Smartcard protocols
ISO 7816

Erik Poll

Digital Security

Radboud University Nijmegen

Standard for contact smartcards ISO 7816

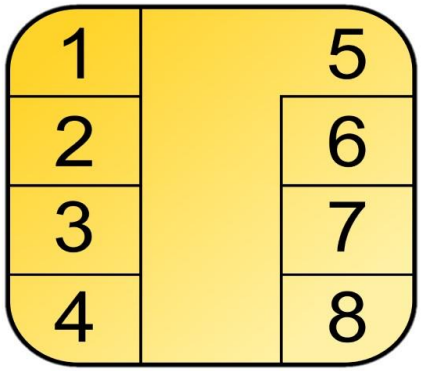
- 7816-1 Physical characteristics
- 7816-2 Dimension & size of contacts
- 7816-3 Electronic signals and transmission protocols
  - defines voltage & current requirements
- 7816-4 Inter-industry commands
  - standard set of commands
- 7816-5 Numbering system for application identifiers (AIDs)
- 7816-6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

For contactless (aka proximity) cards: ISO 14443

Additional application-level specs for bank cards (EMV), SIM cards, e-passports, e-driving licenses, …

Contact cards (ISO 7816-2)

1. Vcc
2. Clock
3. Reset



5. Ground
6. Vpp
7. I/O

4 & 8 RFU (Reserved for Future Use)

- 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
- Clock originally 3.57 MHz or 4.92MHz
- I/O speeds in order of > 100 Kbit/s
  - C4 & C8 can be used for USB2.0 up to 12 Mbit/s
- C6 can be used for Single Wire Protocol (SWP) to connect SIM card to the phone's NFC antenna

Communication with 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 was used to overcome this limitation: the handset regularly polls 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 devices such as  means these have to *trusted*

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

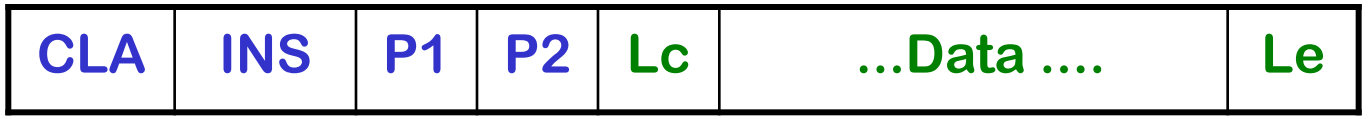
Application Protocol Data Units

which are just sequences of bytes in particular format

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

etc, etc ....

Command APDU



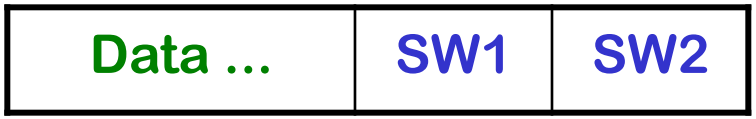
- CLA class byte
- INS instruction byte
- P1,P2 parameters

} obligatory

- Lc length of data block
- Data Lc bytes of data
- Le length of expected response

} optional

Response APDU



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

APDU coding conventions

- Conventions for some CLA& INS bytes 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

ISO 7816 standard commands

- ISO 7816-4 also standardises some functionality & associated commands, for
  - a file system
  - PIN codes
  - (building blocks for) authentication protocols
- You do not have to stick to using & implementing these commands, but they may provide inspiration.
  - The aim was to standardise basic functionality, so that applications can created by configuring standard functionality rather than 'real' programming
- Other standards for more specific functionality:
  - EMV for banking cards
  - GSM 11.11 and its superset EN 726-3 for SIM cards
  - United Nations ICAO specs for e-passport
  - IS0 18013 for e-driving license

PIN verification command in ISO 7816

- `VERIFY` command for PIN code verification
  - aka CHV = Card Holder Verification = PIN

ISO 7816-4 defines that instruction (INS) byte for `VERIFY` is 20, and class (CLA) byte is 00

Example Authentication Protocols

ISO 7816-4 defines some standard instructions for authentication using challenge-response

- For authentication of *card* by terminal
  - `INTERNAL AUTHENTICATE`
    - arguments: random, algorithm , key no
    - card returns: enc(key,random)
- For authentication of *terminal* by the card
  - `GET CHALLENGE`
    - card returns random number
  - `EXTERNAL AUTHENTICATE`
    - arguments: enc(key, random), algorithm, key no

Example Authentication Protocols

For *mutual* authentication

- `GET CHIP NUMBER`
  - card returns `chip number`
- `GET CHALLENGE`
  - card returns smart card random, `s_rnd`
- `MUTUAL AUTHENTICATE`
  - arguments: `key, terminal random, s_rnd, chip number, algorithm,`
  - card returns: `enc(key, terminal random || s_rnd)`

Future

ISO7816 protocol stems from 1980s 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?
  - There has been an experimental JavaCard 3.0 version with https support