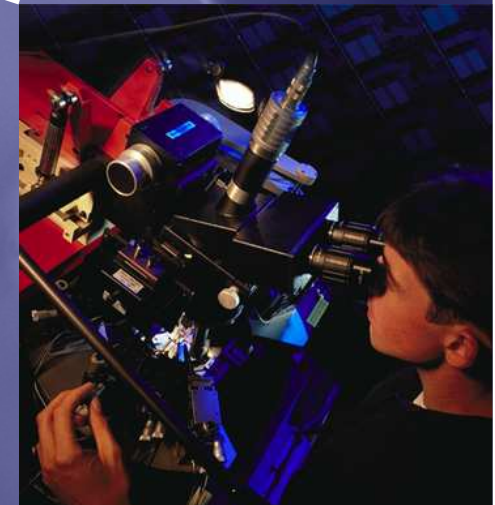


# Formal evaluation of Java Card smart cards

TNO TPD



Joachim van den Berg



# Contents of this talk

- TNO EIB and TNO ITSEF bv
- Common criteria evaluations
- Smart cards and Java Card
- Examples for which models have to be improved

## TNO EIB

- **Evaluation centrum for Information Security**
- **Formulating security requirements and test and approval procedures for information systems**
- **Restricted to evaluations (no product development activities)**
- **Evaluations during whole trajectory of development of security products**
- **Strict procedures for maintaining client secrecy of sensitive information**

# TNO EIB

- **Customers:**
  - Endusers
    - Financial organisations
    - Governments
    - Mobile network operators
  - Producers of secure components like
    - Chip manufacturers
    - Smart card manufacturers
    - Terminal manufacturers

# TNO EIB becomes TNO ITSEF BV

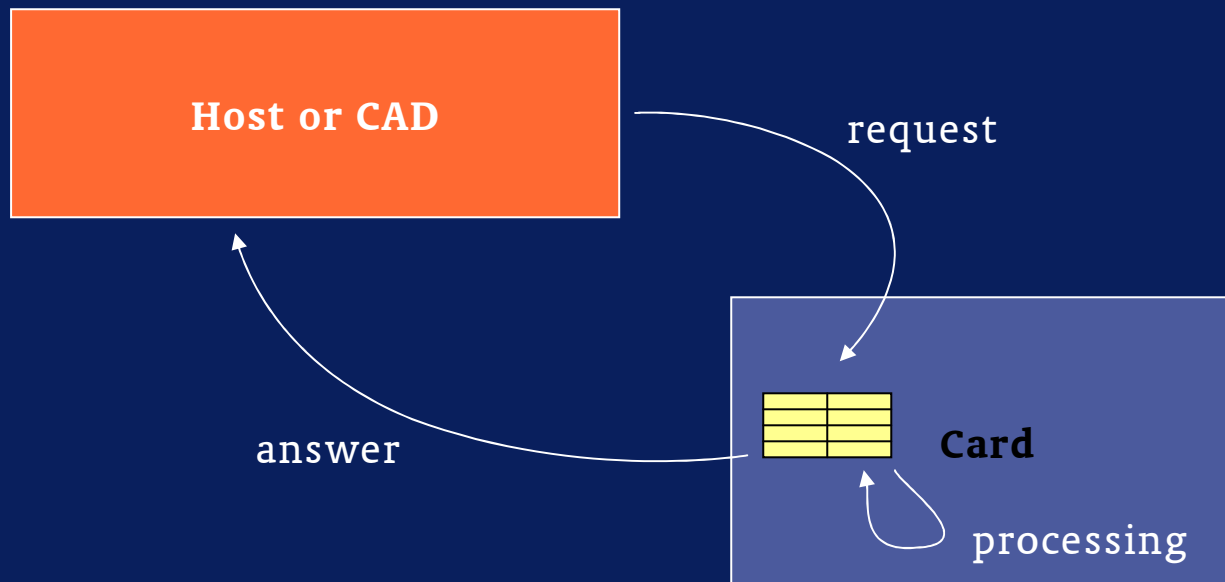
- Teams up with TNO ITSEF BV, per January 1, 2004
- ITSEF: IT Security Evaluation Facility
- Staff of 40 experts (B.Sc, M.Sc., Ph.D)
- Accredited lab for:
  - Common Criteria, MasterCard, Visa, ZKA, Interpay, ...
- Common Criteria (CC) evaluations
  - For IT security components
  - Use of formal methods for higher evaluation levels

# About smart cards

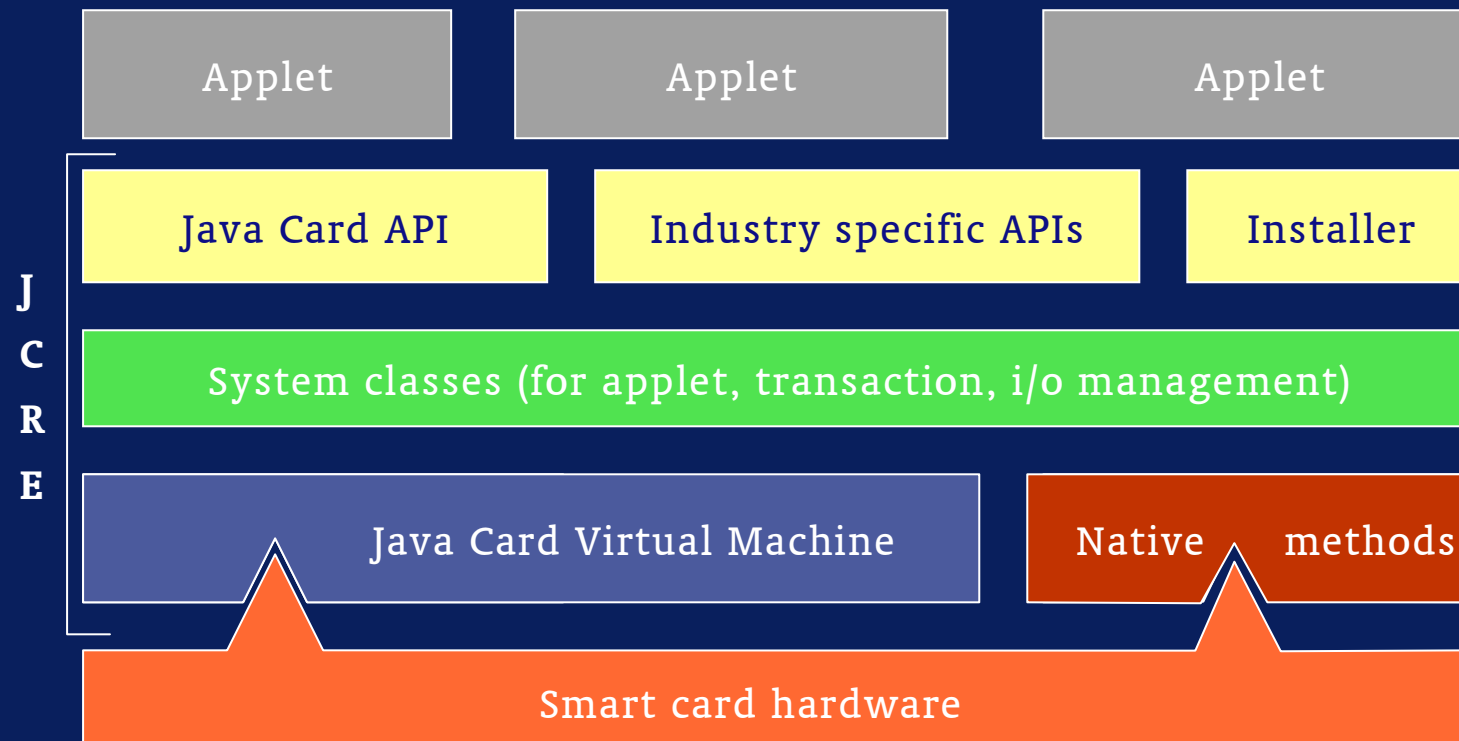
- Smart cards are security devices that may hold personal information protected by a secret.
- Smart cards are currently used in many areas, such as
  - Finance
  - Health
  - Telecom
- Old smart cards have a proprietary operating system which makes it difficult to develop applications
- Market shifts to Java Card smart cards
- One platform: less effort to develop applications

# Communication with a smart card

- Step 1: Host sends request to the card
- Step 2: Card processes this request
- Step 3: Card responds with an answer



# Architecture of a smart card



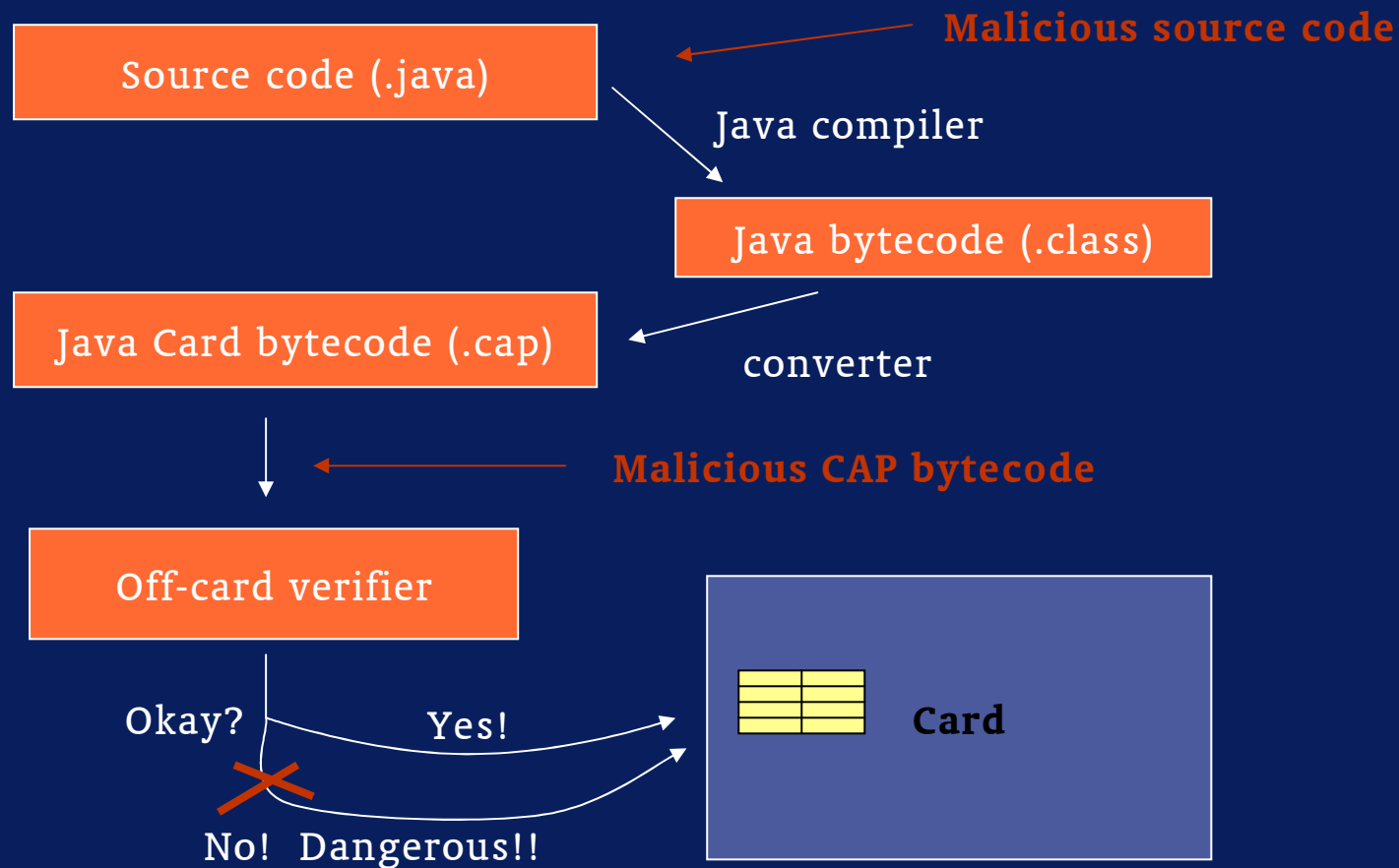
# Java Card smart cards

- **Java Card is intended for smart cards and memory-constrained devices**
- **Java Card is alike Java, but much and much smaller**
- **Therefore, it excludes**
  - Large datatypes
  - Multi-dimensional arrays
  - Garbage collection
  - On-card bytecode verifier (until now)
- **Multiple applications may be stored on a card**
- **It supports post-issuance downloading of applications**

## Java Card smart cards (2)

- Java's sandbox model does not exist in Java Card
- Verification of bytecode happens off-card
- Therefore, other security measures are implemented
- Security features of Java Card include
  - Transaction mechanism with roll-back (atomicity)
  - Applet firewall and object sharing
  - Persistent and transient memory models

# Application development and loading



# TNO and Java Card security

- **Evaluation of smart cards:**
  - External attacks (voltage manipulation, DPA, DFA, ...)
  - Internal attacks (probing, FIB, optical ROM analysis, ...)
- **As a security lab, TNO EIB tests the security features of Java Card:**
  - Transaction mechanism with roll-back
  - Applet firewall and object sharing
  - Persistent and transient memory models
  - Using verified CAP files
- **Joined forces with Riscure, Collis, KUN**
- **As a future CC lab, TNO ITSEF BV is interested to offer formal methods for security evaluations to customers**

# Common Criteria evaluations

- **Seven Evaluation Assurance Levels (EALs)**
- **EAL1-EAL5 make use of in- and semi-formal methods**
- **EAL6 and EAL7 require formal methods**
- **For Java Card applications:**
  - functional specification with Java Modelling Language**
    - Formal behaviour of an application expressed in class invariants, method pre- and post-conditions
  - **Allows for program verification using**
    - JML Runtime checker (testing)
    - ESC/Java (semi-formal verification)
    - LOOP (formal verification)

# Program verification

- When doing program verification, one *proves* properties about programs
- Programs may terminate normally or abruptly (or not at all)
- For example, a wallet application maintains a balance
- A property for this wallet application may be:  
*the balance is never negative*
- This property should (at least) hold before and after processing of the incoming request

# Formal verification

- For a correct application, one needs to verify that it behaves properly
- However, only proving properties about a single application is not always sufficient
- One needs to take into account the interaction of the application with its environment
- **Examples:**
  - Transaction mechanism
  - Applet firewall
  - Persistent and transient memory models

# Transactions

- The JVM guarantees that assignments to fields are atomic
- However, multiple consecutive assignments are *not* atomic
- Java Card introduces *transactions*
- A transaction makes a part of the program atomic
- **Example: implementation of a `changePIN` method.**
  - Assume a new PIN value of four digits;
  - Ensure that after applying this method that:
    - the PIN value has changed to its new value,
    - or it has not changed at all.

## Example – Transactions

```
void changePIN (byte[] newpin) {  
    pin[0] = newpin[0];  
    pin[1] = newpin[1];  
    pin[2] = newpin[2];  
    pin[3] = newpin[3];  
}
```

**What is the PIN value if the power is lost during the update?**

**Solution: use Java Card's transaction mechanism**

## Example – Transactions (2)

```
void changePIN (byte[] newpin) {  
    JCSystem.beginTransaction();  
    pin[0] = newpin[0];  
    pin[1] = newpin[1];  
    pin[2] = newpin[2];  
    pin[3] = newpin[3];  
    JCSystem.commitTransaction();  
}
```

The assignments are now conditional:

if the transaction is completed, the PIN value is  
committed to memory,

otherwise, the PIN value is reverted to its old value

## Example – Transactions (3)

- **Dedicated (native) methods to copy arrays:**
  - `arrayCopy (src, src_offset, dst, dst_offset, length)`  
Makes use of the transaction mechanism (slow)
  - `arrayCopyNonAtomic (src, src_offset, dst, dst_offset, length)`  
Does not make use of the transaction mechanism (fast)
- **Other implementations of the changePIN method:**

```
void changePIN (byte[] newpin) {  
    arrayCopy (newpin, (short)0, pin, (short)0, (short)4);  
}
```

```
void changePIN (byte[] newpin) {  
    JCSystem.beginTransaction();  
    arrayCopyNonAtomic (newpin, (short)0, pin, (short)0, (short)4);  
    JCSystem.commitTransaction();  
}
```

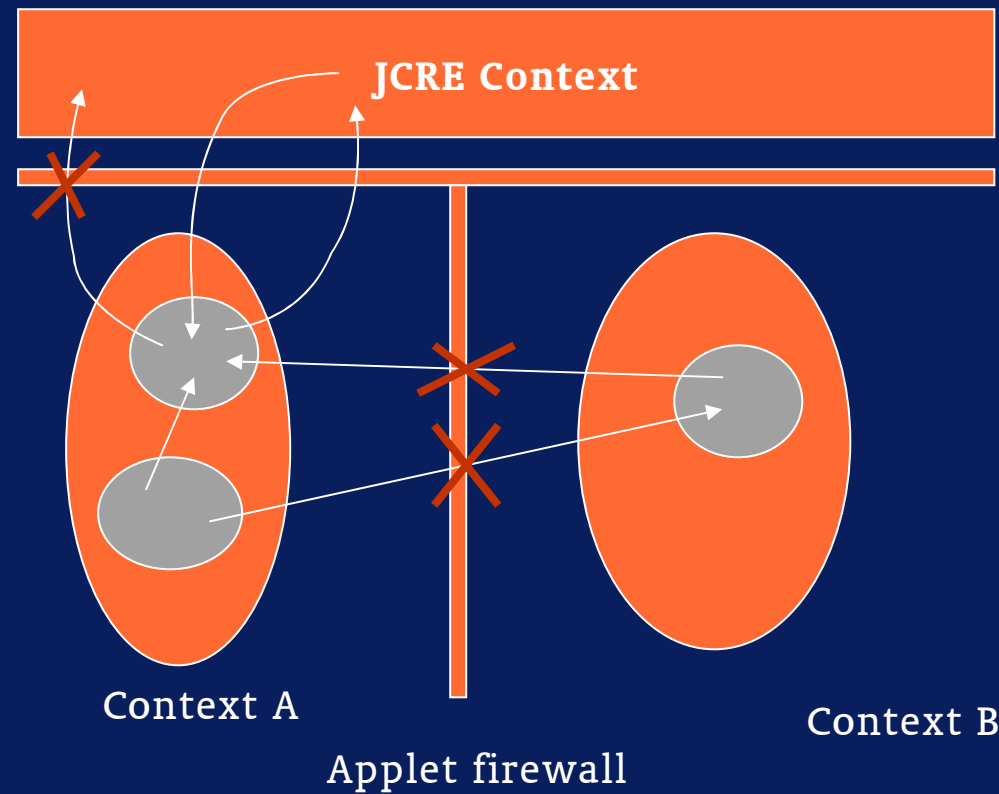
## Transactions – What's missing?

- Extra dimension to program verification: *power loss*
- And thus making program verification a lot more difficult (or impossible?)
- Transaction mechanism may reverse memory changes, but it is sometimes bypassed
- Again, another complication to show that a program is correct

# Applet firewall

- The *applet firewall* protects applet's data from unauthorised access
- Each applet is defined in a package – on the card: context
- JCRE (System) context is allowed to access all objects
- Objects in a context *A* are freely accessible by other objects in the same context *A*
- Objects in other contexts are not allowed to access objects that belong to context *A*

# Applet firewall



## Example – Applet firewall

```
public void process (APDU apdu) {  
    // store APDU buffer in local variable  
    byte[] buffer = apdu.getBuffer();  
  
    // further processing of incoming APDU  
    ...  
}
```

```
public void process (APDU apdu) {  
    // store APDU buffer in field  
    buffer = apdu.getBuffer();  
    // error!  Raises SecurityException  
    // further processing of incoming APDU  
    ...  
}
```

# Applet firewall – What's missing?

- Models should include a notion of contexts and ownership relations
- Each object has an owning context
- Models should include which objects may be referenced, and which may not

# Memory models

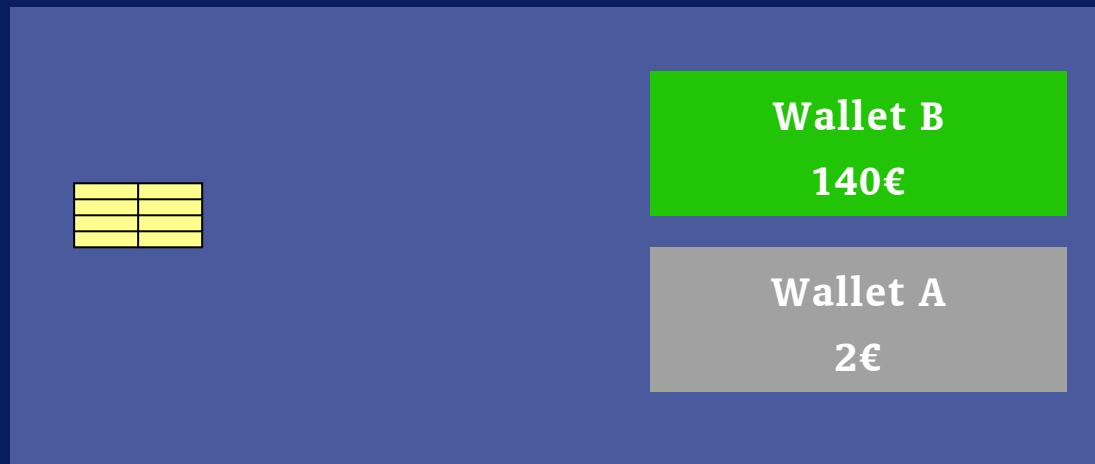
- Information might be sensitive to illegal disclosure, modification or use
- Java Card introduces different memory models
  - Persistent memory model (ROM, EEPROM)
  - Transient memory model (RAM)
- Different memories, are they used properly?

## Memory models (2)

- **Persistent memory (EEPROM):**
  - Secret key
  - Applications
  - Transaction logging
  - slow
- **Transient memory (RAM):**
  - Session keys (CAD session keys, or applet session keys)
  - Intermediate results (scratch pad)
  - Two types: one gets cleared after reset, the other after deselect
  - fast

## Example – Memory models

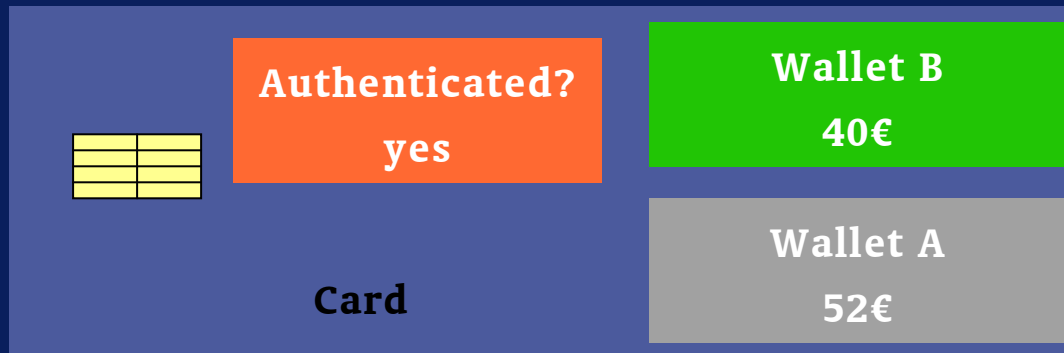
- Card with two Wallets A and B
- Authentication field for changing balance on the Wallet (used for increasing and decreasing)



## Example – Memory models (2)

- Step 1: Authenticate
- Step 2: Select Wallet A
- Step 3: Increase with 50€
- Step 4: Select Wallet B
- Step 5: Decrease with 100€

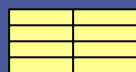
**No  
authentication  
for Wallet B!**



## Example – Memory models (3)

- Step 1: Select Wallet A
- Step 2: Authenticate Wallet A
- Step 3: Increase with 50€
- Step 4: Select Wallet B
- Step 5: Authenticate Wallet B
- Step 6: Decrease with 100€

**Wallet A still  
authenticated!**



Card

Auth?

yes

Wallet B

40€

Auth?

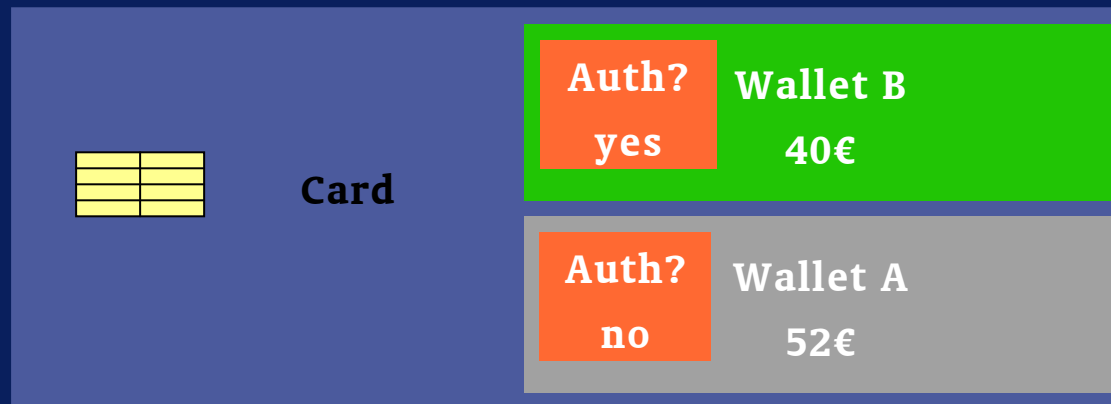
yes

Wallet A

52€

## Example – Memory models (4)

- Step 1: Select Wallet A
- Step 2: Authenticate Wallet A
- Step 3: Increase with 50€
- Step 4: Select Wallet B
- Step 5: Authenticate Wallet B
- Step 6: Decrease with 100€



## Memory models – What's missing?

- For correctness of a single Wallet, it does not matter where Authenticated field is stored
- For security of the applications on the card it does!
- Distinguish between persistent data, card session data and applet session data (persistent and transient)
- Distinguish between high and low sensitive data; which data must remain private to the applet?

## Conclusions

- TNO EIB tests security features of Java Card
- TNO ITSEF BV might move towards the formal parts of CC evaluations
- Requires formal specification and development of products
- For this purpose, formal models developed at universities need to be improved
- Suggestion for improvements of these models by examples

# Questions?

Joachim van den Berg

TNO EIB

E: [bergjo@tpd.tno.nl](mailto:bergjo@tpd.tno.nl)

T: +31 15 269 2979

