Smartcards

ISO 7816 & smartcard operating systems

Erik Poll
Digital Security
Radboud University Nijmegen
Standard for contact smartcards ISO7816

• 7816-1 Physical characteristics
• 7816-2 Dimension & size of contacts
• 7816-3 Electronic signals and transmission protocols
  - defines voltage & current requirements
• 7816-4 Inter-industry commands
  - standard set of commands
• 7816-5 Numbering system for application identifiers (AIDs)
• 7816-6 ...
• 7816-...
• 7816-...
• 7816-15
Contact cards (ISO 7816-2)

1. Vcc
2. Clock
3. Reset
4 & 8 RFU (Reserved for Future Use)
4. Ground
5. Vpp
6. I/O
7. Vcc originally 5 V, now also 3V or 1.8V
   - Vpp, higher voltage for writing to EEPROM, no longer used as it introduces security weakness
8. Clock originally 3.57 MHz or 4.92MHz
9. I/O speeds in order of > 100 Kbit/s
   - C4 & C8 can be used for USB2.0 up to 12 Mbit/s
10. C6 can be used for Single Wire Protocol (SWP) to connect SIM card to the phone’s NFC antenna
Smart card terminals

Master-Slave communication:

terminal (aka CAD, card acceptance device) is master
smartcard is slave

Hence: terminal takes the initiative,
smartcard cannot initiate actions

For SIM cards a polling mechanism is used to overcome this limitation: the handset will regularly poll the SIM card to ask if it wants to do something
The Terminal Problem!

No I/O between user and card
- no display
- no keyboard

Why is this a problem?

Some experimental cards with displays, keyboards, or fingerprint readers.
Trusted I/O to the card holder

I/O via such devices is always trusted but not necessarily trustworthy
Card Activation (ISO 7816-3)

1. terminal activates card
   - earth; voltage; clock; reset

2. card responds with ATR (Answer To Reset)
   - max 33 bytes, usually a lot less (for speed)
   - must be sent between 400 & 40,000 clock cycles
     - obligatory info about the protocol used
       - T=0 byte-oriented
       - T=1 block-oriented
     - supported baud rate for I/O
     - usually some manufacturer info
       - id of OS and version no. of ROM mask
     - obligatory last byte XOR checksum
**APDU communication (ISO 7816-4)**

All subsequent communication via **APDUs**

*Application Protocol Data Units*

which are just sequences of bytes in particular format

1. Terminal sends **command APDU**
2. Card replies with **response APDU**

etc, etc ....
Command APDU

<table>
<thead>
<tr>
<th>CLA</th>
<th>INS</th>
<th>P1</th>
<th>P2</th>
<th>Lc</th>
<th>...Data....</th>
<th>Le</th>
</tr>
</thead>
</table>

- **CLA** class byte
- **INS** instruction byte
- **P1,P2** parameters
  - **Lc** length of data block
  - **Data** Lc bytes of data
  - **Le** length of expected response

  obligatory

  optional
Response APDU

| Data ... | SW1 | SW2 |

- **Data**: Le bytes of data *(optional)*
- **SW1, SW2**: status word *(obligatory)*
APDU coding conventions

• Conventions for CLA, INS etc. are given in ISO 7816-4

• Conventions for status word SW1 SW2
  - normal processing 61xx, 9000
  - warning processing 62xx, 63xx
  - execution error 64xx, 65xx
  - coding error 67xx, 6Fxx
Logical channels

- Modern cards provide several logical channels to talk to multiple applications on the card concurrently
  - eg mobile phone talking to contact list and other applets on SIM card
Future

ISO7816 protocol stems from 1880s and it shows!
Slow speed & small size of APDUs can be a bottleneck

• Faster communication speeds wanted?
  - eg. USB 2.0

• More modern protocols wanted?
  - eg http(s) support on experimental JavaCard 3.0x Connected edition
Smartcard software and Operating Systems
Smartcard operating systems

• Similar evolution as for normal OSs, but faster

• Still *very* primitive compared to normal OSs such as Windows or Linux/UNIX
  - no multi-programming, hardly any I/O, ...
  - but multi-threading in newest JavaCard 3.0...
Smartcard OS

Tasks:

- **life-cycle management**
  - of card + individual applications (called **applets**)

- instruction processing

- memory management

- I/O

- **hardware error handling**
  - incl. support for atomic EEPROM updates, needed because the possibility of power failure by card tear
Typical application life cycle

- **installation of application (aka applet)**
  - uploading & installing code

- **personalisation**
  - uploading application data
  - afterwards, application starts in normal active life

- **end-of-life**
  - disabling all functionality
  - possibly leaving logging functionality enabled
  - upon external command or because the card notices something fishy going on
Smartcard life cycle (ISO 10202-1 - cancelled)

- Production of chip & card
  - testing & removing test functionality
- Card preparation
  - completing OS
- Application preparation incl. Personalisation
  - initialising applications
  - personalisation aka individualisation
    - both electrically & optically
- Card utilisation
  - (de)activation of applications
- End of card utilisation
  - de-activating applications
  - de-activating card
Smartcard OS evolution

1. no OS: one application, burnt into ROM
2. standard libraries in ROM, applications in EEPROM
3. proprietary operating systems
   - programs written in machine code or C
   - providing standardised file system (ISO7816-4) with access control
4. modern multi-application smartcards
   - MULTOS
   - JavaCard
5. next generation, experimental `concept' card
   - JavaCard 3.n Connected Edition
OS "completion"

1. Initially, card contains ROM mask
2. Simple loader in ROM executed to load EEPROM
3. Checksum computed
4. Switch to mode where code in ROM and EEPROM can be executed
Modern multi-application cards

- **Multi-application**: more than one program (applet) on a card
- **Post-issuance download**: applet can be installed or removed on issued cards in the field
- **Programs written in a standard high(er) level language**, not a proprietary instruction set specific to chip vendor

Examples

- **MULTOS**
  - first of these "modern" smartcard OSs
- **JavaCard**
  - popular as GSM sims, used in Dutch passport & ID cards
- **Windows for Smartcards †**
  - since abandoned
Modern multi-application cards

Pros

• vendor-independence
  • old cards have proprietary OSs and instruction sets
• fast development & quick time-to-market
  • esp. important in telecom market

Cons

• overhead – more memory & CPU power needed
  • more expensive card needed
• complexity
  • which brings security concerns

Open question: do security advantages of a higher level platform, with built-in standard security mechanisms, outweigh the additional security risks this complexity brings?

Financial sector much more conservative than telecom sector in adopting modern platforms.
Multi-application cards

• Multi-application vision: everyone carrying *one* card, with all their smartcard applications
• This is not going to happen. Problems include:
  - **trust**
    bank won't allow untrusted applet code on their cards, despite security guarantees of VM / OS
  - **marketing**
    who gets to put their logo on the plastic

• Still, multi-application is useful for development & card management
  - eg adding services to GSM SIM, adding contactless or internet banking applets to bank card, ...
MULTOS

• Card provides a Virtual Machine interpreting MEL (MULTOS Executable Language)

• Originally developed for electronic purse system Mondex
  - by BT, Westminster & Midland banks in the UK

• Designed for ITSEC EC6-high evaluation
JavaCard 3.0 Connected Edition

The next-generation smart card OS

- multi-threading
  security worries!
- communication with https://
  The smartcard is a web-server!

But who will use it??

- intended market: telco
- not all card manufacturers produce JC 3.0 Connected, or have any intention to