GSM Network

Carte à puce et Java Card
ATAC 2011-2012

Jean-Louis Lanet
Jean-louis.lanet@unilim.fr
Outline

• GSM Security consideration
• GSM Architecture
• GSM Security Function
  – Authentication
  – Encryption
• GSM Standard GSM 11.11
  – Basic definition
  – Security features
  – Commands
SIM standardization group 1998

- Billions of Calls,
- Millions of Subscribers,
- Thousands of different types of telephones,
- Hundreds of countries,
- Dozens of Manufacturers,

...and only one Card... the SIM
GSM Security Concerns

• Operators
  – Bills right people
  – Avoid fraud
  – Protect Services

• Customers
  – Privacy
  – Anonymity

• Make a system at least secure as PSTN
GSM Security Goals

• Confidentiality and Anonymity on the radio path
• Strong client authentication to protect the operator against the billing fraud
• Prevention of operators from compromising of each others’ security
  – Inadvertently
  – Competition pressure
GSM Security Design Requirements

• The security mechanism
  – MUST NOT
    • Add significant overhead on call set up
    • Increase bandwidth of the channel
    • Increase error rate
    • Add expensive complexity to the system
  – MUST
    • Cost effective scheme
  – Define security procedures
    • Generation and distribution of keys
    • Exchange information between operators
    • Confidentiality of algorithms
GSM Security Features

• **Key management is independent of equipment**
  – Subscribers can change handsets without compromising security

• **Subscriber identity protection**
  – Not easy to identify the user of the system intercepting a user data

• **Detection of compromised equipment**
  – Detection mechanism whether a mobile device was compromised or not

• **Subscriber authentication**
  – The operator knows for billing purposes who is using the system

• **Signaling and user data protection**
  – Signaling and data channels are protected over the radio path
GSM Mobile Station

Mobile Station
  – Mobile Equipment (ME)
    • Physical mobile device
    • Identifiers
      – IMEI – International Mobile Equipment Identity
  – Subscriber Identity Module (SIM)
    • Smart Card containing keys, identifiers and algorithms
    • Identifiers
      – $K_i$ – Subscriber Authentication Key
      – IMSI – International Mobile Subscriber Identity
      – TMSI – Temporary Mobile Subscriber Identity
      – MSISDN – Mobile Station International Service Digital Network
      – PIN – Personal Identity Number protecting a SIM
      – LAI – location area identity
Subscriber Identity Protection

- **TMSI** – Temporary Mobile Subscriber Identity
  - **Goals**
    - TMSI is used instead of IMSI as a temporary subscriber identifier
    - TMSI prevents an eavesdropper from identifying of subscriber
  - **Usage**
    - TMSI is assigned when IMSI is transmitted to AuC on the first phone switch on
    - Every time a location update (new MSC) occurs the networks assigns a new TMSI
    - TMSI is used by the MS to report to the network or during a call initialization
    - Network uses TMSI to communicate with MS
    - On MS switch off TMSI is stored on SIM card to be reused next time
  - The Visitor Location Register (VLR) performs assignment, administration and update of the TMSI
TIMSI-IMSI

MSC/VLR

Location Updating Request (TMSIold)

TMSI Reallocation Command (TMSInew)

Stored in the dedicated EF

TMSI Reallocation Complete

Assignment of TMSInew

De allocation of TMSIold
Key Management Scheme

• **$K_i$** – Subscriber Authentication Key
  – Shared 128 bit key used for authentication of subscriber by the operator
  – Key Storage
    • Subscriber’s SIM (owned by operator, i.e. trusted)
    • Operator’s Home Locator Register (HLR) of the subscriber’s home network

• SIM can be used with different equipment
Detection of Compromised Equipment

- **International Mobile Equipment Identifier (IMEI)**
  - Identifier allowing to identify mobiles
  - IMEI is independent of SIM
  - Used to identify stolen or compromised equipment

- **Equipment Identity Register (EIR)**
  - Black list – stolen or non-type mobiles
  - White list - valid mobiles
  - Gray list – local tracking mobiles

- **Central Equipment Identity Register (CEIR)**
  - Approved mobile type (type approval authorities)
  - Consolidated black list (posted by operators)
Authentication

• Authentication Goals
  – Subscriber (SIM holder) authentication
  – Protection of the network against unauthorized use
  – Create a session key

• Authentication Scheme
  – Subscriber identification: IMSI or TMSI
  – Challenge-Response authentication of the subscriber by the operator
  – Unilateral Authentication
    • Counterfeit network,
    • Eavesdrop on call using a suitable piece of equipment!
Authentication and Encryption Scheme

**Mobile Station**
- SIM
- $K_i$
- $A3$
- $A8$
- $A5$
- $m_i$
- $F_n$

**Radio Link**
- Challenge RAND
- Signed response (SRES)
- $K_c$
- $m_i$
- Encrypted Data

**GSM Operator**
- $K_i$
- $A3$
- $A8$
- $A5$
- $m_i$

Authentication: are SRES values equal?
Authentication

• **AuC – Authentication Center**
  – Provides parameters for authentication and encryption functions (RAND, SRES, $K_c$)

• **HLR – Home Location Register**
  – Provides MSC (Mobile Switching Center) with triples (RAND, SRES, $K_c$)
  – Handles MS location

• **VLR – Visitor Location Register**
  – Stores generated triples by the HLR when a subscriber is not in his home network
  – One operator doesn’t have access to subscriber keys of the another operator.
Triplet Generation

Map_Send_Authentication_info (IMSI)

Locally stored

BS/MSC/VLR

Map_Send_Authentication_info (SRES, RAND, Kc)_1^n

Generate from 1 to n RAND

HLR/AuC

Ki
Triple Generation

Ki never moves outside the SIM or the AuC

Locally stored

Map_Send_Authentication_info
(SRES, RAND, Kc)\textsuperscript{n}

In case of roaming the triple is sent to the foreign network not the Key

BS/MSC/VLR

HLR/AuC
A3 – MS Authentication Algorithm

- **Goal**
  - Generation of SRES response to MSC’s random challenge RAND

![Diagram showing the A3 algorithm process]

RAND (128 bit) → A3 → SRES (32 bit)
A8 – Voice Privacy Key Generation Algorithm

• Goal
  – Generation of session key $K_c$
  • A8 specification was never made public

```
RAND (128 bit) → A8

K_i (128 bit) → A8

K_C (64 bit)
```
Logical Implementation of A3 and A8

- Both A3 and A8 algorithms are implemented on the SIM
  - Operator can decide, which algorithm to use.
  - Algorithms implementation is independent of hardware manufacturers and network operators.
Logical Implementation of A3 and A8

- COMP128 is used for both A3 and A8 in most GSM networks.
  - COMP128 is a keyed hash function

```
<table>
<thead>
<tr>
<th>RAND (128 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K_i (128 bit)</td>
</tr>
<tr>
<td>COMP128</td>
</tr>
<tr>
<td>128 bit output</td>
</tr>
<tr>
<td>SRES 32 bit and Kc 64 bit</td>
</tr>
</tbody>
</table>
```
A5 – Encryption Algorithm

- A5 is a stream cipher
  - Implemented very efficiently on hardware
  - Design was never made public
  - Leaked to Ross Anderson and Bruce Schneier

- Variants
  - A5/1 – the strong version
  - A5/2 – the weak version
  - A5/3
    - GSM Association Security Group and 3GPP design
    - Based on Kasumi algorithm used in 3G mobile systems
Logical A5 Implementation

Mobile Station

$F_n$ (22 bit) \rightarrow A5 \rightarrow 114$ bit \rightarrow XOR \rightarrow Ciphertext (114 bit)

BTS

$F_n$ (22 bit) \rightarrow A5 \rightarrow 114$ bit \rightarrow XOR \rightarrow Data (114 bit)

Real A5 output is 228 bits for both directions
A5 Encryption

Base Station Subsystem

Network Management

Subscriber and terminal equipment databases

OMC

Exchange System

VLR

HLR

AUC

EIR

BTS

BSC

MSC

A5 Encryption
Part II

Accessing the Sim application, the file system
For coherent Communication between

SIM <-> MOBILE
SIM <-> SUBSCRIBER
SIM <-> NETWORK (OTA)
Typical SIM card organization

During personalization IMSI and Ki and unblocking key are provided by the operator.

GSM operator directory
Telecom user directory

MF

Level 0

DF
GSM

DF
Telecom

EF

EF

EF

EF
The files

- Root directory : 3F 00
- Two main directories : GSM (DFGSM, 7F20) and TELECOM (DFTELECOM, 7F10).
- The identity is coded on two bytes, the first :
  - '3F': Master File;
  - '7F': 1st level Dedicated File;
  - '5F': 2nd level Dedicated File;
  - '2F': Elementary File under the Master File;
  - '6F': Elementary File under a 1st level Dedicated File;
  - '4F': Elementary File under 2nd level Dedicated File.
- After ATR (Answer To Reset), the master file (MF) is implicitly selected.
GSM directory

• The file $\text{EF}_{\text{IMSI}}$ (6F07) includes the IMSI.
• The file $\text{EF}_{\text{LOCI}}$ (6F7E) includes the parameters: TMSI, LAI.
• $\text{EF}_{\text{LP}}$ (Language preference)
• $\text{EF}_{\text{Kc}}$ (Ciphering key Kc) includes the Kc and the sequence number of the key.
• $\text{EF}_{\text{SST}}$ (SIM service table) lists the available services in the SIM.
  – Service n°1: CHV1 disable function
  – Service n°2: Abbreviated Dialling Numbers (ADN)
  – Service n°3: Fixed Dialling Numbers (FDN)
  – Service n°4: Short Message Storage (SMS)
  – etc.
• $\text{EF}_{\text{ACM}}$ (Accumulated call meter) is the total number of unit used for the current call and all the previous.
• $\text{EF}_{\text{MSISDN}}$ (MSISDN) includes the phone number of the subscriber MSISDN.
Telcom Directory

- $\text{EF}_{\text{ADN}}$ (6F3A) include the short diary,
- $\text{EF}_{\text{FDN}}$ (6F3B) the contact list,
- $\text{EF}_{\text{SMS}}$ (6F3C) the received and sent SMS,
- etc.
- These files are accessible in read and write mode and are protected with the Pin code.
File selection

- Only one file is selected at a time,
- The MF is always selectable and is implicitly selected after a reset
- FID are not unique => restriction in selection
EF File structures

• Four data structure
  – Binary (transparent) files (data accessible through an address)
  – Sequential record fixed size or variable size
  – Cyclic buffer

• Transparent file
  – No internal structure
  – Accessed for reading or writing in bytes or blocks with an offset value
  – Often used with a small amount of data,
  – Commands READ BINARY, WRITE BINARY and UPDATE BINARY
EF File structures

• Linear fixed file structure
  – Linking fixed length records,
  – The smallest unit is a record,
  – Commands: READ RECORD, WRITE RECORD and UPDATE RECORD, e.g. phone book
  – From 1..254

• Linear variable file structure
  – Same commands,
  – Need additional info concerning the length of each records,
  – Optimise the memory usage e.g. the phone book…
EF File structures

• Cyclic file structure
  – Based on the linear fixed file structure,
  – The EF contains a pointer on the last written record numbered 1, the previous 2, etc…
  – Can be accessed by addressing the first, the last, the next or previous record.
File Access Conditions

- Security is based on file access privileges,
- Access information coded in the header, defined when a file is created and usually cannot be changed later.
- For MF and DF
  - no information stored for data access (read and write)
  - But for creation and deletion of files.
- The PINs are stored in separate elementary files, $EF_{CHV1}$ and $EF_{CHV2}$ for example
File attributes

• Five kinds of EF files
  • **Always (ALW):** Access of the file can be performed without any restriction.
    – **Card holder verification 1 (CHV1):** Access can only be possible when a valid CHV1 value is presented
    – **Card holder verification 2 (CHV2):** Access can only be possible when a valid CHV2 value is presented
    – **Administrative (ADM):** Allocation of these levels and the respective requirements for their fulfilment are the responsibility of the appropriate administrative authority
    – **Never (NEV):** Access of the file is forbidden
The APDU commands
GSM commands CLA = A0

• Data access commands
  – Select (header), select a EF or MF, with a getResponse the status of the file
  – Status, sent by the terminal to a proactive SIM
  – ReadBinary, UpdateBinary read or update the current file
  – Seek, next record in the current file
  – Increase, add a record in a cyclic file

• Security related commands
  – Verify, Change, Disable, Enable, Unblock a CHV
  – Invalidate, Rehabilitate a file
  – Run GSM Algorithm run the A3 algorithm
The SELECT command

• A0 A4 00 00 02 XX XX (XX XX : FID of the EF of DF to be selected).

• The response of the selection request shall include:
  – size of the unused memory,
  – name of the DF file
  – kind of DF (MF or not)
  – PIN code request
  – number of included DF
Example

• How to read the IMSI?
  – File is $EF_{\text{IMSI}} (6F07)$ of the GSM directory,
  – Any idea?
Example

• How to read the IMSI?
  – File is $\text{EF}_{\text{IMSI}}$ (6F07) of the GSM directory,
  – Select the Master File (3F00)
  – Select $\text{DF}_{\text{GSM}}$ (7F20)
  – Select $\text{EF}_{\text{IMSI}}$ (6F07)
  – Read 9 bytes with READ BINARY
Example

• How to read the IMSI?
  – File is $\text{EF}_{\text{IMSI}}$ (6F07) of the GSM directory,
  – Select the Master File (3F00)

$$\Rightarrow \text{A0 A4 0000 02 3F00}$$

$$\Rightarrow 9F22$$

  – Select $\text{DF}_{\text{GSM}}$ (7F20)
  – Select $\text{EF}_{\text{IMSI}}$ (6F07)
  – Read 9 bytes with READ BINARY
Example

• How to read the IMSI?
  – File is \( \text{EF}_{\text{IMSI}} \) (6F07) of the GSM directory,
  – Select the Master File (3F00)
  – Select \( \text{DF}_{\text{GSM}} \) (7F20)

\[ \Rightarrow \text{A0 A4 0000 02 7F20} \]

\[ \Leftarrow 9F22 \]

– Select \( \text{EF}_{\text{IMSI}} \) (6F07)
– Read 9 bytes with READ BINARY
Example

• How to read the IMSI?
  – File is $\text{EF}_{\text{IMSI}}$ (6F07) of the GSM directory,
  – Select the Master File (3F00)
  – Select $\text{DF}_{\text{GSM}}$ (7F20)
  – Select $\text{EF}_{\text{IMSI}}$ (6F07)

  $\implies$ A0 A4 0000 02 6F07

  $\leqslant$ 9F0F

  – Read 9 bytes with READ BINARY
Example

• How to read the IMSI?
  – File is $\text{EF}_{\text{IMSI}}$ (6F07) of the GSM directory,
  – Select the Master File (3F00)
  – Select $\text{DF}_{\text{GSM}}$ (7F20)
  – Select $\text{EF}_{\text{IMSI}}$ (6F07)
    \[ \Leftarrow 9F0F \]
  – $9FXX$ which mean success with $XX$ bytes of response data, you can pull
    the response with GET RESPONSE command ‘C0’,

\[ \Rightarrow \text{A0 C0 0000 0F} \]

\[ \Leftarrow 00 00 00 09 6F 07 04 00 15 F5 15 01 02 00 00 9000 \]
Example

• How to read the IMSI?

  – GET RESPONSE command ‘C0’,

  0000 0009 6F07 04 00 15F515 01 02 0000 9000
  | | | | | | | | | | status
  | | | | | | | | | | structure 00 = transparent
  | | | | | | | | | | length of data following
  | | | | | | | | | | status
  | | | | | | | | | | access READ|UPDATE INCREASE|
  | | | | | | | | | | RFU|REHABILITATE|INVALIDATE
  | | | | | | | | | | file type 04 = EF
  | | | | | | | | | | file id
  | | | | | | | | | | size
  | | | | | | | | | | RFU
Example

• How to read the IMSI?
  – File is $EF_{\text{IMSI}}$ (6F07) of the GSM directory,
  – Select the Master File (3F00)
  – Select $DF_{\text{GSM}}$ (7F20)
  – Select $EF_{\text{IMSI}}$ (6F07)
  – Read 9 bytes with READ BINARY

$\Rightarrow \text{A0 B0 0000 09}$

$\Leftarrow 08 \text{ 29 80 02 12 34 54 90 03 9000}$

$EF_{\text{IMSI}}$ (IMSI)
  Byte 1 length of IMSI
  Byte 2-9 IMSI 8 bytes
Access Right

- When you’re granted to CHV1 you can read its value but neither change it nor deactivate it.
  - Access Conditions:
    - READ CHV1
    - UPDATE ADM
    - INVALIDATE ADM
    - REHABILITATE CHV1
  - Access rights are coded as:
    - READ|UPDATE
    - INCREASE|RFU
    - REHABILITATE|INVALIDATE
    - knowing that '0' means always, '1' CHV1, 'F' never and '4'... 'E' ADM.
    - 15 F5 15
PIN code commands

• PIN code is coded on 8 bytes. The non significant bytes are coded with FF.
  – my sim pin code is 0973, must be coded as 30 39 37 33 FF FF FF FF
• VERIFY CHV : verify the Pin
  – A0 20 00 P2 08 PIN (P2=01 for CHV1 (user PIN code), = 02 for CHV2).
• DISABLE PIN disable PIN usage.
  – A0 26 00 01 08 PIN
• ENABLE PIN enable PIN usage
  – A0 28 00 01 08 PIN
• CHANGE CHV modify the value of the PIN code
  – A0 24 00 01 10 previous_PIN new_PIN
• UNBLOCK CHV unblock a card that has its PIN code blocked (CHV1).
  – A0 2C 00 01 10 PUK PIN.
Read a GSM File

Select (DF\textsubscript{GSM})
\[ \text{SW1}=9F, \text{SW2}=xx \]

GetResponse

[Nb EF files, access condition], SW1=90, SW2=00

Select (EF\textsubscript{LOCI})
\[ \text{SW1}=9F, \text{SW2}=xx \]

GetResponse

[Type of EF file, access condition], SW1=90, SW2=00

ReadBinary
\[ \text{[data]}, \text{SW1}=90, \text{SW2}=00 \]

P1 & P2 provide the offset and the number of data to read in the file
Part III

The SIM Toolkit framework
SIM Application Toolkit (SAT)

- Specified by the standard 3GPP TS 11.14,
- Additional framework that allows the SIM to interact with the mobile
- Identified with the content of the EF_{SST}
- Event programming application
- SMS used as an administrative mean (3GPP byte code interpreter) for RPC by the network admin or third tiers applications.
Architecture of the Java-SIM

GSM Applet

Toolkit Applets

Applets

Loader Applet

SIM Toolkit Framework

Toolkit Registry

Toolkit Handler

File System

JCRE

shareable interface
SIM Toolkit apps

• The SAT applications:
  – can initiate actions (pro-active commands),
  – can be externally triggered with events,
  – can get the characteristics of the mobile (a mean for the ME to tell the card what is able to do)
  – Four new APDU commands are defined to manage SIM Toolkit features
    • Fetch,
    • Terminal Response
    • Envelope
    • Terminal Profile
Pro active mode

• Only if the terminal supports this mode,
  – Get the terminal profile during initialization,
  – 20 bytes are sent back by the terminal
    • each bit codes facilities (TRUE= supported)
    • e.g. second byte, bit 8 Display Text is supported

• Command with the ME display:
  – Display Text, Set up menu, Send DTMF, Play Tone, Language Notification,…

• Commands with the keyboard/display
  – Get Inkey, get Input,…

• With the Radio equipment of the terminal
  – Set up, Send SMS, Send Sup. Services, Provide Local information,…
  – Launch browser, Perform Card APDU
Pro Active Commands

Setup Menu

- NETWORK
- BANKING
- NEWS
- WEATHER

Display Text

- The weather today is going to be fine.
- ok

Get Input

- Please enter name:
- ok

Select Item

- BANKING
- BALANCE
- PURCHASE
- TRANSFER

Send SMS

- SMS in progress.
- Please Wait...

Setup Call

- CALLING
- 01 4746 6667
- Please Wait...
To send a command

- Wait a poll request by the ME : command Status,
  - each second the ME polls the card (can be modified with a command),
- Wait for a regular command (e.g : Select File,..)
  - Answer with a 91xx status word,
  - The handset sends the SIM a FETCH command with an expended data length of xx size,
  - The handset parse it and execute the proactive command,
  - Send a response to the Card : Terminal Response which depends on the pro active command.
Pro active command

Select ($DF_{GSM}$)

SW1=91, SW2=xx

FETCH

[Display text], SW1=90, SW2=00

Terminal Response

SW1=90, SW2=00, or command request
Pro active command

Status
SW1=90, SW2=00

... Status
SW1=91, SW2=xx

Fetch
[Display text], SW1=90, SW2=00

Terminal Response
SW1=90, SW2=00, or command request
Structure of proactive command

- SIM gives the handset a sequence of Tag length Value (TLV),
  - The tag is always 1 byte and the length 1 byte (00..7F) or 2 bytes (81 + 80..FF),
  - Tags are well defined and can depend on the context,
    - Tag ME to SIM : 0xD1, 0xD2, 0xD3 & 0xD4
    - SIM to ME : 0xD0
  - A TLV can include other TLV and becomes a compound TLV,
  - Example :
    - Play music on our phone,
    - Expect a command or a status,
    - Send a request 91 23,
    - Wait for a Fetch,
    - Send the correct PLAY TONE command…
Structure of proactive command

• SIM gives the handset a sequence of Tag length Value (TLV),
  – The tag is always 1 byte and the length 1 byte (00..7F) or 2 bytes (81 + 80..FF),
  – Tags are well defined and can depend on the context,
  – A TLV can include other TLV and becomes a compound TLV,
  – Example:
    • Play music on our phone,
    • Expect a command or a status,
    • Send a request 91 23,
    • Wait for a Fetch,
    • Send the correct PLAY TONE command…

Start with ?
Play Tone

- The SIM toolkit application can request the application to play a short tone
  - Busy, CallWaiting, Congestion, Dial, Dropped, Error, GeneralBeep, NegativeBeep, PositiveBeep, RadioAck, Ringing

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0xD0</td>
<td>Tag Proactive command</td>
</tr>
<tr>
<td>2</td>
<td>0x15</td>
<td>Length 21 bytes</td>
</tr>
<tr>
<td>3</td>
<td>0x01</td>
<td>Tag : command detail</td>
</tr>
<tr>
<td>4</td>
<td>0x03</td>
<td>Length</td>
</tr>
<tr>
<td>5</td>
<td>0x01</td>
<td>Identifier</td>
</tr>
<tr>
<td>6</td>
<td>0x20</td>
<td>Play Tone</td>
</tr>
<tr>
<td>7</td>
<td>0x00</td>
<td>Qualifier</td>
</tr>
<tr>
<td>8</td>
<td>0x02</td>
<td>Device identity</td>
</tr>
<tr>
<td>9</td>
<td>0x02</td>
<td>Length</td>
</tr>
<tr>
<td>10</td>
<td>0x81</td>
<td>Source (uicc)</td>
</tr>
<tr>
<td>11</td>
<td>0x82</td>
<td>Destination (me)</td>
</tr>
<tr>
<td>12</td>
<td>0x05</td>
<td>Alpha identifier</td>
</tr>
<tr>
<td>13</td>
<td>0x03</td>
<td>Length</td>
</tr>
<tr>
<td>14</td>
<td>0x42</td>
<td>Ascii value ‘S’</td>
</tr>
<tr>
<td>15</td>
<td>0x4F</td>
<td>Ascii value ‘O’</td>
</tr>
<tr>
<td>16</td>
<td>0x4F</td>
<td>Ascii value ‘O’</td>
</tr>
<tr>
<td>17</td>
<td>0x0E</td>
<td>Tone Value</td>
</tr>
<tr>
<td>18</td>
<td>0x01</td>
<td>Length</td>
</tr>
<tr>
<td>19</td>
<td>0x01</td>
<td>Play the dial tone</td>
</tr>
<tr>
<td>20</td>
<td>0x04</td>
<td>Duration</td>
</tr>
<tr>
<td>21</td>
<td>0x02</td>
<td>Length</td>
</tr>
<tr>
<td>22</td>
<td>0x01</td>
<td>Unit in second</td>
</tr>
<tr>
<td>23</td>
<td>0x05</td>
<td>Number of unit</td>
</tr>
</tbody>
</table>
ME response

- After the handset blast the dial tone (#1) for 5 seconds, it sends a status response,
  - Terminal response APDU, 0x80 0x14 0x00 0x00 0x00 0x0C data
  - The byte 12 is equivalent to a 90 00 from the ME

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x01</td>
<td>Command detail</td>
</tr>
<tr>
<td>2</td>
<td>0x03</td>
<td>Length</td>
</tr>
<tr>
<td>3</td>
<td>0x01</td>
<td>Identifier</td>
</tr>
<tr>
<td>4</td>
<td>0x20</td>
<td>Play Tone</td>
</tr>
<tr>
<td>5</td>
<td>0x00</td>
<td>Qualifier</td>
</tr>
<tr>
<td>6</td>
<td>0x02</td>
<td>Device identity</td>
</tr>
<tr>
<td>7</td>
<td>0x02</td>
<td>Length</td>
</tr>
<tr>
<td>8</td>
<td>0x82</td>
<td>Source (me)</td>
</tr>
<tr>
<td>9</td>
<td>0x81</td>
<td>Destination (uicc)</td>
</tr>
<tr>
<td>10</td>
<td>0x03</td>
<td>Result</td>
</tr>
<tr>
<td>11</td>
<td>0x01</td>
<td>Length</td>
</tr>
<tr>
<td>12</td>
<td>0x00</td>
<td>Success</td>
</tr>
</tbody>
</table>
Event command

- The SIM can register for events
  - Can use the proactive command Setup Event list,
  - SMS-PP, Menu Selection, MT Call, Location Status, Browser Termination,…
  - The handset uses the Envelope APDU to send a description of the event to the card,
  - Structured with TLV as proactive commands,
  - Events can be routed by the handset from a service provider server.
Java Card SIM Applet

• The difference between Java Card Applet and Toolkit the latter does not handle APDU directly
• The API provides two packages:
  – The `sim.access` package, which allows applets to access the GSM files
  – The `sim.toolkit` provides methods to register to events, generate pro active commands,
  – The interface `ToolkitConstants`, encapsulates constants related to the Toolkit applets.
  – The `ToolkitInterface` must be implemented by a toolkit applet so that it can be triggered by the toolkit handler according to the registration information.

```java
import sim.toolkit.*;
import sim.access.*;

public class MyToolkitApplet extends Applet implements ToolkitInterface, ToolkitConstants;
```
### sim.toolkit Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EditHandler</td>
<td>This class is the basic class for the construction of a list of simple TLV elements.</td>
</tr>
<tr>
<td>EnvelopeHandler</td>
<td>The EnvelopeHandler class contains basic methods to handle the Envelope data field.</td>
</tr>
<tr>
<td>EnvelopeResponseHandler</td>
<td>The EnvelopeResponseHandler class contains basic methods to handle the Envelope response data field.</td>
</tr>
<tr>
<td>ProactiveHandler</td>
<td>This class is the basic class for the definition of Proactive commands.</td>
</tr>
<tr>
<td>ProactiveResponseHandler</td>
<td>The ProactiveResponseHandler class contains basic methods to handle the Terminal Response data field.</td>
</tr>
<tr>
<td>ViewHandler</td>
<td>The ViewHandler class offers basic services and contains basic methods to handle TLV list.</td>
</tr>
<tr>
<td>ToolkitRegistry</td>
<td>The Registry class offers basic services and methods to allow any Toolkit applet to register its configuration during the install phase.</td>
</tr>
<tr>
<td>MEProfile</td>
<td>The MEProfile class contains methods to question the handset profile.</td>
</tr>
</tbody>
</table>
# sim.toolkit Exceptions

| ToolkitException | This exception extends the Throwable class and allows the classes of this package to throw specific exceptions in case of problems. |
public class MyToolkitApplet extends Applet implements ToolkitInterface, ToolkitConstants {
    public MyToolkitApplet() {
        reg = ToolkitRegistry.getEntry();
        menuId = reg.initMenuEntry(menuEntry, (short)0, (short)menuEntry.length, PRO_CMD_SET_UP_CALL, false, 0, 0);
        reg.disableMenuEntry(menuId);
        reg.setEvent(EVENT_FORMATTED_SMS_PP_ENV);
    }

    public static void install(byte bArray[], short bOffset, byte bLength) throws ISOException {
        MyToolkitApplet applet = new MyToolkitApplet();
        applet.register();
    }
}
public class MyToolkitApplet extends Applet implements ToolkitInterface, ToolkitConstants {
  public MyToolkitApplet() {
    reg = ToolkitRegistry.getEntry();
    menuId = reg.initMenuEntry(menuEntry, (short)0,
                               (short)menuEntry.length, PRO_CMD_SET_UP_CALL, false, 0, 0);
    reg.disableMenuEntry(menuId);
    reg.setEvent(EVENT_FORMATTED_SMS_PP_ENV);
  }

  public static void install(byte bArray[], short bOffset, byte bLength) throws ISOException {
    MyToolkitApplet applet = new MyToolkitApplet();
    applet.register();
  }

  The applet can be triggered by both selection mechanisms.
}
public class MyToolkitApplet extends Applet implements ToolkitInterface, ToolkitConstants {
    public MyToolkitApplet() {
        reg = ToolkitRegistry.getEntry();
        menuId = reg.initMenuEntry(menuEntry, (short)0, (short)menuEntry.length, PRO_CMD_SET_UP_CALL, false, 0, 0);
        reg.disableMenuEntry(menuId);
        reg.setEvent(EVENT_FORMATTED_SMS_PP_ENV);
    }

    public static void install(byte bArray[], short bOffset, byte bLength) throws ISOException {
        MyToolkitApplet applet = new MyToolkitApplet();
        applet.register();
    }
}
How to handle pro active commands?

- The SIM application toolkit protocol (\textit{i.e.} 91xx, Fetch, Terminal Response) is handled by the GSM applet and the Toolkit Handler, the toolkit applet shall not handle those events.

- The SIM Toolkit Framework shall provide a reference of the \texttt{sim.toolkit.ProactiveHandler} to the toolkit applet so that when the toolkit applet is triggered it can:
  - initialise the current proactive command with the \texttt{init()} method;
  - append several Simple TLV to the current proactive command with the \texttt{appendTLV()} methods;
  - ask the SIM Toolkit Framework to send this proactive command to the ME and wait for the reply, with the \texttt{send()} method.
private static final byte MY_COMMAND = PRO_CMD_DISPLAY_TEXT;
private static final byte MY_TAG = TAG_TEXT_STRING;
private byte[] text = new byte[12];
text[0] = (byte)'L';
text[1] = (byte)'I';
text[2] = (byte)'M';
ProactiveHandler proHdlr;
proHdlr = ProactiveHandler.getTheHandler();
  proHdlr.init(MY_COMMAND, (byte)0, DEV_ID_ME);
proHdlr.appendTLV((byte)(MY_TAG | TAG_SET_CR), DCS_8_BIT_DATA, text, (short)0, (short)3);
result = proHdlr.send();
private byte[] data;
data = new byte[32]; // build a buffer
ProactiveResponseHandler ProRespHdlr;
ProRespHdlr = ProactiveResponseHandler.getTheHandler();
byte result = ProRespHdlr.getGeneralResult();
respHdlr.findTLV(TAG_DEVICE_IDENTITIES, 1);
byte sourceDev = ProRespHdlr.getValueByte((short)0);
byte destinDev = ProRespHdlr.getValueByte((short)1);
if (ProRespHdlr.findTLV(TAG_TEXT_STRING, (byte)1) ==
TLV_FOUND_CR_SET) {
    if (((short) len = ProRespHdlr.getValueLength()) > 1) {
        ProRespHdlr.copyValue((short)1, data, (short)0,
        (short)(len - 1));
    }
}
private static final byte MY_TAG = (byte)0x54;
private byte[] data;
data = new byte[32];
void processToolkit (byte event) throws ToolkitException {
    // get the EnvelopeHandler system instance
    EnvelopeHandler theEnv = EnvelopeHandler.getTheHandler();
    // look for MY_TAG TLV
    if (theEnv.findTLV(MY_TAG, (byte)1) != TLV_NOT_FOUND) {
        // check first element byte
        if (theEnv.getValueByte((short)0) == (byte)1) {
            // copy element part into data buffer
            theEnv.copyValue((short)1, data,(short)0,
                                (short)(theEnv.getValueLength() - 1));
        }
    }
}
The complete example: Hello World

import javacard.framework.*;
import sim.toolkit.*;

public class HelloWorld extends Applet implements
  ToolkitConstants, ToolkitInterface {
  private final byte COMMAND_QUALIFIER = (byte)0x80;
  private final byte[] MENU_ENTRY =
    {'C','r','y','p','t','i','s'};
  private final byte[] HELLO_WORLD =
    {'H','e','l','l','o',' ','w','o','r','l','d',' ','!'};
  private ToolkitRegistry registry;
  public HelloWorld() {
    registry = ToolkitRegistry.getEntry();
    registry.initMenuEntry(menuEntry, (short)0,(short)
      MENU_ENTRY.length, PRO_CMD_DISPLAY_TEXT, false, 0, 0);
  }
public static void install(byte bArray[], short bOffset, byte bLength) throws ISOException {
    HelloWorld applet = new HelloWorld();
    applet.register();
}

public void processToolkit (byte event) throws ToolkitException {
    ProactiveHandler proHdlr = ProactiveHandler.getTheHandler();
    if (event == EVENT_MENU_SELECTION) {
        proHdlr.init((byte) PRO_CMD_DISPLAY_TEXT, (byte) COMMAND_QUALIFIER, DEV_ID_ME);
        proHdlr.appendTLV((byte)(TAG_TEXT_STRING), HELLO_WORLD, (short) 0, (short) HELLO_WORLD.length);
        proHdlr.send();
    }
}
**sim.access Interfaces & Classe**

<table>
<thead>
<tr>
<th>SIMView</th>
<th>SIMView is the interface between the GSM application and any SIM Toolkit applet.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMSystem</td>
<td>The Class SIMSystem provides a way to get access to the GSM file system.</td>
</tr>
<tr>
<td></td>
<td>In any case, the SIM Toolkit applet will only access to methods of the SIMView interface. No instance of this class is needed.</td>
</tr>
</tbody>
</table>
import javacard.framework.*; import sim.toolkit.*;
public class MyApplet extends Applet implements ToolkitInterface {
private SIMView simView; private byte[] buffer;
private ToolkitRegistry registry;
public MyApplet () {
    registry = ToolkitRegistry.getEntry();
    simView = SIMSystem.getTheSIMView();
    buffer = new byte[32];
}
public static void install(APDU apdu) throws ISOException {
    MyApplet applet = new MyApplet();
    applet.register();
}
public void getADN(short adnNumber) {
    simView.select(SIMView.FID_EF_TELECOM);
    simView.select(SIMView.FID_EF_ADN);
    simView.readRecord((short)adnNumber, SIMView.MODE_ABSOLUTE,
                        (short)0, buffer, (short)0, (short)32);
}
Any question?