

# Evaluating GSM A5/1 security on hopping channels

Bogdan Diaconescu

v1.0

This paper is a practical approach on evaluating A5/1 stream cipher on a GSM hopping network air interface called Um. The end goal is to evaluate how easy is to break A5/1 with COTS hardware and open source software.

What this paper is NOT about: a full description of GSM standard and A5/1 stream cipher.

## *About GSM hopping channels.*

This work is based on the previous work that resulted in Airprobe being able to decode the GSM logical channels when a capture of IQ data is provided for a specific ARFCN. We modified Airprobe to support hopping channels by taking IQ data samples from files corresponding to each ARFCN of interest.

GSM slow frequency hopping is intended to reduce the adjacent cells interference by switching between a set of ARFCNs that are orthogonal to the set used by own cell. The whole set of ARFCNs used by the cell is broadcast by BTS on BCCH channel in SI Type 1 message and the hopping parameters are sent from BTS to MS in an RR Immediate assignment message.

## **Method used for frequency hopping:**

1. Capture spectrum so that all the needed ARFCNs for hopping will be included into the bandwidth.
2. Channelize and downsample the spectrum in order to obtain a data stream for each needed ARFCN. A PFB (Poliphase Filter Bank) is used for that.
3. Identify your call and in turn identify the hopping parameters.
4. Feed the modified gsm-receiver from airprobe with the hopping parameters.
5. Run the same steps as in the case of non hopping cell. For traffic channel there could be a different set of hopping parameters.

**Prerequisites:** Use USRP N210 or USRP2 from Ettus. For this work USRP N210 has been used. It can acquire 25Mhz bandwidth from the spectrum without any FPGA changes or other software changes on PC.

## ***Step 1: Capture the data on the main ARFCN***

The main ARFCN for the cell would be the one with them maximum signal strength and this can be determined with kal tool [Ref 4]

```
sudo kal -s GSM900
```

```
kal: Scanning for GSM-900 base stations.
```

```
chan: 33 (941.6MHz + 234Hz) power: 10791.92
```

Therefore I have my main cell physical channel on ARFCN 33 and in turn the parameters for the engine needs to be set in *capture\_decode\_channelize2.sh*:

```
CONFIGURATION="0B"
```

```
CA="33"
```

*./capture\_decode\_channelize2.sh > out.txt* will produce the data streams for channel 33 and output of decoded

messages in Wireshark.

**Step 2: Get all CA channels and determine cell hopping parameters for the call**

Determine Cell Allocation(CA) from System Information Type 1: 12 22 33 42 49 54

The image shows a Wireshark capture of a GSM CCCH - System Information Type 1 message. The packet list pane shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
120	2.738997000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
121	2.766104000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
122	2.784147000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
123	2.815724000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) System Information Type 1
124	2.833770000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) System Information Type 13
125	2.860876000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1

The packet details pane for packet 123 shows the following structure:

- ..00 0000 0000 0000 = ARFCN: 0
- .0.. .... .... .... = Uplink: 0
- Signal/Noise Ratio (dB): 0
- Signal Level (dBm): 0
- GSM Frame Number: 640565
- Channel Type: BCCH (1)
- Antenna Number: 0
- Sub-Slot: 0
- GSM CCCH - System Information Type 1**
  - ▶ L2 Pseudo Length
  - ▶ Protocol Discriminator: Radio Resources Management messages
    - Message Type: System Information Type 1
  - ▼ Cell Channel Description
    - 00.. 000. = Format Identifier: bit map 0 (0x00)
    - List of ARFCNs = 54 49 42 33 22 12
  - ▶ RACH Control Parameters
  - ▶ SI 1 Rest Octets

The packet bytes pane shows the raw hex and ASCII data:

```

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E.
0010 00 43 46 7e 40 00 40 11 f6 29 7f 00 00 01 7f 00 .CF~@.@. ....
0020 00 01 9f 40 12 79 00 2f fe 42 02 04 01 00 00 00 ...@.y/.B.....
0030 00 00 00 09 c6 35 01 00 00 00 55 06 19 00 00 00 .....5..U.....
  
```

Then next step is to determine SDCCH channel Timeslot and hopping parameters from Immediate Assignment message: channel Timeslot=1; Subchannel=0; MAIO=0; HSN=2; MA=07

Now run again the script in order to obtain all channels:

```

CONFIGURATION="0B"           # Timeslot 1
CA="12 22 33 42 49 54"      # Cell Allocation in increasing order
C0="33"                     # C0 ARFCN
MA="07"                     # Mobile Allocation – ARFCNs used for this channel
MAIO=0                      # Mobile Allocation Index Offset
HSN=2                       # Hopping Sequence Number
  
```

KEY="00 00 00 00 00 00 00 00"

./capture\_decode\_channelize2.sh > out.txt

The screenshot shows the Wireshark interface with a list of packets and a detailed view of packet 147. The packet list shows several GSMTAP packets between 127.0.0.1 addresses, including Immediate Assignments and Paging Request Type 1 messages. The detailed view for packet 147 shows it is a Radio Resources Management message, specifically an Immediate Assignment. It includes details for Page Mode, Dedicated mode or TBF, Channel Description (SDCCH/8 + SACCH/C8 or CBCH), Request Reference, Timing Advance, and Mobile Allocation (Length: 1). A highlighted orange bar indicates the 'Bitmap of increasing ARFCNs included in the Mobile Allocation: 11100000'. Below this, the 'IA Rest Octets' are shown in hexadecimal and ASCII format.

No.	Time	Source	Destination	Protocol	Length	Info
147	3.366545000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Immediate Assignment
148	3.384593000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
149	3.411709000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Immediate Assignment
150	3.429758000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
151	3.456874000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1
152	3.474932000	127.0.0.1	127.0.0.1	GSMTAP	81	(CCCH) (RR) Paging Request Type 1

► Protocol Discriminator: Radio Resources Management messages  
 Message Type: Immediate Assignment  
 ► Page Mode  
 ► Dedicated mode or TBF  
 ▼ Channel Description  
 0100 0... = SDCCH/8 + SACCH/C8 or CBCH (SDCCH/8), Subchannel 0  
 .... .001 = Timeslot: 1  
 110. .... = Training Sequence: 6  
 ...1 .... = Hopping channel: Yes  
 Hopping channel: MAIO 0  
 Hopping channel: HSN 2  
 ► Request Reference  
 ► Timing Advance  
 ▼ Mobile Allocation  
 Length: 1  
 Bitmap of increasing ARFCNs included in the Mobile Allocation: 11100000  
 ► IA Rest Octets

```

0030  00 00 00 09 c6 af 02 00 00 00 31 06 3f 00 41 d0  .....1.?A.
0040  02 f2 18 79 01 01 07 2b 2b 2b 2b 2b 2b 2b 2b 2b  ...y..+ ++++++
0050  2b                                             +
  
```

○ Text item (text), 1 byte      ≡ Packets: 541 Displ...      ≡ Profile: Default

**Step 3: Decode the SDCCH channel**

```

CONFIGURATION="1S"           # Timeslot 1
CA="12 22 33 42 49 54"      # Cell Allocation in increasing order
C0="33"                     # C0 ARFCN
MA="07"                     # Mobile Allocation – ARFCNs used for this channel
MAIO=0                      # Mobile Allocation Index Offset
HSN=2                       # Hopping Sequence Number
KEY="00 00 00 00 00 00 00 00" # KC – zero for now
  
```

./capture\_decode\_channelize2.sh > out.txt

will produce Wireshark data for the frames that can be decoded and the rest of the data in out.txt.

Note: gsm-receiver will produce output for all the sub-slots currently on the SDCCH so it will be useful to filter the messages in Wireshark based on the known sub-slot from the Immediate Assignment command. In our case the sub-slot value is 0.

The screenshot shows the Wireshark interface with the following details:

- Filter:** gsmtap.sub\_slot==0
- Table of Frames:**

No.	Time	Protocol	Length	Info
9	3.430269000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 6
10	3.517773000	LAPDm	81	U, func=UI
12	3.753124000	LAPDm	81	U F, func=UA(DTAP) (MM) CM Service Request
13	3.901189000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5
15	3.989026000	LAPDm	81	I, N(R)=1, N(S)=0(DTAP) (RR) Classmark Enquiry
17	4.225851000	LAPDm	81	I, N(R)=1, N(S)=1(DTAP) (MM) Authentication Request
18	4.374474000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5ter
22	4.697638000	LAPDm	81	S, func=RR, N(R)=3
26	4.845566000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 6
29	4.933395000	LAPDm	81	I, N(R)=3, N(S)=2(DTAP) (RR) Ciphering Mode Command
- Packet Details (Selected Frame 29):**
  - .0.. .... = uplink: 0
  - Signal/Noise Ratio (dB): 0
  - Signal Level (dBm): 0
  - GSM Frame Number: 640697
  - Channel Type: SACCH/8 (136)
  - Antenna Number: 0
  - Sub-Slot: 0
  - ▼ SACCH L1 Header, Power Level: 5, Timing Advance: 1
    - ...0 0101 = MS power level: 5
    - .... .1.. = FPC: In use
    - Actual Timing Advance: 1
  - ▶ Link Access Procedure, Channel Dm (LAPDm)
  - ▶ GSM A-I/F DTAP - System Information Type 6
- Packet Bytes:**

```

0030  00 00 00 09 c6 b9 88 00 00 00 05 01 03 03 2d 06  .....
0040  1e 81 a7 22 f6 10 2b 83 93 ff 2b 2b 2b 2b 2b 2b  ...".+. .+++++
0050  2b

```
- Summary:** GSM Radiotap (gsmtap), 2 bytes | Packets: 96 Displa... | Profile: Default

The RR Ciphering Mode Command is the last message sent in clear text before the encryption is enabled. The encrypted bursts are to be found in out.txt in the form

```

Cx 488134 754090:
011100011000110111000111000111000110001100011000101100011100111100001010011110101010101...
Px 488134 754090:
011100011000110111000111000111000110001100011000101100011100111100001010011110101010101...
Sx 488134 754090:
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000...

```

where Cx is the cipher text bits, Px is the plain text bits and Sx is cipher stream. If x==1 then the burst in question is the first burst from the 4 ones that make up a frame.

#### **Step 4: Finding Kc**

The procedure used to find Kc based on rainbow tables can be found here [Ref 5]. The general idea is to know a plain text together with its position inside cipher text stream. In our case it can be System Information Type 5, System Information Type 5Ter, System Information Type 6 or any other known messages.

These messages are usually sent at fixed intervals and based on this information one can guess the frame numbers where message supposed to be in the crypted stream.

Suppose you have the known plain-text at frame number 641105, it actually is all over four frames: 641102, 641103, 641104, 641105:

If you know what is the plain-text message there then use gsmframecoder to generate the four bursts that make up the message:

```
./gsmframecoder message → burst1 burst 2 burst 3 burst4
```

Then use xor.py to generate the cipher stream:

```
xor.py C1 burst1 → results rs1
xor.py C0 burst2 → results rs2
xor.py C0 burst3 → results rs3
xor.py C0 burst0 → results rs4
```

Then use the result from the xor operations as input to crack tool:

```
crack rs1
crack rs2
crack rs3
crack rs4
```

In my specific case I get something on third burst: Found ccc6de6b6e3d3bb0 @ 22 #7 (table:250)

Then use find\_kc to actually determine the Kc:

```
./find_kc ccc6de6b6e3d3bb0 22 754765
##### Found potential key (bits: 22)#####
251f71485a37e2af -> 251f71485a37e2af
Framecount is 754765
KC(0): 2f ce 35 dc 3e 09 20 3c mismatch
...
KC(30): 49 b2 90 6c 98 5f ce 2b mismatch
KC(31): 88 0b 84 cd 8f 24 74 00 mismatch
```

If there is another frame known then find\_kc can match the exact Kc from the list above:

```
./find_kc ccc6de6b6e3d3bb0 22 754765 754798
100010110101000001010011001101011001001100010001000111110010100000101111110100010010100011
10110100100001000011001
```

```
##### Found potential key (bits: 22)#####
251f71485a37e2af -> 251f71485a37e2af
Framecount is 754765
KC(0): 2f ce 35 dc 3e 09 20 3c mismatch
...
KC(30): 49 b2 90 6c 98 5f ce 2b mismatch
KC(31): 88 0b 84 cd 8f 24 74 00 *** MATCHED ***
```

Now changing the KEY="00 00 00 00 00 00 00 00" to KEY="88 0b 84 cd 8f 24 74 00" and running again ./capture\_decode\_channelize2.sh > out.txt will reveal the messages on SDCCH after the ciphering started:

The screenshot displays the Wireshark interface with the following details:

- Filter:** gsmtap.sub\_slot==0
- Packet List:**

No.	Time	Protocol	Length	Info
1	0.000000000	LAPDm	81	U, func=UI
2	0.148808000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5
3	0.237061000	LAPDm	81	S, func=RR, N(R)=4
4	0.473744000	LAPDm	81	S, func=RR, N(R)=5
5	0.622227000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5ter
6	0.710396000	LAPDm	81	I, N(R)=5, N(S)=3(DTAP) (CC) Call Proceeding
7	0.945963000	LAPDm	81	I, N(R)=5, N(S)=4 (Fragment)
8	1.094512000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 6
9	1.182320000	LAPDm	81	I, N(R)=5, N(S)=5(DTAP) (RR) Assignment Command
- Packet Details (Packet 9):**
  - Channel Description 2 - Description of the First Channel, after time
  - 0000 1... = TCH/F + FACCH/F and SACCH/F
  - .... .111 = Timeslot: 7
  - 110. .... = Training Sequence: 6
  - ...1 .... = Hopping channel: Yes
  - Hopping channel: MAIO 2
  - Hopping channel: HSN 2
  - Power Command
  - ▼ Frequency List - Frequency List, after time
    - Element ID: 5
    - Length: 16
    - 00.. 000. = Format Identifier: bit map 0 (0x00)
    - List of ARFCNs = 33 22 12
  - Channel Mode - Mode of the First Channel(Channel Set 1)
- Packet Bytes:**

```
0000 06 2e 0f d0 82 05 05 10 00 00 00 00 00 00 00 00 .....
0010 00 00 00 01 00 20 08 00 63 01 ..... C.
```
- Frame (81 bytes) | Reassembled LAPDm (26 bytes)**
- Text item (text), 16 bytes | Packets: 10 Displa... | Profile: Default**

From the decrypted SDCCH channel it is easy to observe the Assignment Command that is sent by the BTS to MS in order to assign the traffic channel. The important parameters are traffic channel Timeslot and hopping parameters for the traffic channel.

*Note:* the traffic channel can have different hopping parameters than SDCCH channel. Although the MA in this

case is the same, it can be quite different than the MA used in the SDCCH channel.

### Step 3: Decode the traffic channel

The saga continues with another run of `./capture_decode_channelize2.sh > out.txt`. This is necessary to decode the traffic channel and obtain the audio stream and the configuration is presented below. One could also see in Wireshark the SI messages sent on the FACCH channel that share the same Timeslot with TCH.

```
CONFIGURATION="7T"           # Timeslot 7 for traffic
CA="12 22 33 42 49 54"      # Cell Allocation in increasing order
C0="33"                     # C0 ARFCN
MA="07"                     # Mobile Allocation
MAIO=2                      # Mobile Allocation Index Offset
HSN=2                      # Hopping Sequence Number
KEY="88 0b 84 cd 8f 24 74 00" # KC
```

The screenshot shows the Wireshark interface with a list of captured packets. The selected packet (No. 16) is expanded to show the GSM TAP header details.

No.	Time	Protocol	Length	Info
7	1.910103000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5
8	2.395049000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5ter
9	2.873688000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 6
10	3.352363000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5
11	3.831181000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5ter
12	4.029136000	LAPDm	81	I, N(R)=1, N(S)=0 (Fragment)
13	4.167334000	LAPDm/GSI	81	I, N(R)=1, N(S)=1(DTAP) (CC) Alerting (GSM MAP) invoke notify
14	4.310138000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 6
15	4.788698000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5
16	5.267459000	LAPDm	81	U, func=UI(DTAP) (RR) System Information Type 5ter

**GSM TAP Header, ARFCN: 0 (Downlink), TS: 7, Channel: FACCH/F (0)**

```
Version: 2
Header length: 16 bytes
Payload Type: GSM Um (MS<->BTS) (1)
Time Slot: 7
..00 0000 0000 0000 = ARFCN: 0
.0.. .... .... .... = Uplink: 0
Signal/Noise Ratio (dB): 0
Signal Level (dBm): 0
GSM Frame Number: 642246
Channel Type: FACCH/F (9)
```

0000 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00 .....E.  
0010 00 43 72 74 40 00 40 11 ca 33 7f 00 00 01 7f 00 .Crt@.@. .3.....  
0020 00 01 2b f5 12 70 00 2f fe 42 02 04 01 07 00 00 y / P

Frame (81 bytes) Reassembled LAPDm (24 bytes)

○ Frame (frame), 81 bytes      ☰ Packets: 19 Displaye      Profile: Default



At this moment the voice stream will be found in speech.au.gsm file that can be listen with the following:

```
untoast speech.au.gsm
mplayer speech.au      # or other player
```

### **Conclusion:**

1. The security of the GSM hopping channels is at the same level with the security of the non hopping channels, the only difference an attacker will encounter being the computing and storage resources needed to channelize and produce the data for each ARFCN.
2. We observed a very rapid degrading of the ability of Airprobe to correctly decode the GSM messages when the overall SNR degrades under -60dB for C0 channel, making necessary for a better demodulator.
3. The L1 header of the known plaintext (SI message) could change in both MS power level when FPC is in use and Timing Advance when MS is moving. Therefore some prediction is necessary to be used in order to correctly guess the known plain text.

### **References:**

1. **Airprobe git:** [git://git.gnumonks.org/airprobe.git](https://git.gnumonks.org/airprobe.git)
2. **Hopping channels description GSM 05.02 chapter 6**
3. [http://srlabs.de/research/decrypting\\_gsm/](http://srlabs.de/research/decrypting_gsm/)
4. **Kal tool git:** [git://github.com/ttsou/kalibrate-uhd.git](https://github.com/ttsou/kalibrate-uhd.git)
5. <http://lists.lists.reflexor.com/pipermail/a51/2010-July/000688.html>