# Sniffing, Decoding and Decryptionof GSM signals using Open Source Software and Low Cost Hardware

Muhammad Talha Choudary[1*],Arish Yaseen[1], Muhammad A Javaid[1], Abeer R Khan[1], Bilal A Khawaja[1], Sajid Saleem[1] and Muhammed Mustaqim[1]

[1] Department of Electronics and Power Engineering,PNEC-NUST, Karachi,Pakistan (* Corresponding author)
(talha.choudhry-ee@pnec.nust.edu.pk*, ssaleem@pnec.nust.edu.pk, mmustaqim@pnec.nust.edu.pk)

**Abstract**: We report- a software defined platform is used to sniff, decode and decrypt GSM signals. A RTL-SDR [8] along with GR-GSM and Wireshark is used to decrypt GSM signals. This approach eliminates the need of GPUs and also provides us decoded GSM signals in real time. In addition, this approach enabling applications in telecommunication to monitor GSM signals in real time also useful for military purposes. This approach is discussed on the basis of experimental results.

**Keywords:** GSM signal, TMSI, KC, GR-GSM, RTL-SDR (Software Defined Radios), A5/1 A5/3 encryption.

## I. INTRODUCTION

The decryption of GSM signals is becoming increasingly important for defense and security applications. Moreover, the GSM is used by nearly 4.9 billion people [1] their privacy is real concern.

Traditionally the decryption is done by highly expensive hardware and also this process was time consuming as well as hit rate was very low. First known attack on GSM was launched back in 2003 [2].

An alternative approach to decrypt GSM signals with the help of RTL-SDR dongle and by using open source software has been discussed.

A5/1 is the encryption [3] type which is used commonly to encrypt the data packages sent our mobile phones.A5/1 is a stream cipher algorithm which uses three LFSR (Linear Feedback Shift Registers) to encode the data.

## II. OUR WORK

**Pre-Requisites:**

To start GSM sniffing and decoding there were some pre-requisite hardware and software requirements we needed to fulfill. The hardware requirements were a personal computer, a RTL-SDR dongle with antenna, a USB cable and a mobile phone with active network.The software requirements were Linux based virtual machine, GR-GSM, Wire Shark & GNU Radio [4]-[6].

**Process:**

To start with the process of sniffing it was required to make a change in the settings of the mobile we are using i.e. change network setting to ‘only 2G’.

Next step was to identify the downlink channels our mobile is using to receive data. There were two methods which we could have used for that, one is the hit and trail method using software called GQRX, Fig 1 and second is using a website [7]in which we use the ARFCN(Absolute Radio Frequency Channel Number) from our mobile phone to find the downlink frequency, Fig 2. For further process we required the access to virtual machine I mentioned earlier. The virtual machine we used was KALI LINUX 32-BIT. First thing required after installing and setting up KALI was to install GQRX. After installation and connecting our RTL-SDR to the virtual machine, Fig 3 GQRX was used. We adjusted the frequency in the upper bar and checked if we can catch any downlink frequencies frequency which showed in the window below. The darker the color in the below window the stronger the downlink channel.

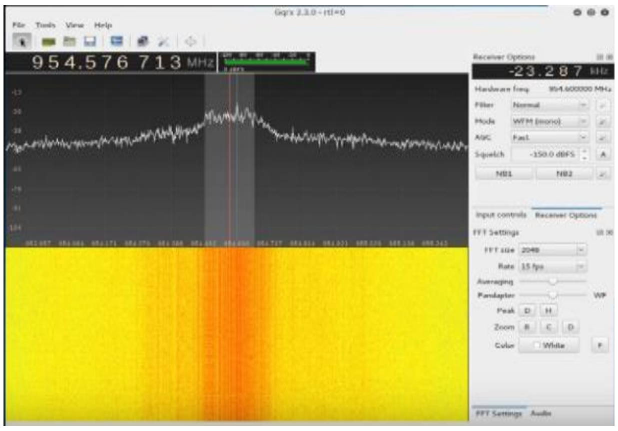
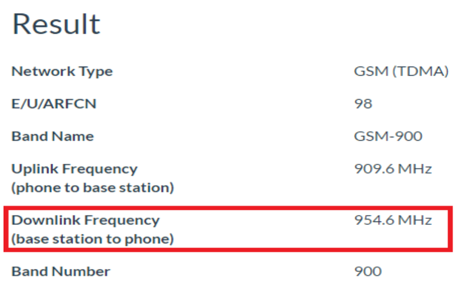


Fig. 1 GQRX Method



Result

| | |
|---|---|
| Network Type | GSM (TDMA) |
| E/U/ARFCN | 98 |
| Band Name | GSM-900 |
| Uplink Frequency (phone to base station) | 909.6 MHz |
| Downlink Frequency (base station to phone) | 954.6 MHz |
| Band Number | 900 |

Fig. 2 Website Method

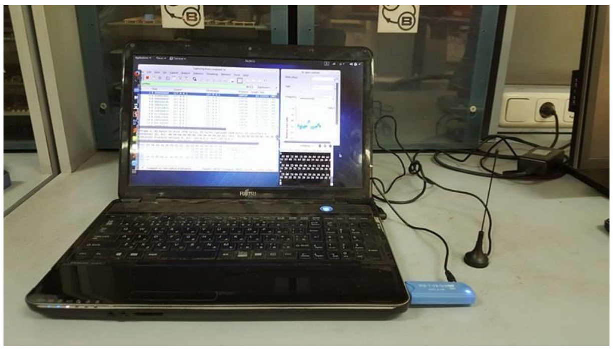
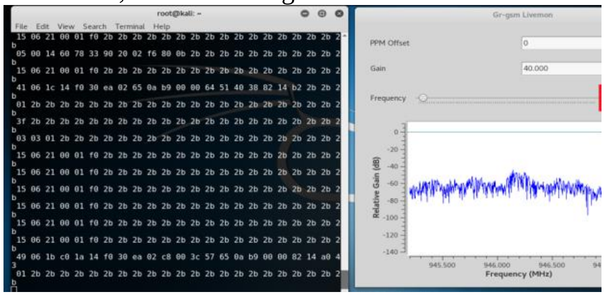


Fig. 2.1 Hardware Set up

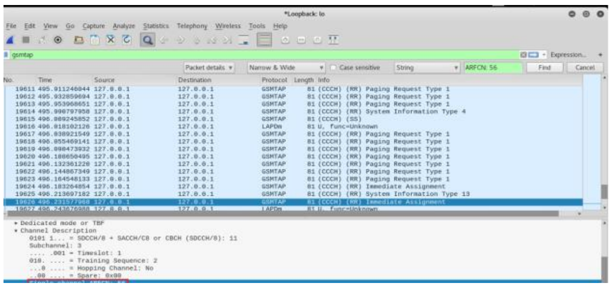
The next step was to install the GR-GSM, available on the internet, on the virtual machine. The easy install guide was available on the website which we used to install all the packages required. [6]

After installation, GR-GSM was used with WireShark to sniff out GSM data packages from air and decode them [7]. To do this we opened WireShark and selected loop back and clicked start. We then entered the command 'gr-gsm_Livemon' which opened the live monitoring of GSM signals in the air. All we had to do then was to select the downlink frequency we know and we started catching data packages which were shown in WireShark, as shown in fig 3 and 4



```
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 05 00 14 60 78 33 90 20 02 f6 80 0b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 41 06 1c 14 f0 30 ea 02 65 0a b9 00 00 64 51 40 38 82 14 b2 2b 2b 2
b
 01 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 3f 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 03 03 01 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 15 06 21 00 01 f0 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
 49 06 1b c0 1a 14 f0 30 ea 02 c8 00 3c 57 65 0a b9 00 00 82 14 a0 4
3
 01 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2b 2
b
```

Fig 3. GR-GSM sniffing GSM Data



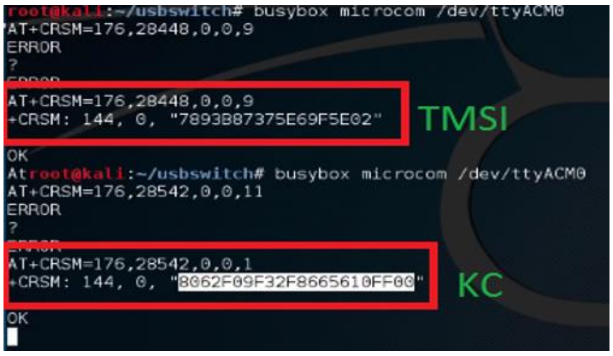
```
No.   Time          Source     Destination  Protocol Length Info
19611 495.911246844 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19612 495.932859694 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19613 495.953968651 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19614 495.990797958 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) System Information Type 4
19615 496.009245852 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (SS)
19616 496.018102126 127.0.0.1  127.0.0.1    LAPDm    81 U, func=Unknown
19617 496.038921549 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19618 496.055469141 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19619 496.090473932 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19620 496.100650495 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19621 496.132361228 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19622 496.144867349 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19623 496.164548133 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Paging Request Type 1
19624 496.183264854 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Immediate Assignment
19625 496.213697182 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) System Information Type 13
19626 496.231577968 127.0.0.1  127.0.0.1    GSMTAP   81 (CCCH) (RR) Immediate Assignment
19627 496.243676988 127.0.0.1  127.0.0.1    LAPDm    81 U, func=Unknown

 ▸ Dedicated mode or TBF
 ▾ Channel Description
     0101 1... = SDCCH/8 + SACCH/C8 or CBCH (SDCCH/8): 11
     Subchannel: 3
     .... .001 = Timeslot: 1
     010. .... = Training Sequence: 2
     ...0 .... = Hopping Channel: No
     ..00 .... = Spare: 0x00
```

Fig 4. WireShark has showing decoded GSM Data.

The packages caught using this method were of multiple types and provided different valuable information such as the area code, the channel used etc.

For the next step we connected our mobile phone to the computer and connected it with our virtual machine. Although our mobile was in MTP (media transfer protocol) mode, we used 'lsusb –v|less' command to get additional information. In that information it was found that our device can work as abstract modem and support AT-commands, which could be used extract the TMSI and KC [4]. For that a small program called USBswitcher was used which switches the USB's (i.e. our mobile) configuration that allowed us to use the mobile as modem (using ttyl command). It is very dangerous to play with the device as modem as we may brake our sim cards but if we know what command to use specifically we can extract the TMSI and KC.

While accessing the modem we usedthe commands "AT+CRSM=176.28448.0.0.9" to get KC and "AT+CRSM=176.28543.0.0.1" to get TMSI, Fig 5. These values of TMSI and KC are temporary and may change with change in channel or with different type of data (new call or sms). So after each sms or phone call it is necessary to check the TMSI and KC to see if they had changed or remained the same. To decrypt only the first four characters of TMSI are required whereas we require the whole KC except the last two characters so we would remove them.



```
root@kali:~/usbswitch# busybox microcom /dev/ttyACM0
AT+CRSM=176,28448,0,0,9
ERROR
?
ERROR
AT+CRSM=176,28448,0,0,9
+CRSM: 144, 0, "7893B87375E69F5E02"
OK
Atroot@kali:~/usbswitch# busybox microcom /dev/ttyACM0
AT+CRSM=176,28542,0,0,11
ERROR
?
ERROR
AT+CRSM=176,28542,0,0,1
+CRSM: 144, 0, "8062F09F32F8665610FF00"
OK
```

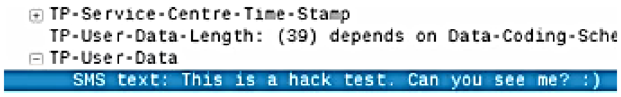
Fig. 5 KC and TMSI

**Decrypt SMS:**

Before initializing the sms decryption process it was made sure that the mobile was working as a modem and also the current TMSI and KC were extracted and noted down.

The command of 'grgsm_capture.py' was used to capture the sms package being sent to our mobile. This command is also required to give additional values with the command. In those additional value 'g' is the gain (or amplification of the signal), 'a' is the channel our mobile is using, 's' is sample rate, 'c' is the destination file where the data will be saved and 'T' is the time period of sampling in seconds.

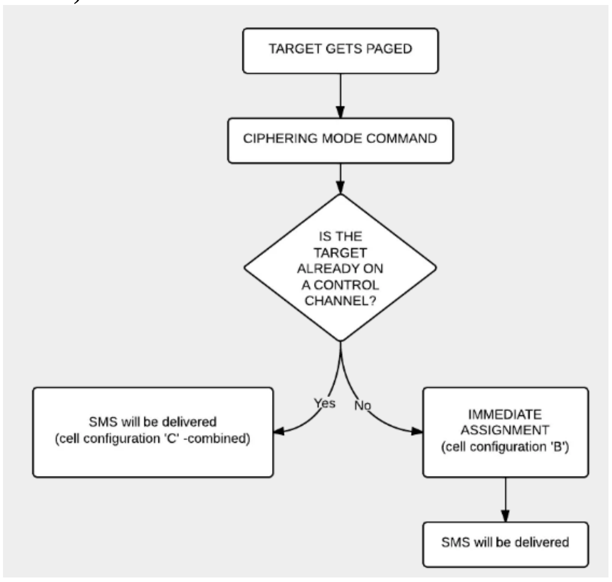
After the file was captured, 'grgsm_decode' command was used with WireShark (in loopback mode) first time without the key to identify the encryption(whether A5/1 or A5/3), mode and timeslotused. It is necessary to recheck TMSI and KC as they may have changed from their previous values. In WireShark TMSI was used to identify the packages that were sent to our device. Decode command was used second time with the key to decrypt the sms we have sent, Fig 6. Shows The decrypted sms was shown in WireShark window. 'grgsm_decode' command required 'a' and 's' are same, 'c' is now the source file, 'm' is the mode of communication being used by our mobile (it is default for all), ‘t’ is the time slot, ‘e’ is the encryption and ‘k’ is the KC.



```
⊞ TP-Service-Centre-Time-Stamp
  TP-User-Data-Length: (39) depends on Data-Coding-Sche
⊟ TP-User-Data
     SMS text: This is a hack test. Can you see me? :)
```

Fig. 6 Decrypted sms.

Fig 7 is 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):

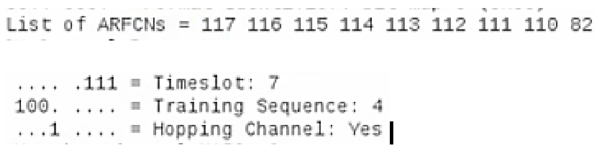


TARGET GETS PAGED
CIPHERING MODE COMMAND
IS THE TARGET ALREADY ON A CONTROL CHANNEL?
Yes
No
SMS will be delivered (cell configuration 'C' -combined)
IMMEDIATE ASSIGNMENT (cell configuration 'B')
SMS will be delivered

Fig 7 Flowchart for decryption of SMS

**Decrypt call:**

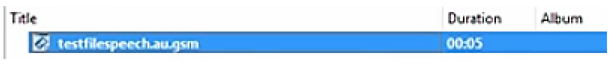
To decrypt the call, the method is pretty much the same as decrypting a sms but there are some limitations. Similar to the decrypting a sms it was necessary to first find TMSI and KC, then we used the similar grgsm_capture command to capture the data packets and save them into a destination file and then decode that file using grgsm_decode command. However when we looked at the packages that were caught, it was found that ‘channel hopping’ was on. Channel Hopping means that our data was received through multiple channels by switching between them during the call. So to receive the whole data package now, it was required to capture data multiple frequencies. This is where the limitation comes because RTL SDR [8] can capture the maximum bandwidth of 2.4 MHz or 3.2 MHz (with probable package loss). So if the bandwidth of channel hopping used by our mobile is higher than this it is not possible to decipher the whole call unless a more powerful device is used.



```
List of ARFCNs = 117 116 115 114 113 112 111 110 82

 .... .111 = Timeslot: 7
 100. .... = Training Sequence: 4
 ...1 .... = Hopping Channel: Yes
```

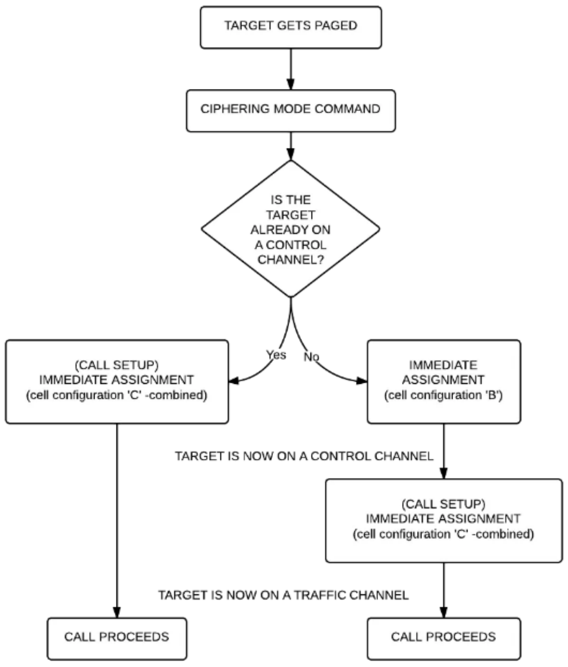
Fig. 9 Hopping Channels

However if the Channel Hopping was off it will be much simpler to decode it and will be almost same as sms decryption. To do that capture command will be used to capture data on the particular channel (as channel hopping is off) and then decode command on the captured file will be used. First time decode command will be used without the key to identify the encryption type, the mode and the time slot. Then we will use the decode command again with the key and all that information we got and create an output file with extension ‘.au.gsm’. This will be the audio file of our call and that’s how the call will be decrypted, Fig 8.
Flow chart for voice decryption has shown in Fig 9



| Title | Duration | Album |
|---|---|---|
| testfilespeech.au.gsm | 00:05 | |

Fig. 8 Audio file of our Call



TARGET GETS PAGED
CIPHERING MODE COMMAND
IS THE TARGET ALREADY ON A CONTROL CHANNEL?
Yes
No
(CALL SETUP) IMMEDIATE ASSIGNMENT (cell configuration 'C' -combined)
IMMEDIATE ASSIGNMENT (cell configuration 'B')
TARGET IS NOW ON A CONTROL CHANNEL
(CALL SETUP) IMMEDIATE ASSIGNMENT (cell configuration 'C' -combined)
TARGET IS NOW ON A TRAFFIC CHANNEL
CALL PROCEEDS
CALL PROCEEDS

Fig 9 Flowchart for voice decryption

## III. CONCLUSION

Concluding our work here, we can say that we were successful in using the RTL-SDR to sniff and decode the GSM data off the air. However there were some limitations such as Channel hopping during the phone calls which caused hindrance in the decoding process. The software we required were easily available and did not cause any issues while working. However a sound knowledge of working in a Linux environment was required in order get this job done successfully.

Secondly we also required access to our mobile phone so we could use is as a modem and extract KC and TMSI. However, now we are working on creating our on Rainbow Tables that will allow us to extract KC and TMSI without the requirement of access to the mobile phone.

To generate our own Rainbow Tables for A5/1 we will use a custom built computer with two GPUs (Graphic Processing Units), one for table generation and second for doing look up in the table. Once these tables will be generated we will be able to decipher any data package with A5/1 encryption without the need to access the receiving device.

## IV. REFERENCES

[1] Definitive data and analysis for the mobile industry URL: https://www.gsmaintelligence.com/. Last accessed Dec 11, 2016.

[2] Elad Barkan, Eli Biham, Nathan Keller, Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication, 2003

[3] J. Golic, W. Fumy, "Cryptanalysis of Alleged A5 Stream Cipher" in Advances in Cryptology EUROCRYPT 1997. LNCS, Heidelberg:Springer, vol. 1233, pp. 239-255, 1997.

[4] Jean-Philippe Lang, Overview of GNU Radio URL: http://gnuradio.org/redmine/projects/gnuradio

[5] Communications System Toolbox Support Package for RTL-SDR Radio. User Guide.MathWorks, 2013.

[6]gr-gsm Documents on Github URL: https://github.com/ptrkrysik/grgsm/tree/master/include/grgsm

[7] ARFCN Calculator URL: https://www.cellmapper.net/arfcn Last accessed Jan 11, 2017.

[8] R. W. Stewart et al., Software Defined Radio using the MATLAB & Simulink and the RTL-SDR, Strathclyde Academic Media, 2015. ISBN-13: 978-0-9929787-1-6.