.

Official description: The Kc is the 64-bit ciphering key that is used in the A5 encryption algorithm to encipher and decipher the data that is being transmitted on the Um interface.

So this Kc key is fundamental to decode the conversation and we have to obtain it

REMARK: It's important to notice that with all these methods you will only be able to get your own Kc key and not the others Kc keys, which means that you won't be able to crack GSM voice calls or SMS from others. You will only be able to crack your own voice calls and SMS messages [18]

7.2 Active Read Kc And TMSI From SIM

How do you get the Kc from a SIM card

7.2.1 BlackBerry Engineering Screen

This is quite an easy catch: pretty much on all of the BlackBerries you can enable the so called "Engineering Mode" which will simply show you the current Kc. Not much of fun, but a reliable, good way to do it.

TESTED:NO (as shown by Karsten Nohl for example at BlackHat 2010)
WORKS: YES

Now, this one is quite a tricky one, because setting up OsmocomBB already requires quite an amount of work, but once you have it up and running AND you are lucky with the cables and the code (which is not usually the case) you can simply run the mobile app and then use the telnet interface to get the Kc:

1. Upload layer1 to your phone
2. Run mobile -i 127.0.0.1
3. telnet 127.0.0.1 4247

After that simply say:

```
show subscriber 1
```

At the top you should see the Kc printed.

TESTED: No

WORKS: PARTIALLY

7.2.3 Using a SIM-card reader/Smart Card Reader

Some people said on the A51 mailing list that by using a simple SIM card reader they were able to extract the last used Kc from the card. I am not sure about this, but it sounds reasonable and the people who wrote about were quite convincing.

We tried this method using 2 cheap Sim-Readers from Ebay for 10 Euros and it didn't work for me. I was running SIMspyII and other forensic investigations tools and didn't recognize both of devices (We already turned off PIN-code verification for the card, not sure however how having a PIN code would change the procedure, but I assume the SIMspyII program has support for PIN-codes).

I tried to find devices that can work with SIMspyII and I find that model Omnikey CardMan 5321 can work with SIMspyII .



Figure 36 . OMNIKEY® 5321 SIM-card reader

TESTED: NO
WORKS: THE GOOGLE SEARCH SAY YES

7.3.4 AT+CSIM Command

This one is the eldest and most well-known command: some phones allow you to use one of the standard-defined-but-not-always-implemented AT command AT+CSIM which let's you to send raw APDUs (= "commands") to the SIM-card via the modem. The amount of phones supporting this is very limited, according to some people older Siemens and Alcatel phones let you do this. Also older iPhone's (3GS/3G/2G) let you do this if you are jailbroken (you need to install minicom from Cydia then connect to the device /dev/tty.debug). Newer iPhone's don't really let you do this, iPhone 5 owners – we all are out of luck.

The command you would like to send is something like this:

A Sample run:

```
AT+CSIM=14,"A0A40000026F20"  
+CSIM: 34,"000000096F2004001100BB010200009000"
```

OK

```
AT+CSIM=10,"A0B0000009"  
+CSIM: 22,"E0940FC09AEFA000009000"
```

OK

Again, you find the last Kc used here: E0 94 0F C0 9A EF A0 00 and also the key sequence number: 00 [19]

Problem With Samsung GT-5839i

The approaches for extracting Kc and TMSI I found use AT+CSIM command to issue raw APDUs. We can't send APDUs to the SIM, because AT-commands like +CSIM are not supported in Android. It is necessary to modify the RIL to make them work, which is what Seek for Android does. SIM IO commands in android are issued using +CRSM, look into reference-ril.c code.

So we looked into +CRSM command to see how it works and what it can do. We use this command already to obtain the ciphering indicator. We managed to read TMSI and LAI

Demo:

- * 18055A1B TMSI
- * 05F5101030 LAI: 50501 4144
- * FF current T3212 value (used on phase 1 devices only)
- * 00 location update status

176 is for READ Binary, 28542 is decimal representation of EF fileid 0x6F7E. Other parameters should specify the record number and length of response.

This should read Kc from file 0x6F20,

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

Read Kc and TMSI number from my device GT-5839i.

- Process to get these 2 numbers or keys:

First of all I remark that the 2 numbers (TMSI and Kc) are obtained by the same procedure. This procedure is named AT commands.

AT commands are used to communicate via terminal or user interface to a Modem and get information about this Modem. In this case the Modem will be the Mobile phone with the android system and the part of the device we want to get access is the SIM card, which are stored these 2 numbers and where we have to enter after every voice call or SMS to get these numbers of the last communication between the BTS and the MS.

For this reason is why we have only been able to get these numbers with a Samsung android device. The Stock ROMs of the Samsung devices brings the possibility to communicate through the android device like a Modem. I mean when you connect the Mobile to the PC via USB, this device will be recognized by the system like a memory storage (DCIM), like and ADB interface (if you have installed the ADB software of android) and finally like a Modem that brings you the possibility to communicate through AT commands like any other Modem to the SIM card of the Mobile phone and extract the 2 necessary numbers.

It's important to remark that I'm using a Samsung Galaxy s GT-5938i model and that we suppose by some comments on different Forums and some google search that all the Samsung devices have this special option of Modem

ADB (Android Debug Bridge) description: adb are two different applications — one running on your computer (Windows, Linux or Mac) and one running on your phone. When your phone is connected, and USB debugging is enabled, you can issue commands and communicate with the phone using your computer screen and keyboard.

So ADB is the quickest and most useful way to control your Android device through your PC by terminal prompt and make it more simple to enter commands that can be with your device.

Once you have written all of this information the only thing you must know and check about your Samsung android device is that if it has the Stock ROM. We mean that the ROM was not changed by a Custom ROM like CyanogenMOD or others. Because the Stock ROM allows us to access the SIM card like a Modem.

To check out if it's possible to be recognized like a Modem you have to follow the next steps.

First you have to enable some settings of the phone before connecting it via USB to the computer:

Settings → Applications → Unknown sources (enable this option)

Settings → Applications → Development → USB Debugging (enable this option)

Then you are prepared to catch the USB cable that comes with the device and connect it to your computer.

It will appear on your mobile a tab that shows that the device is connected through USB.

In the computer to check if it's recognized like a Modem, you only have to do a pair of things.

- Minicom Install (Linux operating system)

This software hasn't any graphical interface but is really the same than Hyperterminal in Windows but you have to use it through the terminal prompt.

- Create profile to Minicom 2.7 and test if we can communicate with Sim card

```
giannios1983@giannios1983-virtual-machine ~ $ sudo minicom giannis2
[sudo] password for giannios1983:

Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:22.
Port /dev/ttyACM0, 14:18:56

Press CTRL-A Z for help on special keys

OK
AT
OK
GT-S5830i
OK
```

Figure 37 . Test Minicom With Samsung Device

```
Terminal
Welcome to minicom 2.7

OPTIONS: I18n
Compiled on Jan  1 2014, 17:13:22.
Port /dev/ttyACM0, 18:14:16

Press CTRL-A Z for help on special keys

OK
+CRSM: 144,0,"9F9A8A0 [REDACTED] B0000000000000" ← TMSI
OK
+CRSM: 144,0,"84D3 [REDACTED] 04" ← Kc
OK
█
```

Figure 38 . Read TMSI and Kc

AT+CRSM=176,28542,0,0,1

*First 8 digit is TMSI

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

* Here we can identify the KC key hexadecimal number: 84D3***** and the sequence number: 04 [20]

7.2 Receive Live Channel

To decode a live channel using RTL-SDR type in terminal

```
./gsm_receive_rtl.py -s 1e6
```

A window will be showed . We have to tune a known GSM channel that we found earlier using SDRSharp by entering the known frequency. After this step , just click in the middle of the GSM channel in the showing window. Within a few seconds some GSM data should begin to show consecutively in wireshark. Type the command `./gsm_receive_rtl.py -h` for information. Use `-s` flag there to set up the sample rate to 1.0 MSPS, and might work much better than the default value of 1.8 MSPS as we see that there should be only one GSM high level in the wideband spectrum.

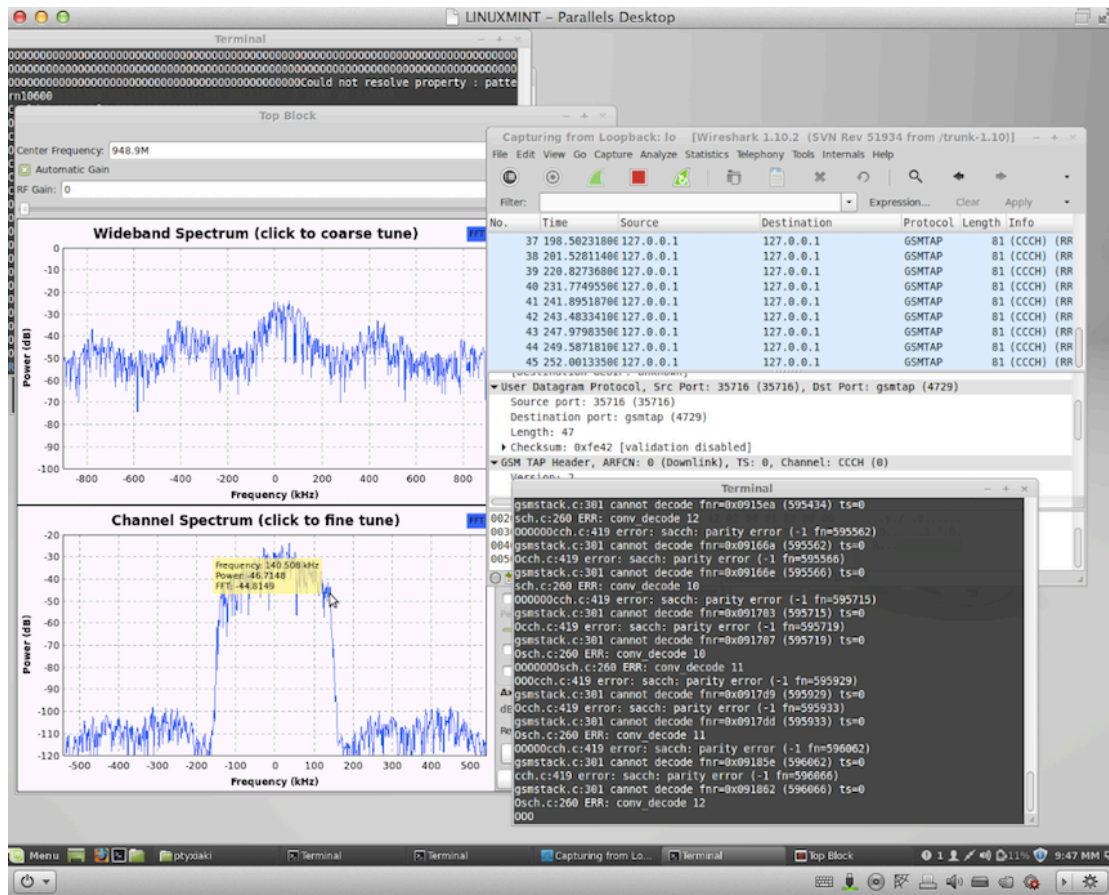


Figure 39 . Sniffing Packets With Rtl-sdr

If we have trouble getting data, but receive many errors like

```
sch.c:260 ERR: conv_decode 12
```

then you should calibrate your RTL-SDR stick using the kalibrate-rtl program
Kalibrate-rtl.

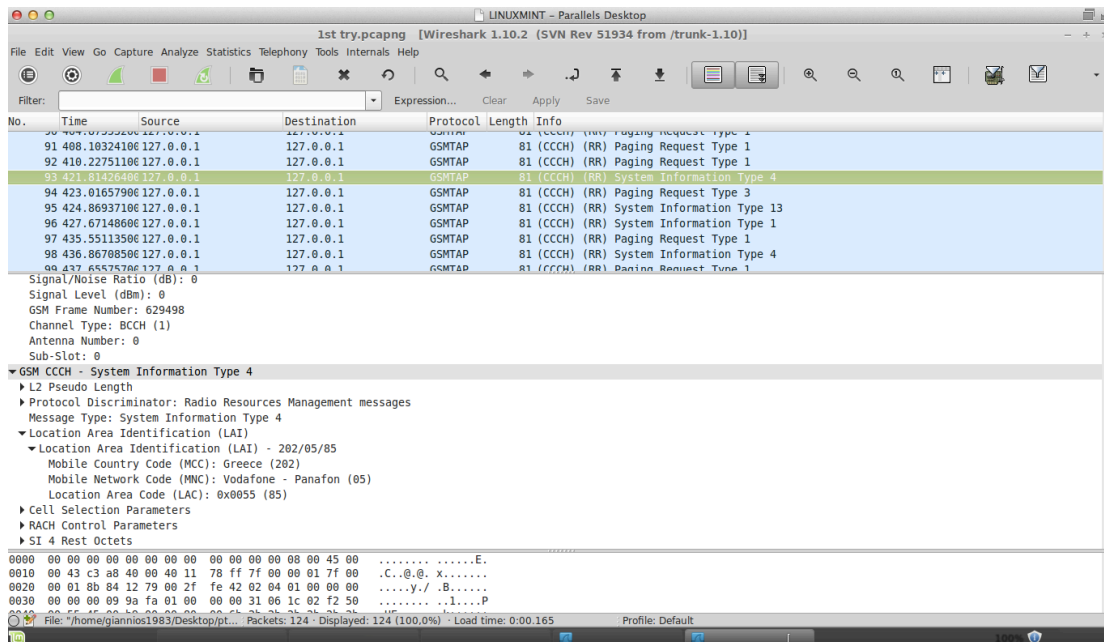


Figure 40 . Captured System Information Type 4 –General Information

This first type of messages contains information of the **BTS towers** (of the system).

A) System Information Types:

Type 1: Channel type = BCCH (But GSM CCCH Info)

LIST of ARFCNs of the cell!!!

RACH control parameters.

Type 2: Channel type = BCCH (But GSM CCCH Info)

Neighbour cell description like LIST of ARFCN's of the cell.

Neighbour cell description – BCCH frequency list.

Type 3: Channel type = BCCH (But GSM CCCH Info)

Cell identity code decoded, LAI(MCC+MNC+LAC)decoded and some GPRS information.

Type 4: Channel type = BCCH (But GSM CCCH Info)

LAI(MCC+MNC+LAC) decoded, Cell selection parameters and RACH control parameters. Some GPRS information too.

Type 2ter: Channel type = BCCH (But GSM CCCH Info)

Neighbour cell description like LIST of ARFCN's of the cell.

Neighbour cell description – Extended BCCH frequency list.

Type 2quater: Channel type = BCCH (But GSM CCCH Info)

3G message with information that we don't take into account in this study. Like 3G neighbour cell description.

Type 13: Channel type = BCCH (But GSM CCCH Info)

They contain all the important information about GPRS like GPRS Cell options, GPRS Power Control Parameters.

This second type contains information of the **Mobile Station (MS)**.

B) Paging Request Types:

Type 1: Channel type = CCCH (And GSM CCCH Info)

Mobile Identity 1 number (IMSI)

Page Mode = normal paging (P1)

Channel Needed

Mobile Identity 1 and 2 = TMSI/P-TMSI

Page Mode = normal paging (0)

Channel Needed

And other types of Paging Request Type 1 can carry any combination of MS identifiers such as TMSI / P-TMSI IMSI of the MS1 and MS2 or IMSI the MS1 and MS2, etc.

Only the IMSI MS1 or No identity code, etc. Type 2: Channel type = CCCH (And GSM CCCH Info)

Mobile Identity 1 and 2 = TMSI/P-TMSI i IMSI del Mobile Idnetity 3

Page Mode = normal paging (0)

Channel Needed

Type 3: Channel type = CCCH (And GSM CCCH Info)

Mobile Identity 1, 2 , 3 and 4 = TMSI/P-TMSI (Not decoded)

Page Mode = normal paging (0)

Channel Needed

C) Immediate Assignment: Channel type = CCCH (And GSM CCCH Info)

Time Advance Value

Packet Channel Description (Time Slot)

Page Mode = Extended Paging(1)

7.3 Capturing a cfile with the RTL-SDR In Our Kali Linux VM

7.3.1 First Way

Use the **rtl_sdr tool** to sniff all the information of the channel you have found. The rtl_sdr tool comes with gnuradio and it will catch all the information with the RTL dongle and save it in a **.bin file** at the directory that you want to save it.

Example of use:

```
./rtl_sdr /tmp/rtl_sdr_capture.bin -s 1.0e6 -f 957e6 -g 44.5
```

So in this case, the tool will save the data into a file

Named: **rtl_sdr_capture.bin** and placed inside the tmp directory. The -s flag is to specify the sample rate which seems to be works better than the 1.8e6 which is the default value. The other settings are the frequency which we want to tune and the gain. I prefer to decode a live channel that use top_block tool that comes with airprobe to get the information because this tool seems to be not as efficient as the rtl_sdr that have a **bandwidth of 3.5MHz** which is enough to include all the GSM channel with the frequencies the system can change during the call. As we know the GSM is a frequency hopping system that will change the transmitter and receiver frequency in a call following the pattern of ARFCNs received in the **System Information Type 1** frames before setting the call.

Convert the .bin file into a .cfile file with a precompiled gnuradio-companion scheme.

STEP 1: Convert the **.bin** file into a **.cfile** file with a precompiled **gnuradio-companion** scheme.

The gnuradio-companion is a tool that comes with gnuradio and it has a graphical interface. You must download the next

file: <http://sdr.osmocom.org/trac/attachment/wiki/rtl-sdr/rtl2832-cfile.grc>

After this, open gnuradio-companion by typing it in a terminal prompt:

```
Sudo gnuradio-companion
```

And the program will open, the go to the tab open file and search the named: **rtl2832-cfile.grc**.

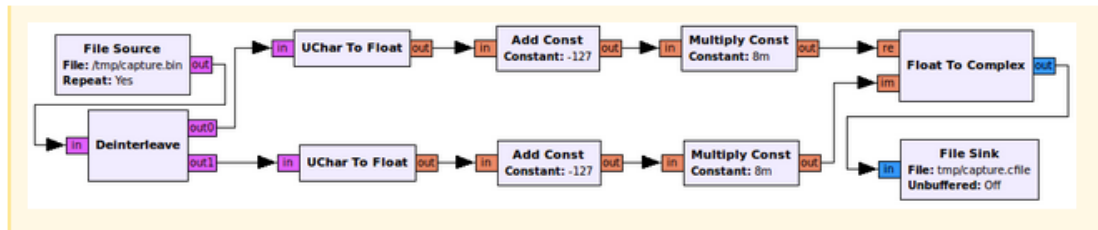


Figure 41 . Bin to Cfile .grc File

Set the file source to the capture.bin file directory where you have saved it with in the previous STEP 3, and set the file output for a file called capture.cfile which should be located in the **'airprobe/gsm-receiver/src/python'** folder. Also, make sure that **'Repeat' in the File Source block is set to 'No'**.

Now execute the GRC flow graph by clicking on the icon that looks like **grey cogs**. This will create the capture.cfile. The flow chart will not stop by itself when it's done, so once the file has been written press the **red X icon** in GRC to stop the flow chart running. Because it won't stop by itself. Then you can close the gnuradio-companion and delete the .bin file saved in the tmp directory

7.3.1 Seceond Way

We goanna to use a Gnu-radio Schema Download it from

http://wiki.hackbbs.org/index.php/Gsm_receive_rtl.py

And we have to open it with the Gnu-radio as the same way as before .

Sudo gnuradio-companion name_of_file.grc

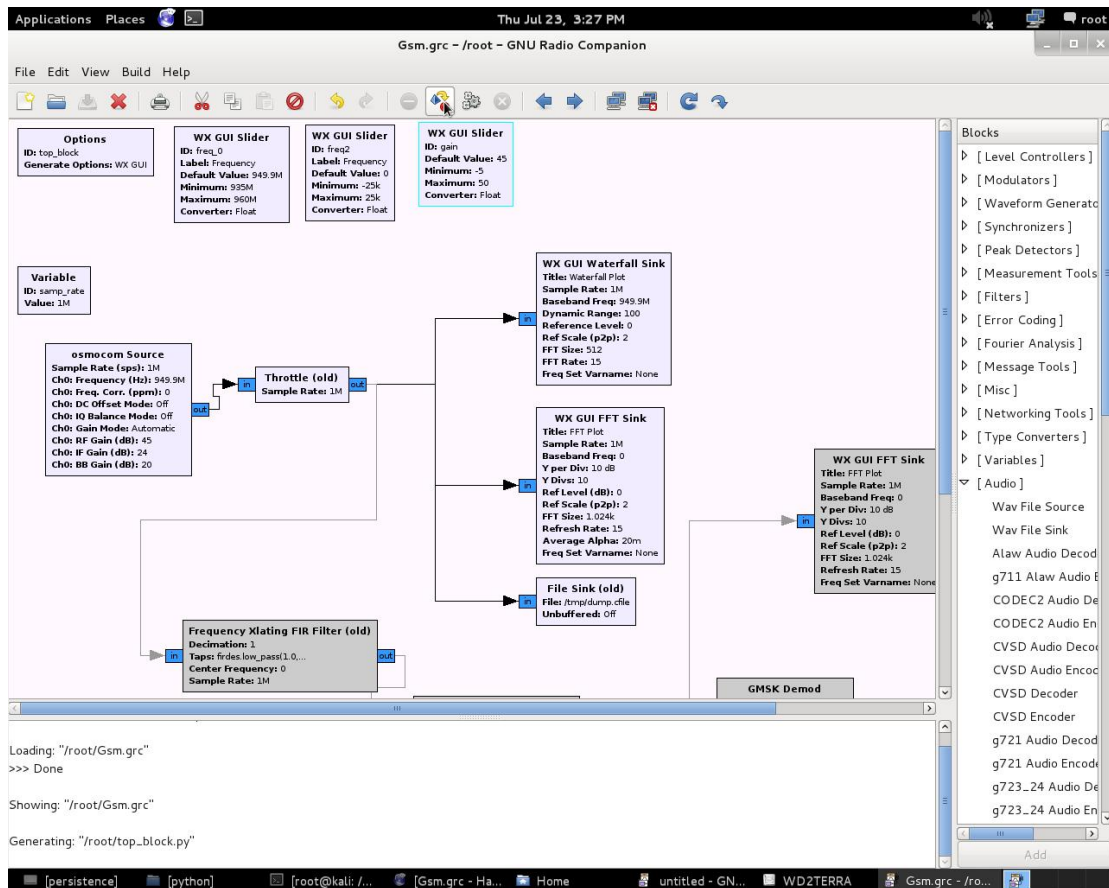


Figure 42 . Gsm.grc

As we see in Figure 39. we can sniff any frequency that we want and generate a .cfile which contains capture traffic on with .cfile gnuradio-companion. We can change the frequency on pop-up menu.

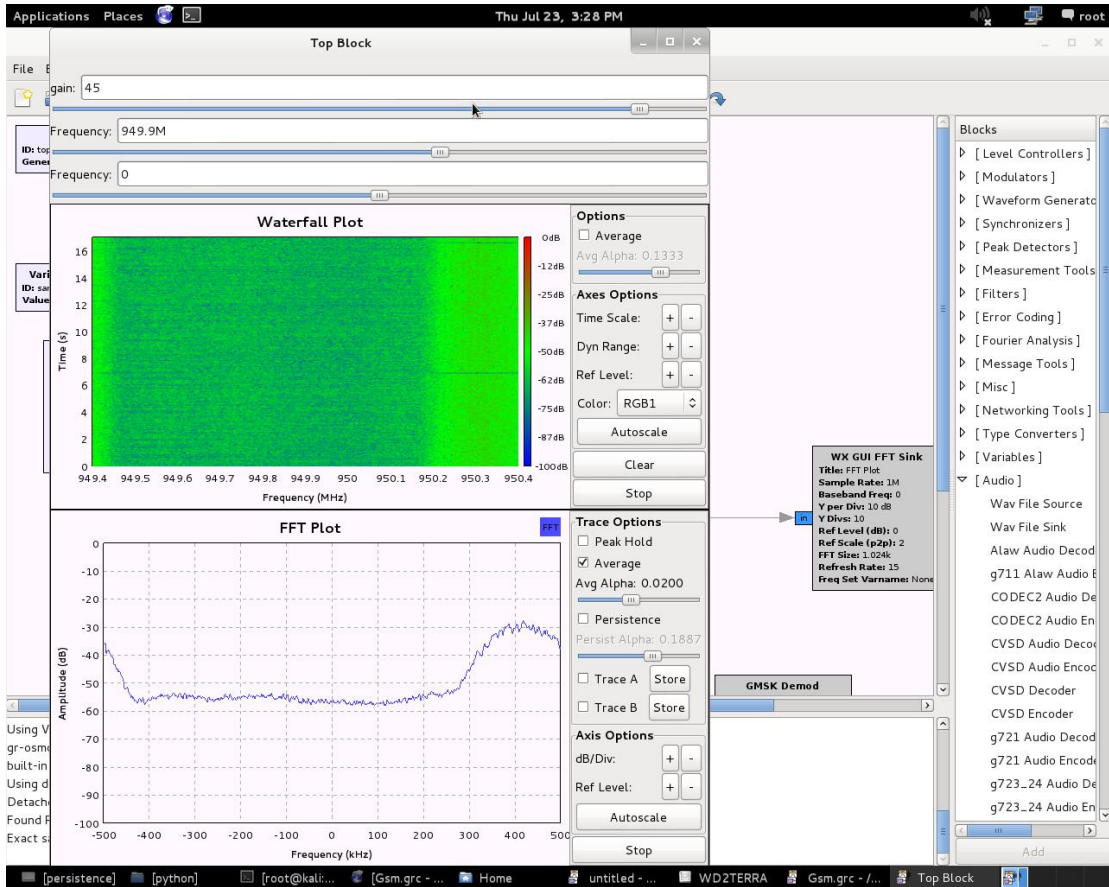


Figure 43 . Gsm.grc Generated with Gnu-radio

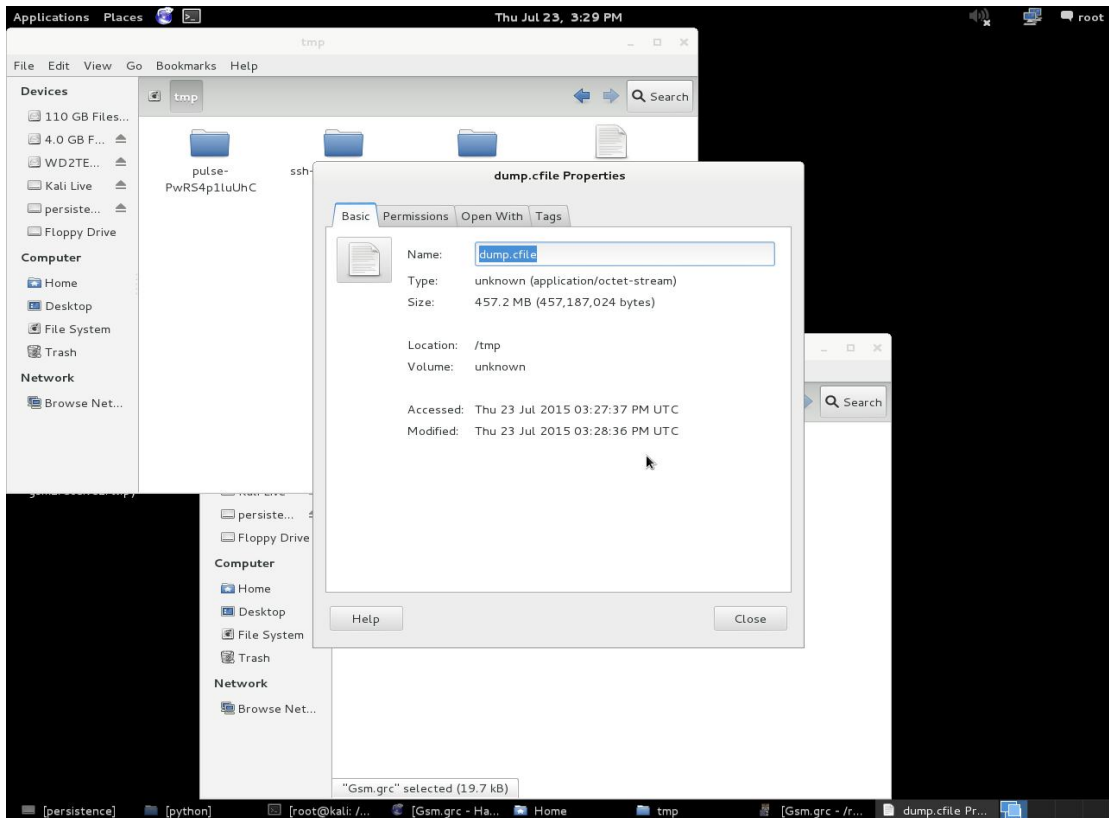


Figure 44 . Dumb.cfile

Open and setup **wireshark**:

Open another terminal prompt and to open wireshark with sudo privileges:

```
Sudo wireshark
```

Note that it's always better to work with sudo privileges when using airprobe, wireshark, gnu-radio, etc.

Then the wireshark software will open and you must select **lo(loopback)** and **start**.

Note that wireshark comes installed in kali linux.

When a window which is capturing is opened go to the **filter box** and write **gsmtap** to see only the GSM frames.

7.4 Use airprobe to send the decoded information to wireshark and analyze the frames.

First, we have to decode the signalling frames to know more things about the calls we are decoding:

```
./go.sh capture.cfile 64 0B
```

The 64 is the decimation rate of the RTL-SDR, 0B is the configuration which go.sh is going to use: 0 means Timeslot 0 (beacon channel), B is the configuration that the cell uses on the beacon channel. With this we will get information about the system and of the calls identification numbers TMSI or IMSI (in few cases). The information that contains any frame of signalling will be explained in a new thread in the next days.

Decoding an SMS:

*Here are **all the available configurations** that are supported by airprobe:*

```

0C : TimeSlot0 "Combined configuration", with SDCCH/4
      (FCCH + SCH + BCCH + CCCH + SDCCH/4)
0B : TS0 "FCCH + SCH + BCCH + CCCH"
1S : TS1 SDCCH/8
2T : TS2 (Full Rate) Traffic
1TE: TS1 Enhanced Full Rate Traffic

```

Figure 45 . Available configurations Supported By Airprobe

As you can see these are the GSM configurations used in the different frames and you will have to be able to recognize each one to identify if we have to use airprobe configuration or another to decode the data of your call or SMS.

We will assume that we sent a text message to ourself while capturing data.

So now we can see all the messages of the beacon channel, but what are you looking for in the Wireshark log? It is quite simple: **first a “Paging Request” for the TMSI of the target phone**, then a **“Cipherring Mode Command”**. These are the messages which point out that a transaction indeed happened.

Now to continue with the flow it is best way trying to decode the specific cfile but now giving the key to go.sh:

We will assume that we sent a text message to ourself while capturing data.

*So now we can see all the messages of the beacon channel. It is a simple thing: **first a all “Paging Request” for the TMSI of the target phone**, then a **“Cipherring Mode Command”**. These are the messages which indicate that a transaction actually been happened.*

Now we will go on with the flow it is better way to trying to decode our cfile but now giving the solution to go.sh:

`./go.sh capture.cfile 64 0C KEY //Kc can get it Actively`

The kc Key shown how was taken actively on previous chapter of my master thesis.

What we expect to see now? Well, it depends on the network: either there is an **“Immediate Assignment”** telling to the mobile phone to go to different timeslot or you will actually be able to see the text message.

Instead of the SMS you find an **“Immediate Assignment”** message you need to open it and see which timeslot the phone is being order to and then you need to decode that timeslot using `go.sh`. So, for example if it says that the phone needs to go to Timeslot 3 then your command would be:

```
./go.sh capture.cfile 64 3S KEY
```

Notice: that we did not only change the Timeslot number from 0 to 3 , but also the configuration from C to S, because the target phone is now on a Standalone Dedicated Control Channel (SDCCH), not on the beacon channel so we need to decode it other way.

It is also worth noting that SMS messages are almost always sent on the Center Channel not on the Traffic Channel.

Here we can see a flowchart of the whole process to make it easier to understand (naturally since we can only see the downlink this shows only what happens on the downlink):

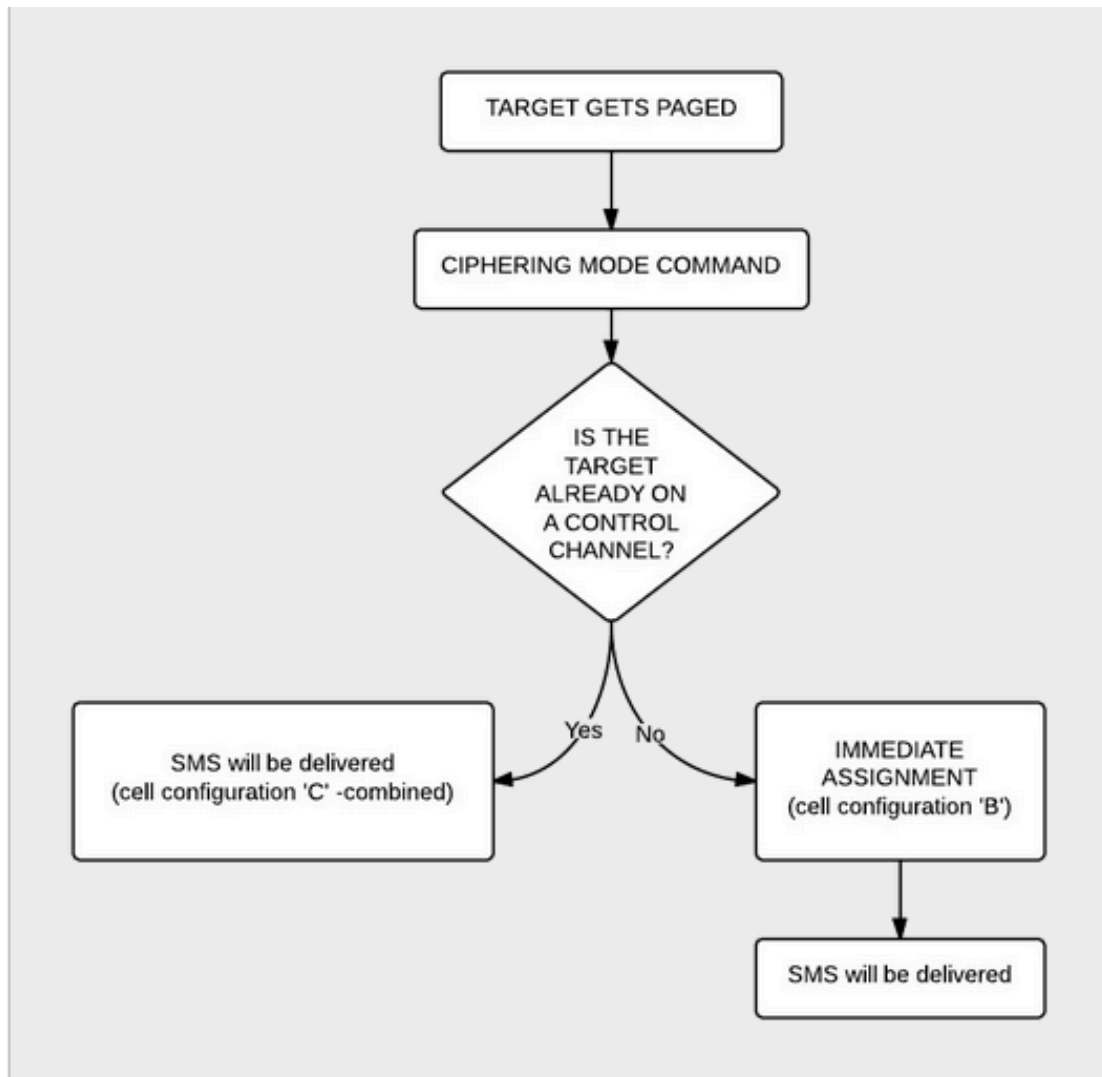


Figure 46 . Flowchart of the whole process Decoding Sms (source <https://ferrancasanovas.wordpress.com/cracking-and-sniffing-gsm-with-rtl-sdr-concept/>)

Decoding a Voice call:

Now we were able to decode an SMS let's get to something a little bit harder: decoding a voice call!

Well the first step is the same as it was when we decoded a text message: we look at the beacon channel, Timeslot 0:

```
./go.sh capture.cfile 64 0C
```

What do we expect to see? Nothing besides the **“Cipher Mode Command”** because we didn't provide the key, so let's do that:

```
./go.sh capture.cfile 64 0C KEY
```

Logically there needs to be an **“Immediate Assignment”** command, because the phone NEEDS to change at least once to a different timeslot to receive voice data (to a Traffic Channel, Timeslot 1-7). What we saw when decoding the SMS is correct here too: depending on the network configuration we can see some messages about the call setup (if it is an incoming call we can even see the caller ID – the phone number calling our target) then an “Immediate Assignment” (configuration ‘C’ – combined) or we can only see an “Immediate Assignment” directing the phone to a Control Channel (just like it happened when receiving an SMS, configuration ‘B’).

Of course if you follow the phone to the Control Channel you will see the call setup messages (in case of an incoming call) then another **“Immediate Assignment”** command, this time directing the phone to a Traffic Channel.

Here is again a flow chart showing the process:

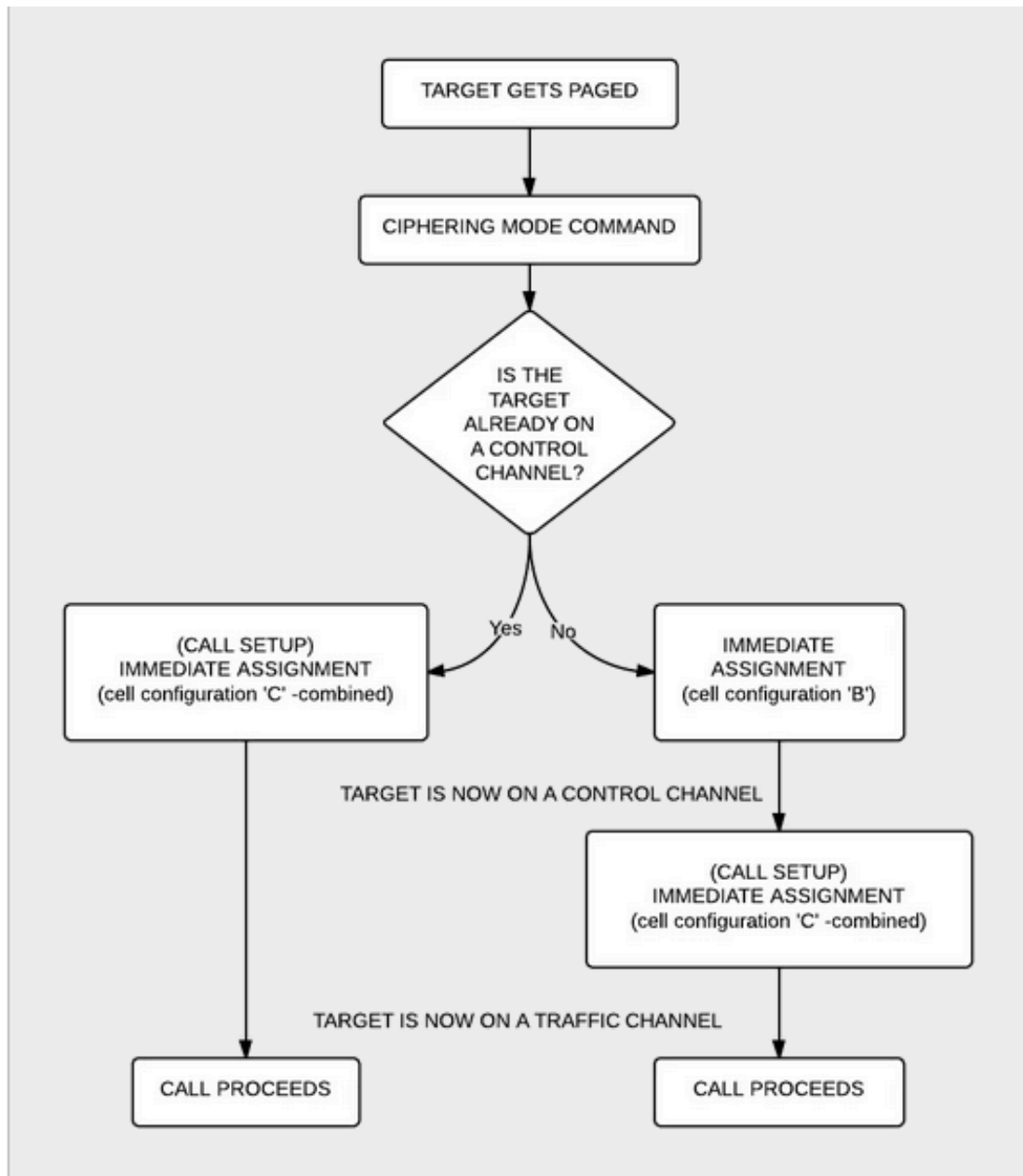


Figure 47 . Flowchart of the whole process Decoding Voice Call (source <https://ferrancasanovas.wordpress.com/cracking-and-sniffing-gsm-with-rtl-sdr-concept/>)

Now there is only one question left: how do we decode the traffic channel to actually get the voice data? Again, it is something that depends on the network: if the network uses simply Full Rate Speech then you can do the same what has been written in Srlabs's tutorial:

`./go.sh capture.cfile 64 1T KEY`

What does this command do? It decodes Timeslot 1 as a Traffic Channel. We know what timeslot to decode from the "Immediate Assignment" command message, T

means Full Rate Speech. **The command results in a file called “speech.au.gsm”**, which needs to be converted to .au file using ‘toast’:

```
toast -d speech.au.gsm
```

The toast tool is a software to convert files and I will explain the setup process of this software on another thread.

The resulting .au file could be played back using any player, e.g. cvlc (Command Line VLC):

```
cvlc speech.au
```

So we must download and install VLC player too.

If we can not hear anything but beeps and other weird noises then there is a pretty good chance that the cell is using Enhanced Full Rate Speech instead of simple Full Rate Speech.

To decode the channel as an Enhanced Full Rate Speech Traffic Channel:

```
./go.sh capture.cfile 64 ITE KEY
```

This results in a file called “speech.amr” which could be played back without any more modifications using for example Commandline VLC:

```
cvlc speech.amr
```

[21] [22]

Chapter 8. Projects & Tools For Analyzing Intercepting Gsm Signals

In this chapter of our Master Thesis we would want to approach some tools that have been made in amateur level and are used for the analysis and intercepting the Gsm Signals . Most of them have many errors at the source code , we should to correct the code and then to execute.

8.1 The gr-gsm project

All the above installation are implemented on VmWare Linux mint machine 17 machine

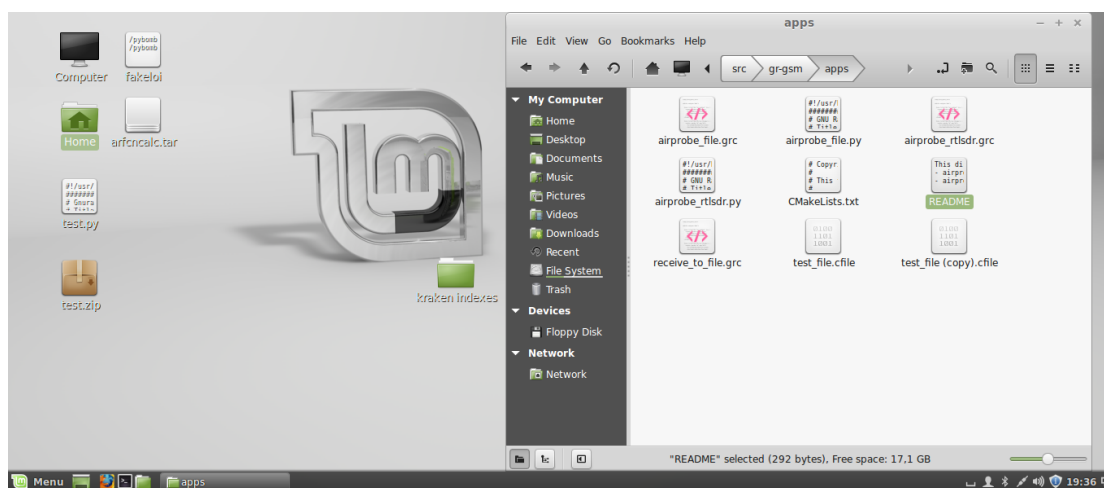


Figure 48 . Linux Mint 17 VMachine & Gr-gsm project application files

The gr-gsm project is based on the gsm-receiver written by Piotr Krysik (also the main author of gr-gsm) for the Airprobe project.

The aim is to provide set of tools for receiving information transmitted by GSM equipment/devices.

Installation

The project is based on GNU Radio signal processing framework and takes advantage of its great features like stream tagging and message passing. Presence of GNU Radio is therefore a basic requirement for compilation and installation of gr-gsm.

The easiest way to install gr-gsm is to use pybombs installer (GNU Radio install management system). Installation with this tool was tested on Ubuntu 14.04 and 14.10 (on 14.10 installation will be much faster as compilation of GNU Radio is not necessary) and also installed on Linux Mint 17 . For installation of pybombs you will need git. On Debian based distributions you can get it with:

sudo apt-get install git

Then download pybombs sources using git:

```
git clone https://github.com/pybombs/pybombs.git
```

Go into pybombs directory and configure it:

```
cd pybombs
```

```
./pybombs config
```

As a install prefix enter /usr/local/. The rest of the options can be left as default.

On distributions that have GNU Radio version 3.7.7 and above in standard repository, GNU Radio can be installed from packages without compilation. To avoid this quite lengthy process use:

```
./pybombs config forcebuild ''
```

To check GNU Radio version use:

```
apt-cache policy gnuradio-dev
```

The key of success is to test the Gnu-Radio if installed correct.

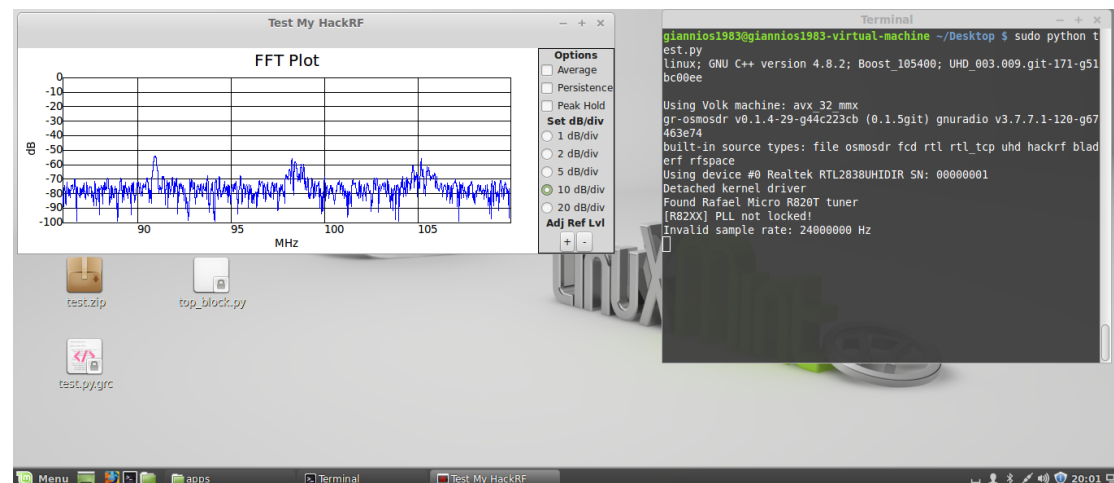


Figure 49 . Test of Gnu-Radio

Then build and install gr-gsm with following command:

```
sudo ./pybombs install gr-gsm
```

Pybombs will take care of downloading all of required libraries and for installation of GNU Radio and building gr-gsm.

At the end create the ~/.gnuradio/config.conf config file so gnuradio-companion can find custom blocks of gr-gsm:

```
[grc]
```

```
local_blocks_path=/usr/local/share/gnuradio/grc/blocks
```

Usage

There are many possible applications of gr-gsm. At this moment there is one application that is ready out of the box. It is improved replacement of the old

Airprobe - the program that lets you receive and decode GSM control messages from timeslot 0 on the broadcasting channel of a BTS. After installation of gr-gsm there are two python executables that will be installed:

- airprobe_rtlsdr.py,
- airprobe_file.py.

Airprobe with RTL-SDR input

This program uses cheap RTL-SDR receivers as a source of the signal. It can be started by running from a terminal:

airprobe_rtlsdr.py

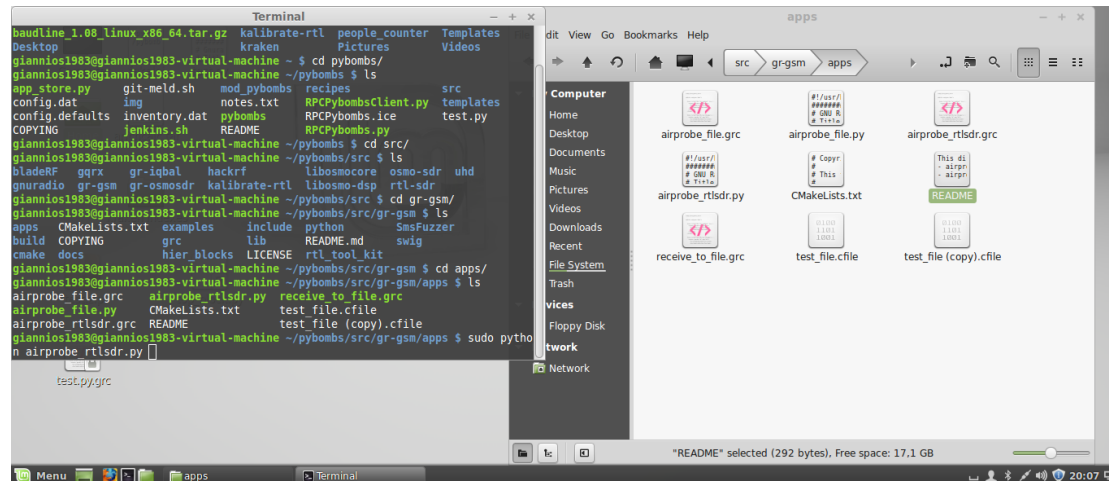


Figure 50 . Commands to run Airprobe_rtlsdr.py

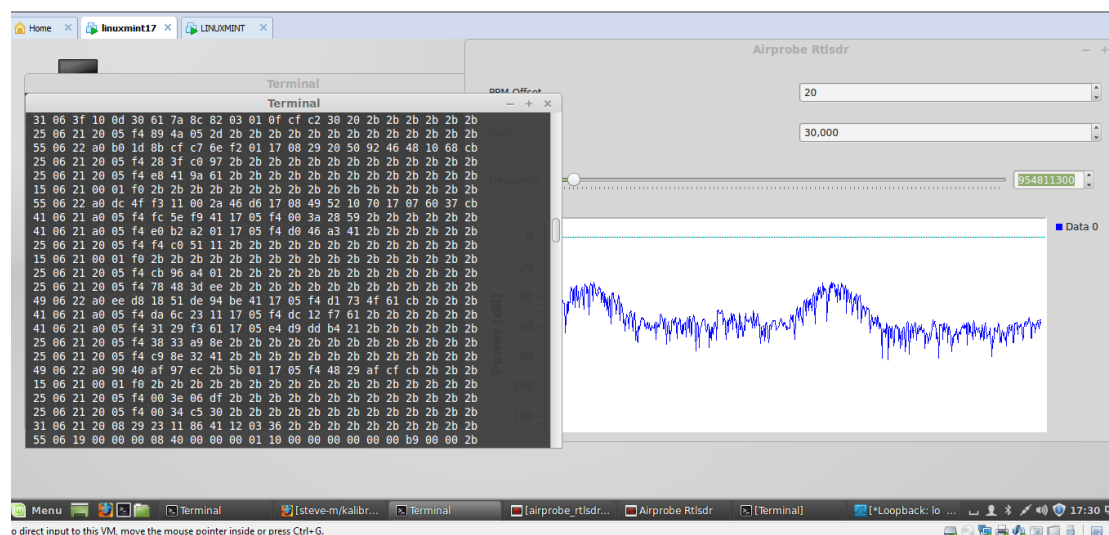


Figure 51 . Airprobe Rtlsdr While running and Capturing GSM packets

We'll see a spectral display tuned to the frequency that you set in the Python class constructor previously. Use the slider buttons to begin tuning through the spectrum to search for a GSM broadcast control channel (BCCH). On the signal display, it'll show up as a relatively wide hill prominent among the rest of the noise.

When you've tuned to the correct center frequency of the BCCH, you'll see the downlink data begin streaming in the same terminal window where you invoked airprobe. The giveaway will be a string of scrolling hexadecimal "2b" octets. These are padding octets among the actual system messages passed on the channel

The window of the program contains amplitude spectrum of the signal drawn in real-time. The central frequency of the signal can be changed by moving fc slider. The GSM signal has bandwidth of around 200kHz. By looking for constant hills on the spectrum of such width you can find a GSM broadcasting channel. After setting the fc slider to a carrier frequency of a broadcasting channel the program should immediately print content of subsequent messages on the standard output.

If it doesn't happen, set ppm slider into different positions. The slider is responsible for setting devices clock offset correction. If the clock offset is too large the clock offset correction algorithm that is implemented in the program won't work. There is intentionally added upper of allowable clock offset - it was done in order to avoid adaptation of the algorithm to neighbour channels that would inevitably lead to instability. You can use the value set later by passing it as argument of the program:

```
airprobe_rtlsdr.py -p <correction>ls
```

Airprobe with file input

This program processes files containing complex data - interleaved float IQ samples. Example of the usage:

```
airprobe_file.py --samp-rate=1M --fc=940M -i input_file
```

where:

--samp-rate - sampling frequency of the data stored in the file,

--fc - central frequency of the recorded data - it is needed for frequency offset correction,

-i - the file containing the complex data.

Analyzing GSM messages in the Wireshark

The Airprobe (file, rtlsdr) application sends GSM messages in GSMTAP format that was created by Harald Welte to the UDP port number 4729. Wireshark interprets packets coming on this port as GSM data with GSMTAP header and it is able to dissect messages.

On Debian like systems *Wireshark can be installed with:

```
sudo apt-get install wireshark
```

To start Wireshark straight to analysis of the GSMTAP packets obtained from gr-gsm's airprobe use following command:

```
sudo wireshark -k -Y '!icmp && gsmtap' -i lo
```

The options are broken down as follows:

-k: tells Wireshark to begin capture immediately

-Y: sets a display filter

'licmp && gsmtap': only displays GSMTAP formatted messages. GSMTAP is a pseudo-header that's not part of any standard protocol and is used to transport GSM messages over the air inside UDP or IP packets. Radiotap headers perform the same function in 802.11 traffic.

-i lo: sets the capture interface as the loopback, through which Airprobe is transmitting the demodulated GSM signal.

Once Wireshark comes up, you'll see the GSMTAP packets scrolling:
[23]

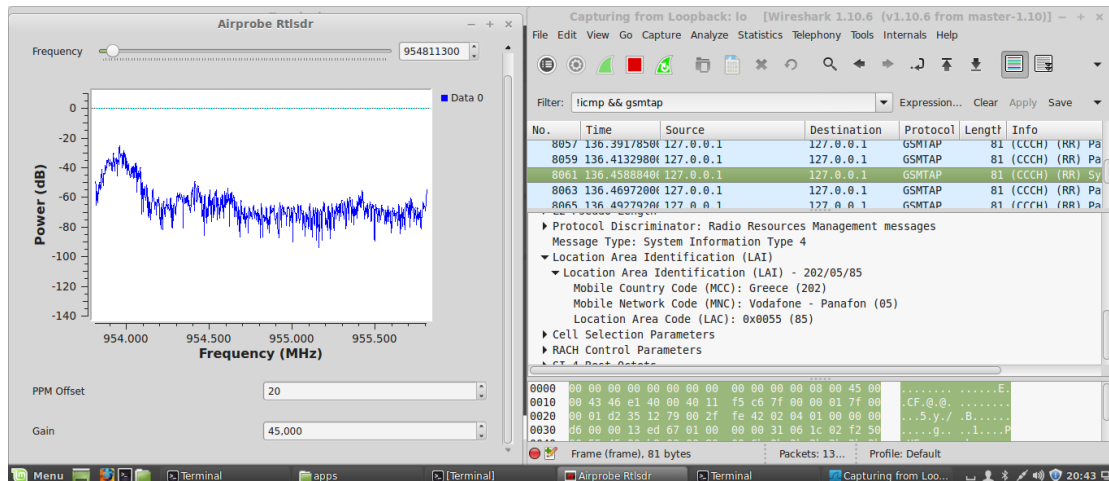


Figure 52 Airprobe RtlSDR & GSM packets on Wireshark

8.2 The Rtl-tool-kit Project

All this code is work in progress as the programmer said on his post to the program. So if we are not a programmer this might not work out of the box for us. If we are we should be able to fix errors to work for us. Go to our airprobe fork if we want to use a more advanced version of this with files

`./gsm_receive_rtl_mod.py`

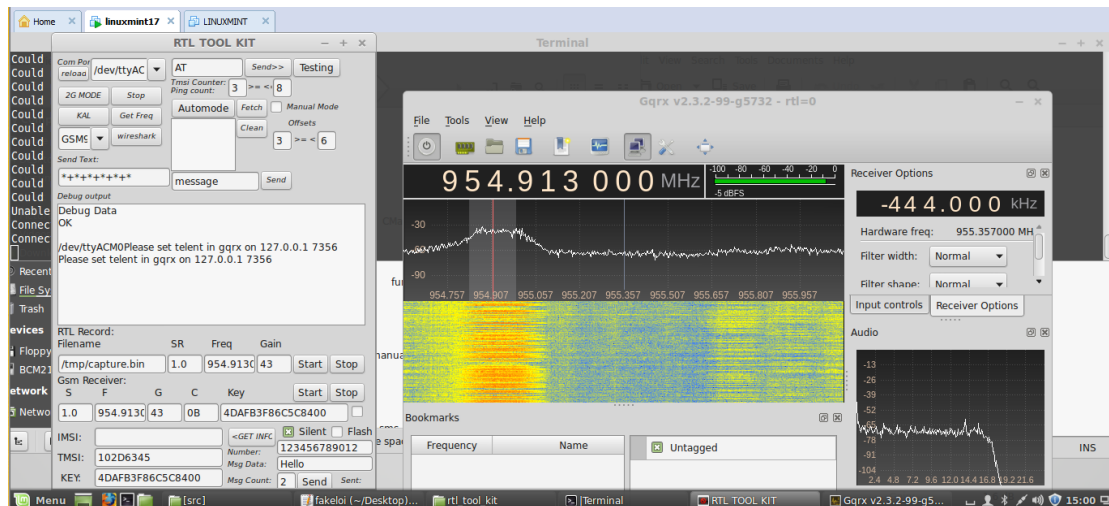


Figure 53. Rtl-Tool Kit While Running

Linking Tmsi now implemented for auto mode (Gets del-report timestamps from Sim card) Manual mode to be implemented next. Also shows Tmsi spammer function to monitor tmsi's spamming on the BTS. Need to implement filter next to filter out the Tmsi's that are spamming.

Also added function to see how many unique tmsi's are on the BTS (not showed on vid).Need to add threading so functions don't mess up main GUI thread.

Test Your Modem With This

<https://github.com/banjaxbanjo/SmsFuzzer>

WHAT HARDWARE AND SOFTWARE DO WE NEED

- Linux Mint 17 or whatever runs gr-gsm
- A cheap rtl dongle from ebay
- A usb dongle modem (search ebay for 3g usb dongle) This modem has to be able to use PDU mode (most do) The Modem we using for demos is Hauwei E173
- Some phones work as a usb modem, Samsung in particular, I'm using Samsung GT-5889i running Android V 2.3.6
- Usb modem is more accurate as a tmsi sniffer because it stores the delivery reports on the sim card and the timestamps are used from these reports. The phone doesnt do this at the moment so I create one when I send each sms.

WHAT CAN I DO WITH THIS APP?

- We can get Imsi or Tmsi and key of the device connected to your pc.

- We can send silent/flash sms
- We can connect/match tmsi to a mobile number if target is on the same BTS and in GSM900/2G mode.

BUTTONS

- 2G MODE = puts dongle in 2G mode
- GQRX = runs gqrX
- Get Freq = get correct freq from gqrX and set in the parameters box for airprobe/gr-gsm
- KAL = starts kal on option you set from drop down box.

Also have airprobe version which runs with extracted key so you can see data in wireshark live rather than having to create a bin file then convert it to a *.cfile.

Msg Count = how many sms to send

Sms Send Delay = how many seconds to send each sms

Del Report Delay = stop recording after x amount of seconds the app has time to get Delivery reports back(not needed for manual mode)

FUTURE CODE

- Start stop app when key changes for airprobe so you have the right key to decode sms data.
- Create an android app to send sms and create timestamps for people that haven't got a modem (not silent sms app just normal sms for a person you know and want to get tmsi)
- Connect Kraken to get key programmatically (not sure yet if its possible but would be nice) not live data of course.
- Wait for Poitr/gr-gsm to add decoding block so gr-gsm can be used for debugging like the airprobe version can with this app with the kc on other timeslots 2S etc.[24]

8.3 The Pytacle-Alpha2 Project

Pytacle Alpha2

Pytacle is a tool inspired by tentacle. It automates the task of sniffing GSM frames of the air, extracting the key exchange, feeding kraken with the key material and finally

decode/decrypt the voice data. All we need is a USRP (or similar) to capture the GSM band and a kraken instance with the berlin tables (only about 2TB) [25].

Changes: Support of RTLSDR sticks, possibility to scan for cells around you, and more.[26]

However with Pytacle this was different: it promised a lot of things to do (“[^]It automates the task of sniffing GSM frames of the air, extracting the key exchange, feeding kraken with the key material and finally decode/decrypt the voice data. “) but it failed to achieve anything like that. First of all I looked at the code, which was surprisingly simple compared to the feature list (for example GSM Analyzer tries to do around 2/3 of the stuff Pytacle promises and it has a large codebase consisting of many classes and files) and however we would never judge a program based on its source code’s size (that would truly be horrible) we were still getting skeptical of the single .py file containing mostly GUI definitions.

So after we read the approach of Domonkos Tomcsanyi for source-code we discovered the same comments:

1. It tries to script together the procedure already known to GSM hackers (capture stuff, then run it through gsm-receive.py, crack the key, decode the conversation), so you still need to have all the tools, but Pytacle tries to figure out a way for you to run them
2. and sadly it doesn’t do very well. It has magic numbers in it hardcoded (“0B” always for Configuration, “06 3f” for Immediate Assignment) which not seem to perform well (or to be honest: at all) in my environment.

we actually tried changing the configuration to 0C, which is the way mobile carriers operate in my country, and fed Pytacle a file with 5 different Immediate Assignments. It told me “No immediate assignments found, sorry...” and it exited.

So, to come to a conclusion: we really appreciate the effort David put into this script, but right now it is useless. It goes through the whole GSM cracking procedure, but it naturally lacks the human intelligence currently needed to distinguish between different Immediate Assignments, cell configurations etc. so it’s almost always destined to fail on you. It is possible to create such input files (basically cleaning out junk from a real-world file, or run a test-network and capture its traffic), which it can interpret, and work on, but that is not going to help someone who would like to try stuff on a real network.[27]

Conclusion

In this Master Thesis we try to coverage how the Hackers work for intercepting our phones simple and with low budget equipment. The Algorith that the GSM uses A5/1 is weakness and not accidental. We have to use our mobile phones with 3G or 4G signal because it offer more secure communication than GSM

- [1] "GNURadio," Introduction To GSM, [Online]. Available: https://gnuradio.org/redmine/projects/gnuradio/wiki/Introduction_To_GSM
- [2] Fabian van den Broek "Catching and Understanding GSM-Signals" in Radboud University Nijmegen ,2010
- [3] "HackRF One," HackRF One an open source SDR platform, [Online]. Available: <https://greatscottgadgets.com/hackrf/> . [Accessed 2 September 2015].
- [4] "Ettus Research" Ettus Research A National Instruments Company [Online]. Available: <http://www.ettus.com/> . [Accessed 2 September 2015].
- [5] "Wikipedia," Universal Software Radio Peripheral [Online]. Available: https://en.wikipedia.org/wiki/Universal_Software_Radio_Peripheral#cite_ref-9. [Accessed 2 September 2015].
- [6] Kanaiya Kanzaria1 , Sanjay S.C2 , " ACTIVE GSM MONITORING," International Journal of Computer Science and Information Technology Research , Vol. 2, Issue 2, pp: (484-494), Month: April-June 2014,
- [7] "LUSHPROJECTS BLOG" Spectrum Analysis and GSM Broadcast Decoding in 2013 [Online]. Available: <http://lushprojects.com/blog/2013/11/> [Accessed 2 September 2015].
- [8] "Gqrx SDR," Welcome to gqrx[Online], Available : <http://gqrx.dk/> [Accessed 2 September 2015].
- [9] "Airprobe", Welcome to Airprobe [Online]. Available: <https://svn.berlin.ccc.de/projects/airprobe> [Accessed 2 September 2015].
- [10] "GNURadio," Welcome to Airprobe , [Online]. Available: <http://gnuradio.org/redmine/projects/gnuradio/wiki> [Accessed 2 September 2015]
- [11] *Kalibrate tool* " KALIBRATE-RTL: CALIBRATE #SDR (SDR SHARP) LINUX/WINDOWS TUTORIAL" , [Online]. Available: <http://rtl-sdr.sceners.org/?tag=kalibrate> [Accessed 2015 February 19].
- [12] "Wikipedia," Wireshark [Online]. Available: <https://en.wikipedia.org/wiki/Wireshark> [Accessed 2 September 2015].
- [13] " RTL-SDR.com," RTL-SDR TUTORIAL: ANALYZING GSM WITH AIRPROBE/GR-GSM AND WIRESHARK, [Online]. Available: <http://www.rtl-sdr.com/rtl-sdr-tutorial-analyzing-gsm-with-airprobe-and-wireshark/> [Accessed 20 February 2015].

- [14] "Wikipedia," A5/1 [Online]. Available: <https://en.wikipedia.org/wiki/A5/1> [Accessed 2 September 2015].
- [15] "Frosty Hacks," Passive GSM interception Part 1 [Online]. Available: <http://frostyhacks.blogspot.gr/2015/04/gsm-wtf-bbq.html>. [Accessed 2 September 2015].
- [16] "Airprobe," Airprobe How-To [Online]. Available: <https://srlabs.de/airprobe-how-to> . [Accessed 2 September 2015].
- [17] "Nuzlan," Practical exercise on the GSM encryption A5/1 [Online]. Available: <https://lynxnuzlan.wordpress.com/2011/02/23/practical-exercise-on-the-gsm-encryption-a51> . [Accessed 2 September 2015].
- [18] "Ferran Casanovas " , Get Kc key and TMSI number! [Online]. Available: <https://ferrancasanovas.wordpress.com/2014/01/28/get-kc-key-and-tmsi-number/> [Accessed 2 September 2015].
- [19] "The big Gsm write-up how to capture ,analyze and Crack Gsm" How to do get the Kc from a SIM card [Online]. Available: <http://domonkos.tomcsanyi.net/?p=369> [Accessed 2 September 2015].
- [20] "GitHub," Sim card File System Access #96 [Online], Available : <https://github.com/SecUpwN/Android-IMSI-Catcher-Detector/issues/96> [Accessed 2 September 2015].
- [21] "The big Gsm write-up how to capture ,analyze and Crack Gsm" <https://ferrancasanovas.wordpress.com/cracking-and-sniffing-gsm-with-rtl-sdr-concept/> [Accessed 2 September 2015].
- [22] "Ferran Casanovas " , Cracking and sniffing GSM with a RTL-SDR [Online]. Available: <https://ferrancasanovas.wordpress.com/2014/01/28/get-kc-key-and-tmsi-number/> [Accessed 2 September 2015].
- [23] " Gr-Gsm ," The GrGsm Project , [Online]. Available: <https://github.com/ptrkrysik/gr-gsm> [Accessed 2 September 2015]
- [24] " RTL-TOOL-PROJECT, " banjaxbanjo/rtl_tool_kit , [Online]. Available: https://github.com/banjaxbanjo/rtl_tool_kit [Accessed 2 September 2015]
- [25] "Insinuator," Pytacle Alpha1 released [Online]. Available: <https://www.insinuator.net/2012/10/> [Accessed 2 September 2015].
- [26] "Insinuator," Pytacle Alpha2 [Online]. Available: <https://www.insinuator.net/2013/10/pytacle-alpha2/> [Accessed 2 September

2015].

- [27] "Pytacle," Going My Way [Online]. Available:
<http://domonkos.tomcsanyi.net/?p=459> [Accessed 9 September 2015].