Faculty of Engineering
Department of Computer Science
Chair of Communication Systems
Prof. Dr. Gerhard Schneider
Konrad Meier, Dennis Wehrle

Freiburg, 23. Februar 2011

Practical exercise on the GSM Encryption A5/1

Due to the request of some students we are today dealing with encryption in GSM. The cipher we are using today is called A5/1. It is a stream cipher which is used to secure data transmitted over the air interface U_m.

Task 1 - Decoding a capture file
The following step by step guide is taken from the How-To from srlabs:
`http://srlabs.de/uncategorized/airprobe-how-to/`

1. Download this capture file:
   `http://reflextor.com/vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile.gz`
   Extract the capture file.

2. Start wireshark and listen to your network interface. Use the following capture filter: "port 4729" (Hint: Ctrl+i, then click Option on eth0 and enter the capture filter)

3. Decode the capture file with the script `cd airprobe/gsm-receiver/src/python/`
   `./go_usrp2.sh vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile`
   You should see a lot of decoded packets in wireshark.

4. 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.

5. 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_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile 174 1S`
   The decoder now only decodes TimeSlot 1 and ignores TimeSlot 0.
   Note: Because this is a USRP2 capturefile, "174" has to be used as the decimation rate.

6. All data on TimeSlot1 which are not encrypted are shown in wireshark. You should remember that the encryption is enabled after the "Ciphering Mode Command". Therefore all data after

this command is encrypted and not shown in wireshark. To see and decode this data we have to find the key K_c.

7. All encrypted bursts are shown in your console like this.

```
C1 862344 1332354: 001111000110000100101110000110111111100010111100010100101111...
P1 862344 1332354: 001111000110000100101110000110111111100010111100010100101111...
S1 862344 1332354: 000000000000000000000000000000000000000000000000000000000000...
C0 862345 1332387: 010001100001111000010010111111000101101100011010101010100 0101...
P0 862345 1332387: 010001100001111000010010111111000101101100011010101010100 0101...
S0 862345 1332387: 000000000000000000000000000000000000000000000000000000000000...
C0 862346 1332420: 110100001010010101100110001101001010100011001011100010010 11...
P0 862346 1332420: 110100001010010101100110001101001010100011001011100010010 11...
S0 862346 1332420: 000000000000000000000000000000000000000000000000000000000000...
C0 862347 1332453: 111110000111111010100110101000000100101100001000011111011 10...
P0 862347 1332453: 111110000111111010100110101000000100101100001000011111011 10...
S0 862347 1332453: 000000000000000000000000000000000000000000000000000000000000...
error: sacch: parity error (-1 fn=862347)
cannot decode fnr=0x0d288b (862347) ts=1
```

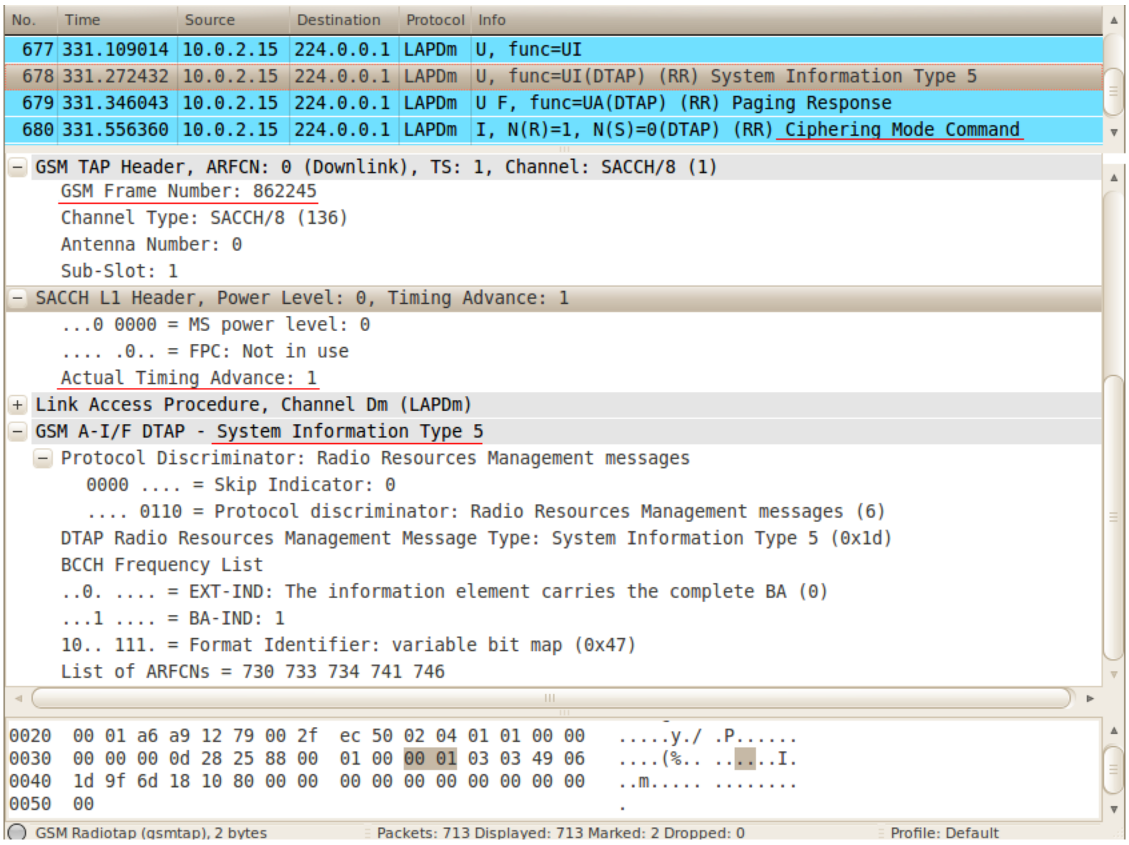
   "Cx" are the encrypted burst bits, "Px" are the decrypted burst bits and "Sx" are the keystream bits (encrypted bits XOR decrypted bits). We do not decrypt right now so the decrypted burst bits are the same as the encrypted burst bits. If x is 1 then this is the first burst of a frame.

   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.

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

Task 2 - Finding the key $\mathbf{K_c}$
In order to find the K_c we are using the Kraken tool. This tool uses a 2 TB rainbowtable 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 clear text. Now we only need the framecount where the same message is transmitted after activation of the encryption.



```
No.  Time        Source     Destination  Protocol  Info
677  331.109014  10.0.2.15  224.0.0.1    LAPDm     U, func=UI
678  331.272432  10.0.2.15  224.0.0.1    LAPDm     U, func=UI(DTAP) (RR) System Information Type 5
679  331.346043  10.0.2.15  224.0.0.1    LAPDm     U F, func=UA(DTAP) (RR) Paging Response
680  331.556360  10.0.2.15  224.0.0.1    LAPDm     I, N(R)=1, N(S)=0(DTAP) (RR) Ciphering Mode Command
```

```
− GSM TAP Header, ARFCN: 0 (Downlink), TS: 1, Channel: SACCH/8 (1)
    GSM Frame Number: 862245
    Channel Type: SACCH/8 (136)
    Antenna Number: 0
    Sub-Slot: 1
− SACCH L1 Header, Power Level: 0, Timing Advance: 1
    ...0 0000 = MS power level: 0
    .... .0.. = FPC: Not in use
    Actual Timing Advance: 1
+ Link Access Procedure, Channel Dm (LAPDm)
− GSM A-I/F DTAP - System Information Type 5
  − Protocol Discriminator: Radio Resources Management messages
      0000 .... = Skip Indicator: 0
      .... 0110 = Protocol discriminator: Radio Resources Management messages (6)
    DTAP Radio Resources Management Message Type: System Information Type 5 (0x1d)
    BCCH Frequency List
    ..0. .... = EXT-IND: The information element carries the complete BA (0)
    ...1 .... = BA-IND: 1
    10.. 111. = Format Identifier: variable bit map (0x47)
    List of ARFCNs = 730 733 734 741 746
```

```
0020  00 01 a6 a9 12 79 00 2f  ec 50 02 04 01 01 00 00   .....y./ .P......
0030  00 00 00 0d 28 25 88 00  01 00 00 01 03 03 49 06   ....(%.. ......I.
0040  1d 9f 6d 18 10 80 00 00  00 00 00 00 00 00 00 00   ..m..... ........
0050  00                                                  .
```

GSM Radiotap (gsmtap), 2 bytes   Packets: 713 Displayed: 713 Marked: 2 Dropped: 0   Profile: Default

2. 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.

   a) To do so you need to find the unencrypted System Information Type 5 message which is displayed in wireshark (Hint: search for the GSM Frame Number). Within 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; example: task 1 step 7). To make it easier for you, you can copy the block into a textfile. You will need the four lines that start with "Cx".

   b) 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 gsmframecoder[1]. 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 bitstreams, denoted as Burst1 to Burst4, which are needed for the next step.

   c) 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

[1] gsmframecoder: `http://www.ks.uni-freiburg.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. Use telnet to connect to the 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!):
   ```
   crack 0110101000111011111110100111001101000011001000000001001110011000101001 1...
   ```

   Note: You need to repeat this step with the next xor'ed "BurstX" / "Cx" combination until you find the correct position on the rainbowtable.
   A successful output looks like this: `Found d5eb21665d2b8f25 @ 13`. This means ad5eb21665d2b8f25 is the key that produces the output at postion 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 rainbowtable 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`.

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

Task 3 - decoding the traffic channel

1. Now you are able to decode the encrypted part of the assigned "SDCCH/8" channel. Pass the key to the decoder like this:
   ```
   ./go_usrp2.sh vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile 174 1S 1EF00BAB3BAC7002
   ```

2. Again you can see the decoded packets in wireshark. Look for the "Assignment Command" and find the timeslot assigned for the TCH.

3. Decode and decrypt the TCH on timeslot 5 with the following command:
   ```
   ./go_usrp2.sh vf_call6_a725_d174_g5_Kc1EF00BAB3BAC7002.cfile 174 5T 1EF00BAB3BAC7002
   ```

4. Now you have a file named *speech.au.gsm* which contains the audio stream of the traffic channel.

5. Convert the *speech.au.gsm* audio file with toast[2]:
   ```
   toast -d speech.au.gsm
   ```

[2]toast: `http://www.quut.com/gsm/`

6. Listen to the audio file with the totem-player:
   ```
   totem speech.au
   ```