University
of
Nijmegen

Creating a Java Card applet

Engelbert Hubbers
Informatica voor Technische Toepassingen
Nijmeegs Instituut voor Informatica en Informatiekunde
Katholieke Universiteit Nijmegen
hubbers@cs.kun.nl

Overview

1. ISO 7816

Overview

1. ISO 7816
2. APDUs

Overview

1. ISO 7816
2. APDUs
3. Applet creation

Overview

1. ISO 7816
2. APDUs
3. Applet creation
4. Terminal creation

Overview

1. ISO 7816
2. APDUs
3. Applet creation
4. Terminal creation
5. Security protocol as FSM

Overview

1. ISO 7816
2. APDUs
3. Applet creation
4. Terminal creation
5. Security protocol as FSM
6. FSM refinements

ISO 7816

- Several parts. . .

ISO 7816

- Several parts. . .
  - Part 1: Physical characteristics
  - Part 2: Dimensions and location of the contacts
  - Part 3: Electronic signals and transmission protocols
  - Part 4: Interindustry commands for interchange
  - Part 5 . . . 10

ISO 7816

- Several parts. . .
- Buy them at NEN: `http://www.nen.nl`

ISO 7816

- Several parts. . .
- Buy them at NEN: `http://www.nen.nl`
- Or search for copies on the internet
  - We have found 1, 2, 3 and 4 so far `http://www.cardwerk.com/smartcards`

ISO 7816-4

- Interindustry commands for interchange

ISO 7816-4

- Interindustry commands for interchange
- Application Protocol Data Unit

ISO 7816-4

- Interindustry commands for interchange
- Application Protocol Data Unit
- Command APDU

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

ISO 7816-4

- Interindustry commands for interchange
- Application Protocol Data Unit
- Command APDU

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

  - CLA: Class byte
  - INS: Instruction byte
  - P1, P2: Parameters
  - Lc: Length data block
  - Le: Expected length response

ISO 7816-4

- Interindustry commands for interchange
- Application Protocol Data Unit
- Command APDU

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

- Response APDU

Data	**SW1**	**SW2**

ISO 7816-4

- Interindustry commands for interchange
- Application Protocol Data Unit
- Command APDU

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

- Response APDU

Data	**SW1**	**SW2**

  - SW1, SW2: Status words

APDUs

- Four cases: parsing upon body length L

APDUs

- Four cases: parsing upon body length L
- $L = 0$

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

APDUs

- Four cases: parsing upon body length L
- $L = 0$
- $L = 1, Le \in \{1, \ldots, 256\}$

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

APDUs

- Four cases: parsing upon body length L
- $L = 0$
- $L = 1$, $Le \in \{1, \ldots, 256\}$
- $L = 1 + Lc$, $Lc \in \{1, \ldots, 255\}$

CLA	**INS**	**P1**	**P2**	**Lc**	**Data**	**Le**

APDUs

- Four cases: parsing upon body length L
- $L = 0$
- $L = 1, Le \in \{1, \ldots, 256\}$
- $L = 1 + Lc, Lc \in \{1, \ldots, 255\}$
- $L = 2 + Lc, Lc \in \{1, \ldots, 255\}, Le \in \{1, \ldots, 256\}$

CLA	INS	P1	P2	Lc	Data	Le

APDUs

- Four cases: parsing upon body length L
- $L = 0$
- $L = 1$, $Le \in \{1, \ldots, 256\}$
- $L = 1 + Lc$, $Lc \in \{1, \ldots, 255\}$
- $L = 2 + Lc$, $Lc \in \{1, \ldots, 255\}$, $Le \in \{1, \ldots, 256\}$

- Some cards can deal with *extended* lengths: $Lc \in \{1, \ldots, 65535\}$ and $Le \in \{1, \ldots, 65536\}$

Coding conventions

- CLA, INS, P1, P2, SW1 and SW2 are defined in the general ISO 7816-4

Coding conventions

- CLA, INS, P1, P2, SW1 and SW2 are defined in the general ISO 7816-4
- or in the specific documentation of an application

Coding conventions

- CLA, INS, P1, P2, SW1 and SW2 are defined in the general ISO 7816-4
- or in the specific documentation of an application

SW1 SW2					
Process completed			Process aborted		
Normal processing	Warning processing		Execution error		Checking error
61XX 9000	62XX	63XX	64XX	65XX	67XX to 6FXX

Standard functions

INS	Function
A4	Select file
B0	Read binary
B2	Read records
C0	Get response
CA	Get Data
D0	Write binary
D2	Write record
E2	Append record

Java Card API

- `java.lang`
  Object, Throwable, Exception,. . .

Java Card API

- `java.lang`
  Object, Throwable, Exception,. . .
- `javacard.framework`
  ISO7816, APDU, Applet, JCSystem,. . .

Java Card API

- `java.lang`
  Object, Throwable, Exception,. . .
- `javacard.framework`
  ISO7816, APDU, Applet, JCSystem,. . .
- `javacard.security`
  KeyBuilder, RSAPrivateKey, CryptoException,. . .

Java Card API

- `java.lang`
  Object, Throwable, Exception,. . .
- `javacard.framework`
  ISO7816, APDU, Applet, JCSystem,. . .
- `javacard.security`
  KeyBuilder, RSAPrivateKey, CryptoException,. . .
- `javacardx.crypto`
  Cipher

Installation

- Start with Java file

University
of
Nijmegen

Installation

- Start with Java file
- Compile into CLASS files using any Java compiler

Installation

- Start with Java file
- Compile into CLASS files using any Java compiler
- Convert into CAP file using Sun’s converter

Installation

- Start with Java file
- Compile into CLASS files using any Java compiler
- Convert into CAP file using Sun's converter
- The converter creates simultaneously an EXP file

Installation

- Start with Java file
- Compile into CLASS files using any Java compiler
- Convert into CAP file using Sun’s converter
- The converter creates simultaneously an EXP file
- CAP file verifier checks CAP file

Installation

- Start with Java file
- Compile into CLASS files using any Java compiler
- Convert into CAP file using Sun's converter
- The converter creates simultaneously an EXP file
- CAP file verifier checks CAP file
- Off card installation program and on card installer load the applet on the card

Example: PayApplet

- See `PayApplet.java`
  - `process`
  - `readBuffer`
  - `install`

Terminal application

- Less restrictions: no need to use Java Card

Terminal application

- Less restrictions: no need to use Java Card
- C: use PC/SC API

Terminal application

- Less restrictions: no need to use Java Card
- C: use PC/SC API
- Java: use OCF API

Terminal application

- Less restrictions: no need to use Java Card
- C: use PC/SC API
- Java: use OCF API
  - which can be built on top of PC/SC API

Finite State Machines

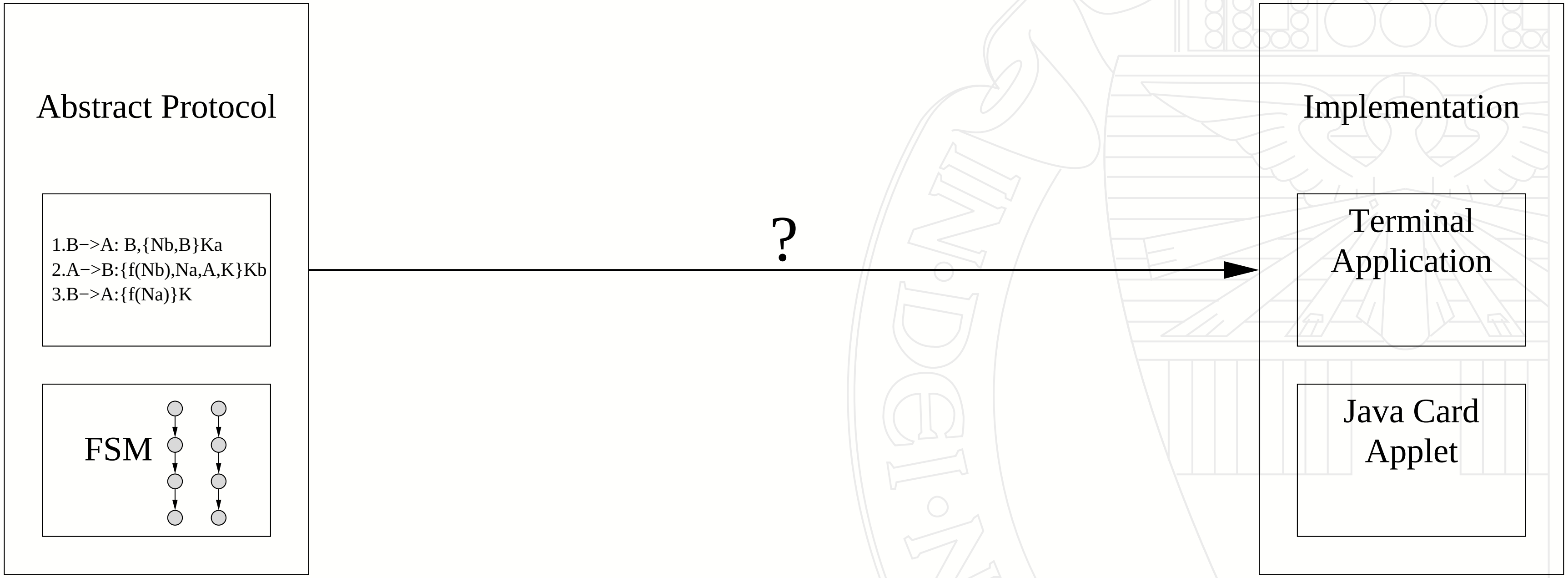
- How to get from an abstract security protocol…

Finite State Machines

- How to get from an abstract security protocol. . .
- . . . to a Java Card implementation?

Finite State Machines

- How to get from an abstract security protocol…
- …to a Java Card implementation?



1.B−>A: B,{Nb,B}Ka
2.A−>B:{f(Nb),Na,A,K}Kb
3.B−>A:{f(Na)}K

Abstract security protocol

- Bilateral Key Exchange with public key

Abstract security protocol

- Bilateral Key Exchange with public key

  1. $A \rightarrow B : A, \{N_a, A\}_{K_b}$
  2. $B \rightarrow A : \{N_a, N_b, B, K\}_{K_a}$
  3. $A \rightarrow B : \{N_b\}_K$

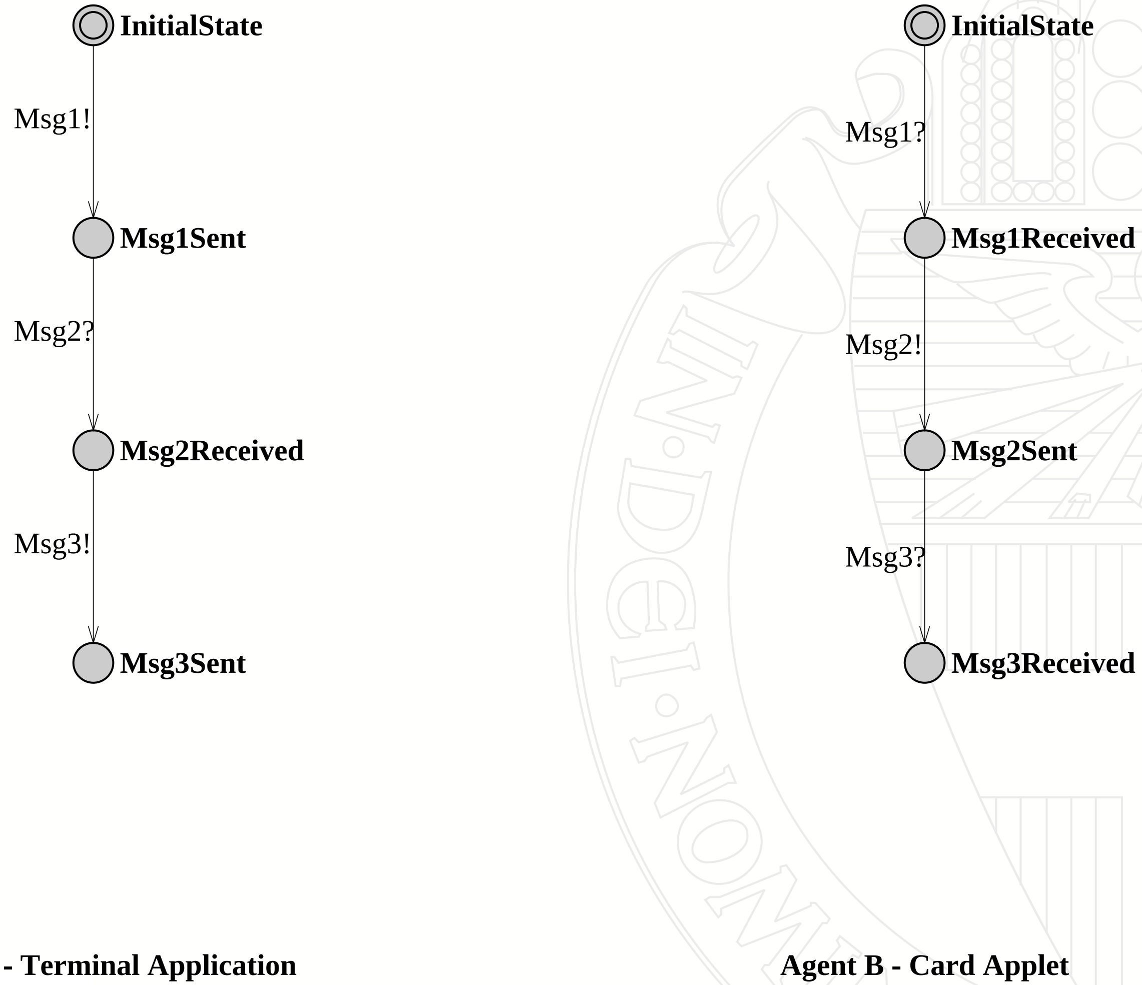
- A and B: agents
- N_a and N_b: their nonces (challenges)
- K_a and K_b: their public keys
- $\{\ldots\}_K$: data … encrypted using key K

Abstract security protocol 2

- Alternative description: finite state machines

Abstract security protocol 2

- Alternative description: finite state machines



Refinement - extending

- Observation

Refinement - extending

- Observation
  - Protocol describes how to agree on a session key

Refinement - extending

- Observation
  - Protocol describes how to agree on a session key
  - It does not describe how to use this session key

Refinement - extending

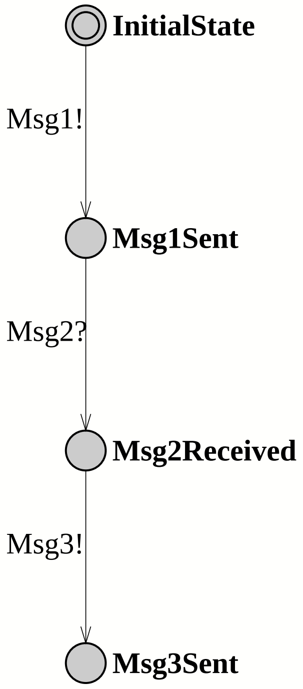
- Observation
  - Protocol describes how to agree on a session key
  - It does not describe how to use this session key
- Decide how to deal with this in the implementation

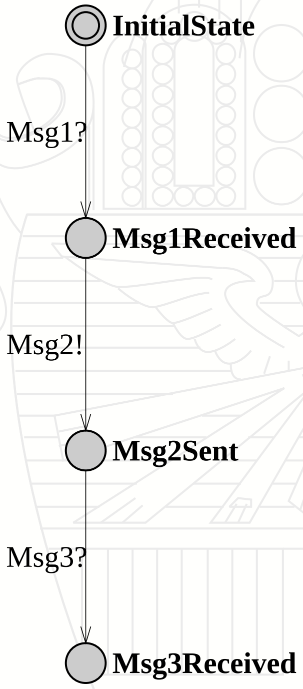
Refinement - extending

- Observation
  - Protocol describes how to agree on a session key
  - It does not describe how to use this session key
- Decide how to deal with this in the implementation
- Note that the actual –quite trivial– choices made here are not the issue!

Refinement - extending 2

- Automata



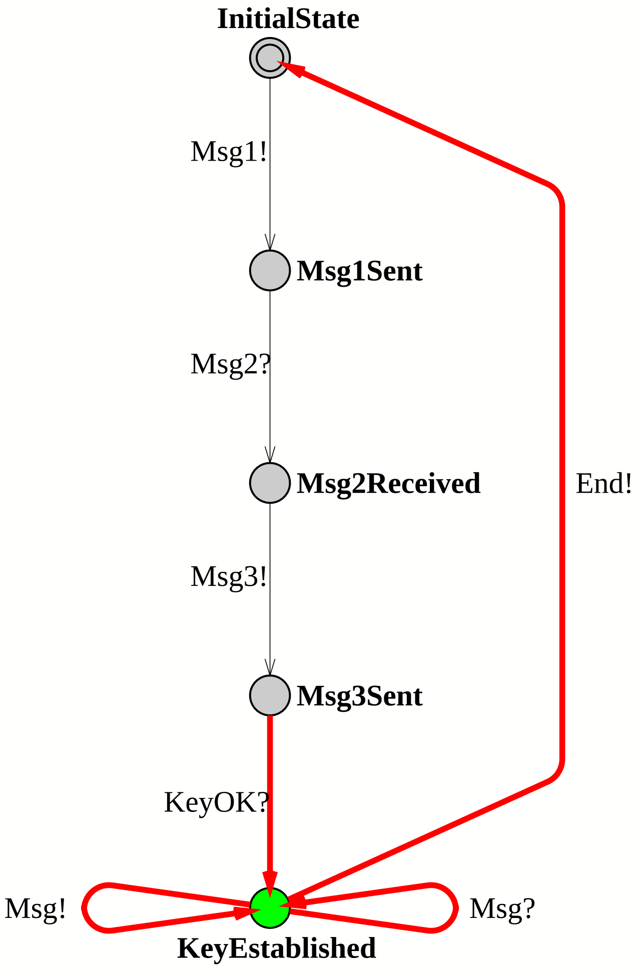


Agent A - Terminal Application

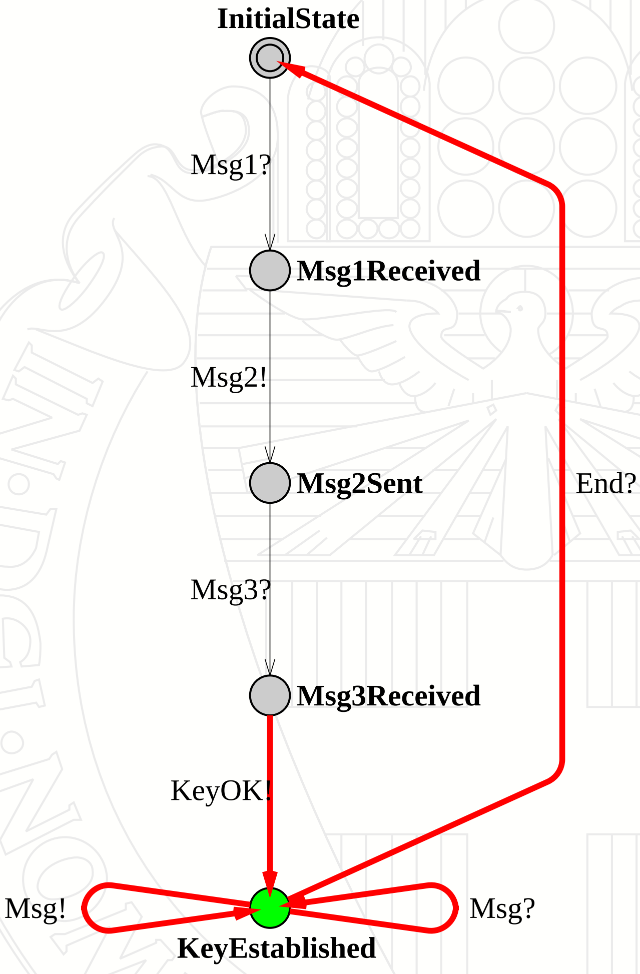
Agent B - Card Applet

Refinement - extending 2

- Automata



Agent A - Terminal Application



Agent B - Card Applet

Refinement - input enabling

- Observation

Refinement - input enabling

- Observation
  - Protocol only describes correct runs

Refinement - input enabling

- Observation
  - Protocol only describes correct runs
  - It does not describe how to handle exceptional situations

Refinement - input enabling

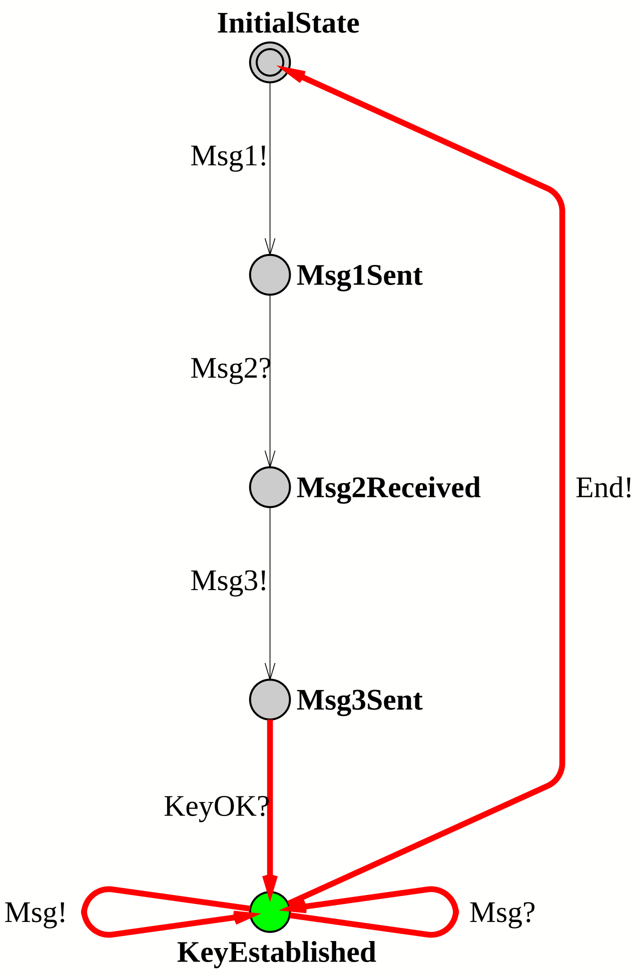
- Observation
  - Protocol only describes correct runs
  - It does not describe how to handle exceptional situations
    - Unsolicited messages
    - Errors while processing expected messages
    - Failure of the communication channel

Refinement - input enabling

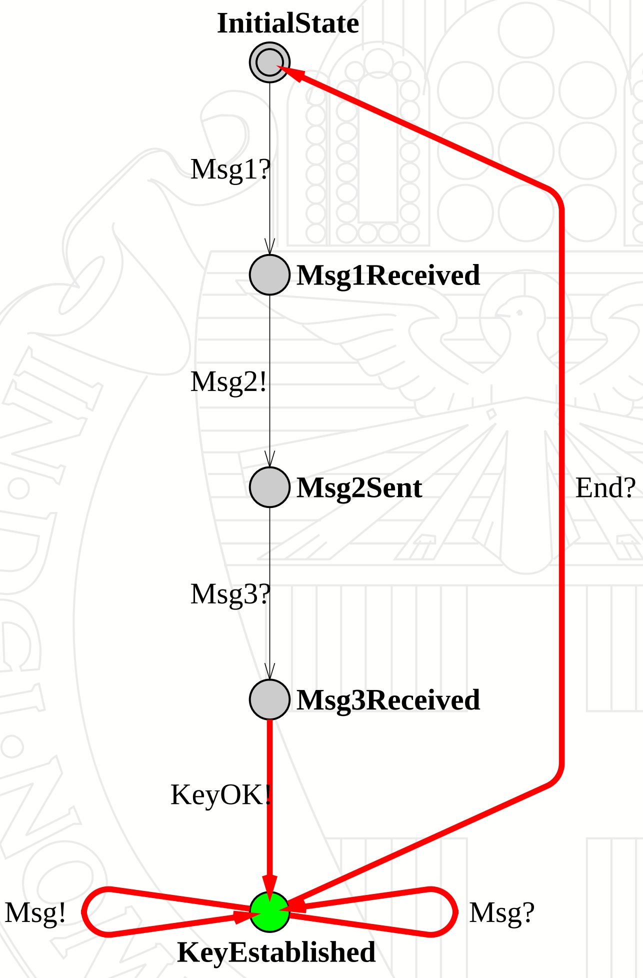
- Observation
  - Protocol only describes correct runs
  - It does not describe how to handle exceptional situations
    - Unsolicited messages
    - Errors while processing expected messages
    - Failure of the communication channel
- Decide how to react in these situations

Refinement - input enabling 2

- Automata



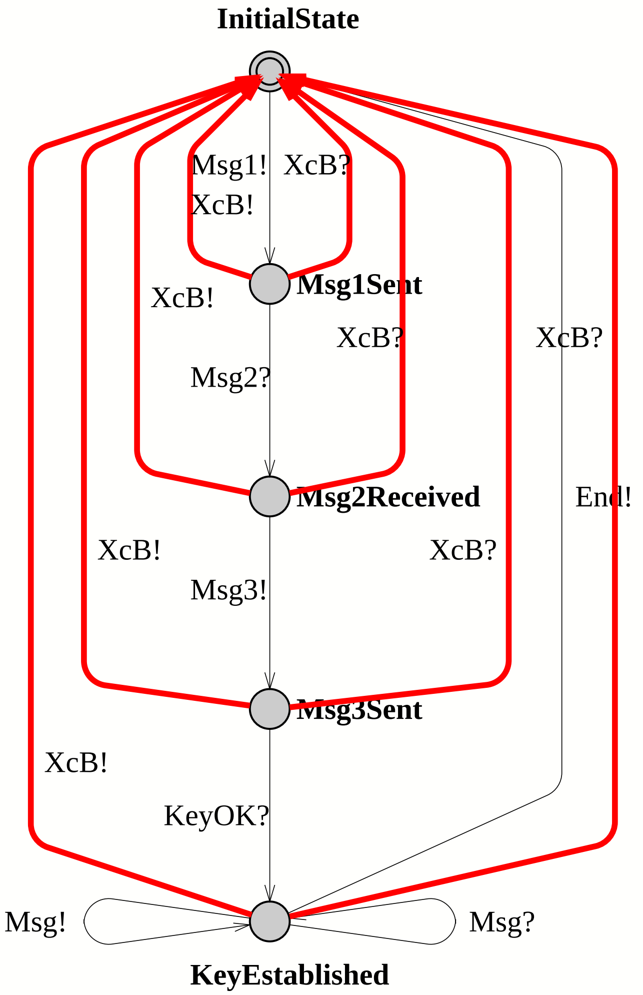
Agent A - Terminal Application



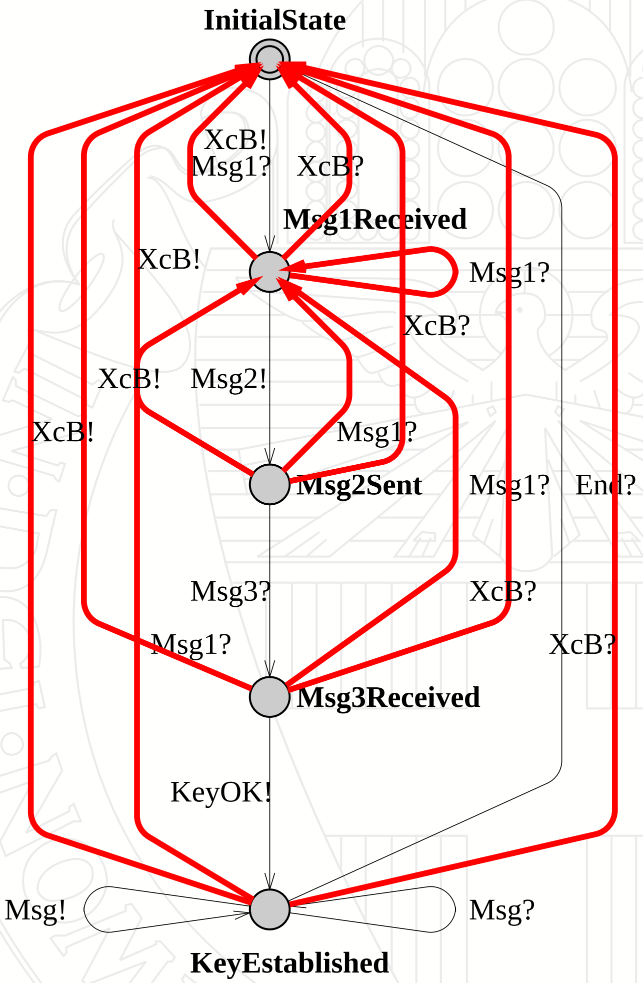
Agent B - Card Applet

Refinement - input enabling 2

- Automata



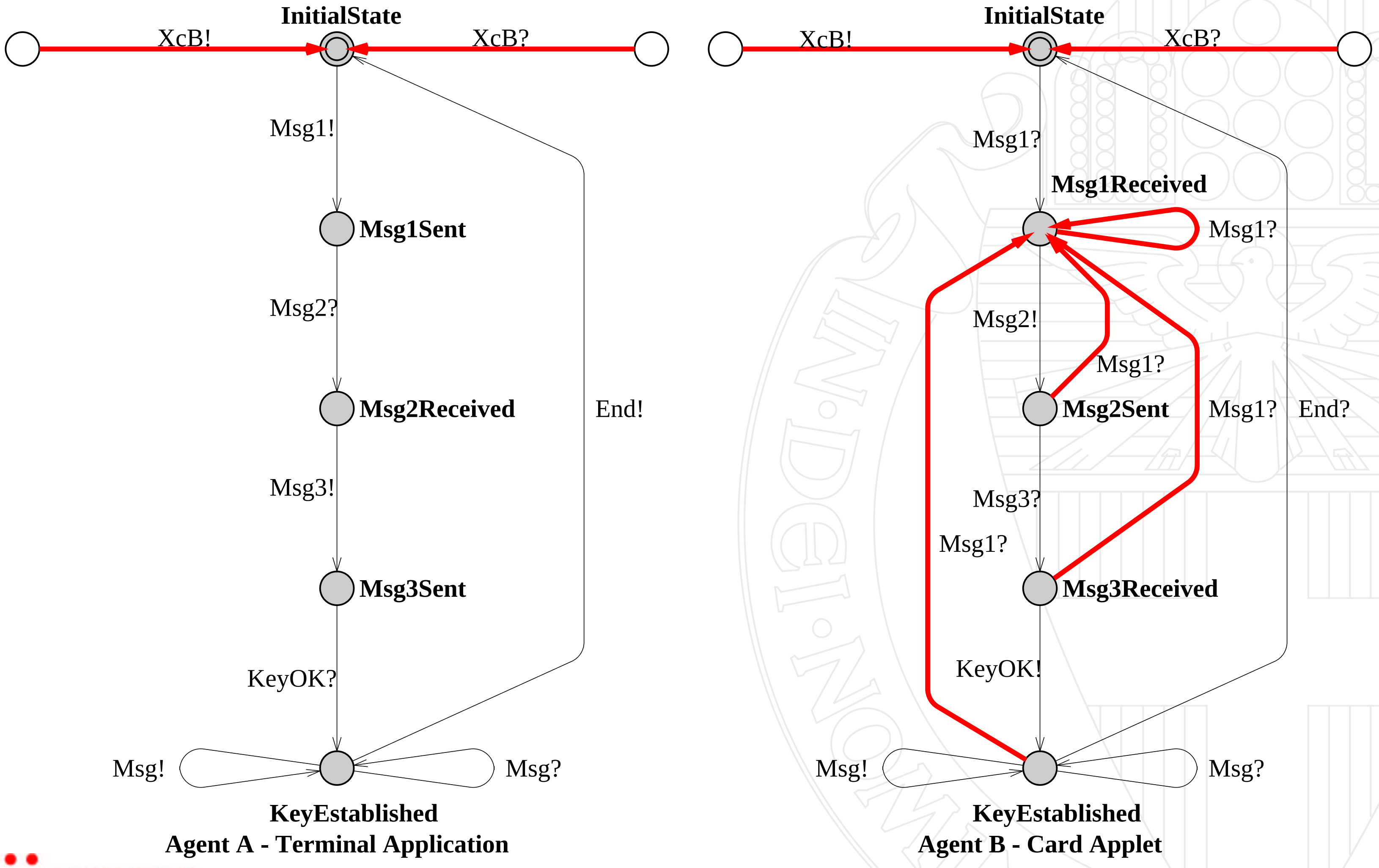
Agent A - Terminal Application



Agent B - Card Applet

Refinement - input enabling 2

- Automata



Refinement - smart card tuning

- Typical for smart cards

Refinement - smart card tuning

- Typical for smart cards
  - Initialization phase
  - Applet selection
  - Persistent or transient memory
  - Card tears
  - Command-response pairs

Refinement - smart card tuning

- Typical for smart cards
  - Initialization phase
  - Applet selection
  - Persistent or transient memory
  - Card tears
  - Command-response pairs
- Decide how to deal with these issues

Refinement - smart card tuning 2

- Initialization phase

Refinement - smart card tuning 2

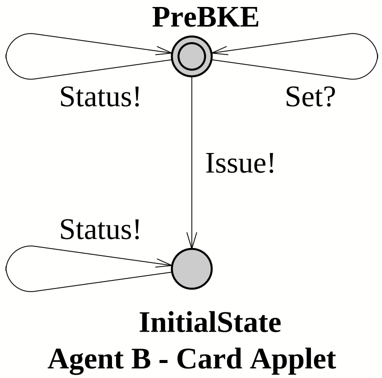
- Initialization phase
  - Each card needs to be personalized before any BKE run
    - Its id
    - Its own private key
    - The public keys of all valid terminals

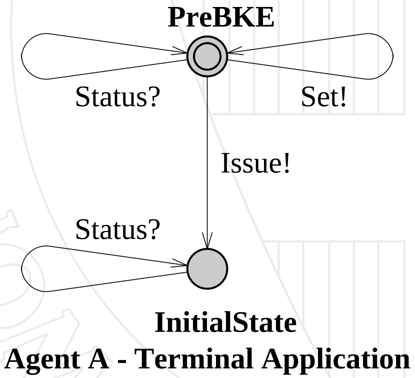
Refinement - smart card tuning 2

- Initialization phase
  - Each card needs to be personalized before any BKE run
    - Its id
    - Its own private key
    - The public keys of all valid terminals
  - Once personalized these settings cannot be modified

Refinement - smart card tuning 2

- Initialization phase
  - Each card needs to be personalized before any BKE run
    - Its id
    - Its own private key
    - The public keys of all valid terminals
  - Once personalized these settings cannot be modified





Refinement - smart card tuning

- Applet selection

Refinement - smart card tuning 3

- Applet selection
  - Multi application platform: Java Card applets need to be selected

Refinement - smart card tuning 3

- Applet selection
  - Multi application platform: Java Card applets need to be selected
  - Go to a different state based upon personalization flag

Refinement - smart card tuning

- Persistent or transient memory

Refinement - smart card tuning

- Persistent or transient memory
  - Persistent memory (EEPROM)
    - Card’s id
    - Private and public keys
    - Personalization flag

Refinement - smart card tuning

- Persistent or transient memory
  - Persistent memory (EEPROM)
    - Card's id
    - Private and public keys
    - Personalization flag
  - Transient memory (RAM)
    - Protocol state
    - Session key

Refinement - smart card tuning 5

- Card tears

Refinement - smart card tuning 5

- Card tears
  - What can the card do after a power failure?

Refinement - smart card tuning

- Card tears
    - What can the card do after a power failure?
        - Nothing!

Refinement - smart card tuning 5

- Card tears
  - What can the card do after a power failure?
    - Nothing!
  - What can the card do after it is powered up again?

Refinement - smart card tuning 5

- Card tears
  - What can the card do after a power failure?
    - Nothing!
  - What can the card do after it is powered up again?
    - Automatically clean up all session information

Refinement - smart card tuning 6

- Command-response pairs

Refinement - smart card tuning

- Command-response pairs
  - Master-slave relation

Refinement - smart card tuning

- Command-response pairs
  - Master-slave relation
    - Master: terminal application, agent A
    - Slave: card applet, agent B

Refinement - smart card tuning

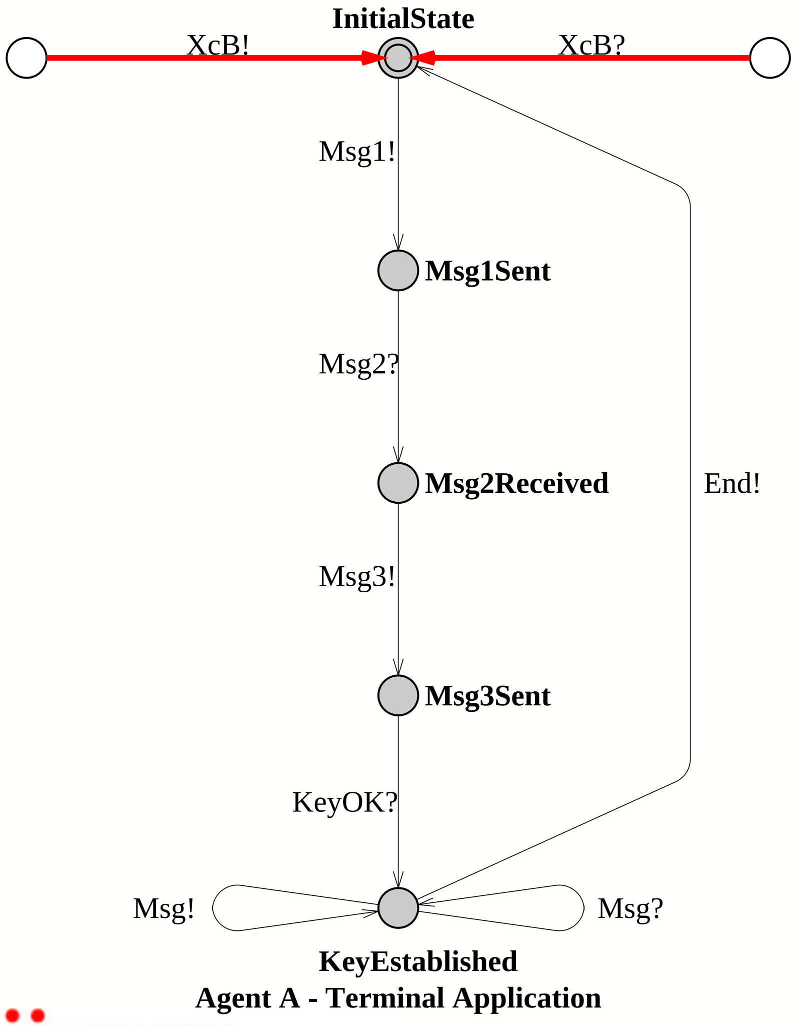
- Command-response pairs
  - Master-slave relation
    - Master: terminal application, agent A
    - Slave: card applet, agent B
  - All incoming messages from B need to be answered by A

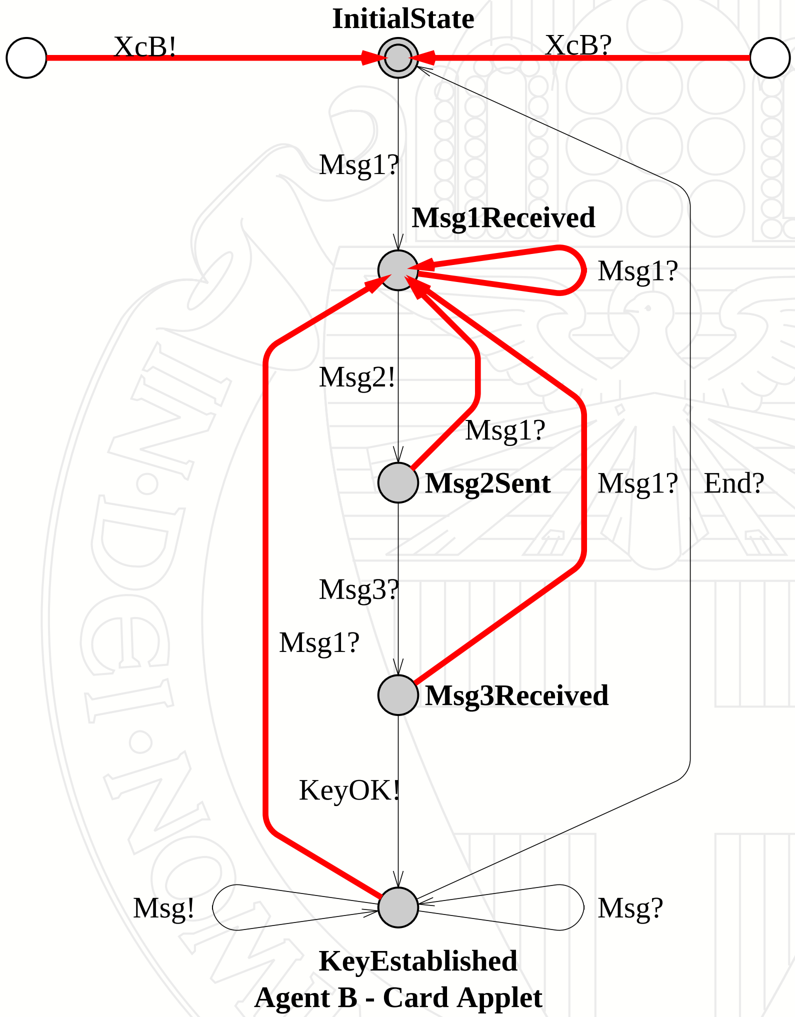
Refinement - smart card tuning

- Automata

Refinement - smart card tuning

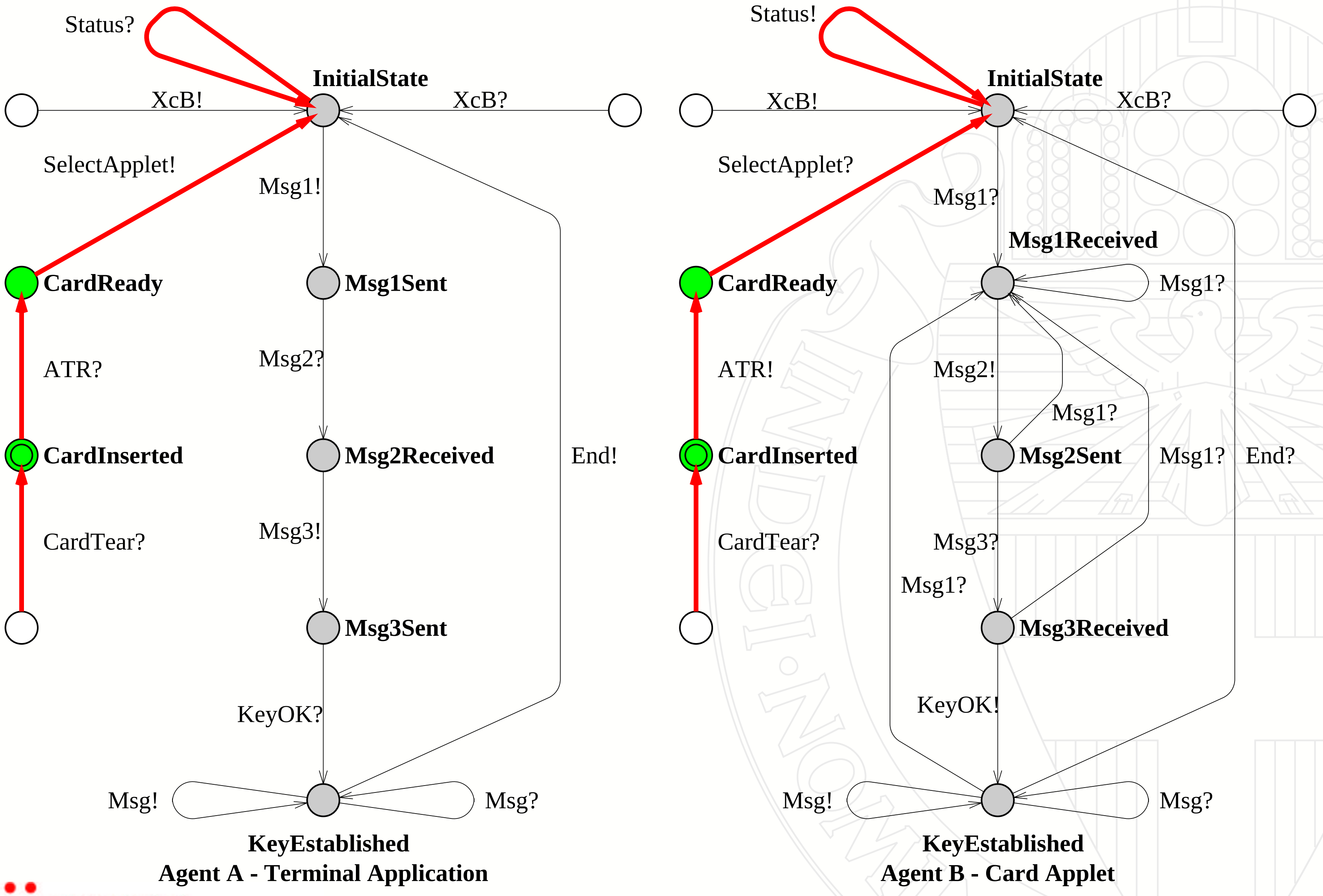
- Automata





Refinement - smart card tuning

- Automata



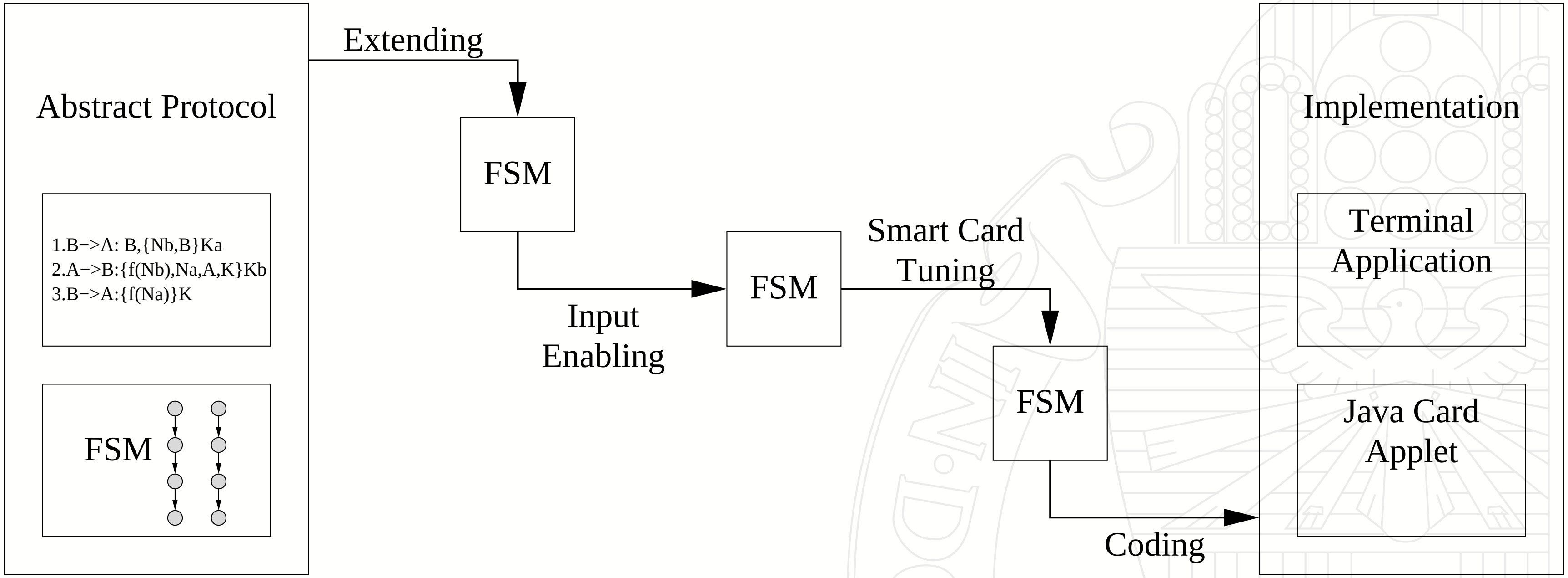
Coding

- Manual derivation of Java code for the applet

University of Nijmegen

Coding

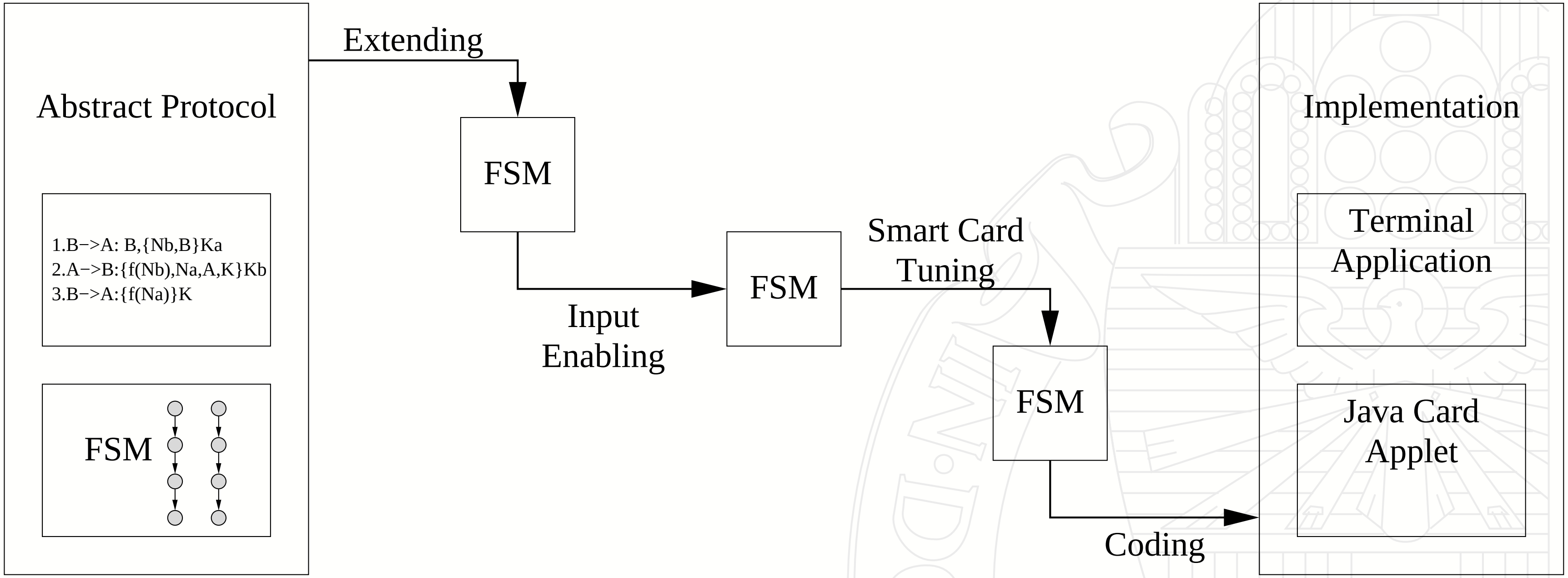
- Manual derivation of Java code for the applet



nijmeegs instituut voor informatica en informatiekunde

Coding

- Manual derivation of Java code for the applet



- Are these intermediate steps safe with respect to security properties?

References

[1] Z. Chen. *Java Card Technology for Smart Cards*. The Java Series. Addison-Wesley, 2000.

[2] E. Hubbers, M. Oostdijk, and E. Poll. Implementing a formally verifiable security protocol in Java Card. In *Proceedings of the 1st International Conference on Security in Pervasive Computing*, LNCS. Springer-Verlag, 2003. To appear.

[3] ISO7816 Information technology – Identification cards – Integrated circuit(s) card with contacts