UNIVERSITY OF TRENTO - Italy

The Embeddable Security-by-Contract Verifier for Java Card

Olga Gadyatskaya
Joint work with F. Massacci, E. Lostal
University of Trento (Italy)

BYTECODE 2012, Tallinn, Estonia
31.03.2012

Multi-App Cards Story

First papers on multi-application smart cards appeared in 1999-2000

And research continued actively until 2003-2004

BUT

Nobody has seen these cards..

New NFC World I

[redacted]

And then NFC appeared

Now we have NFC-payments, NFC-ticketing, NFC-discounts

New NFC World II

[redacted]

Sensitive apps need a secure element

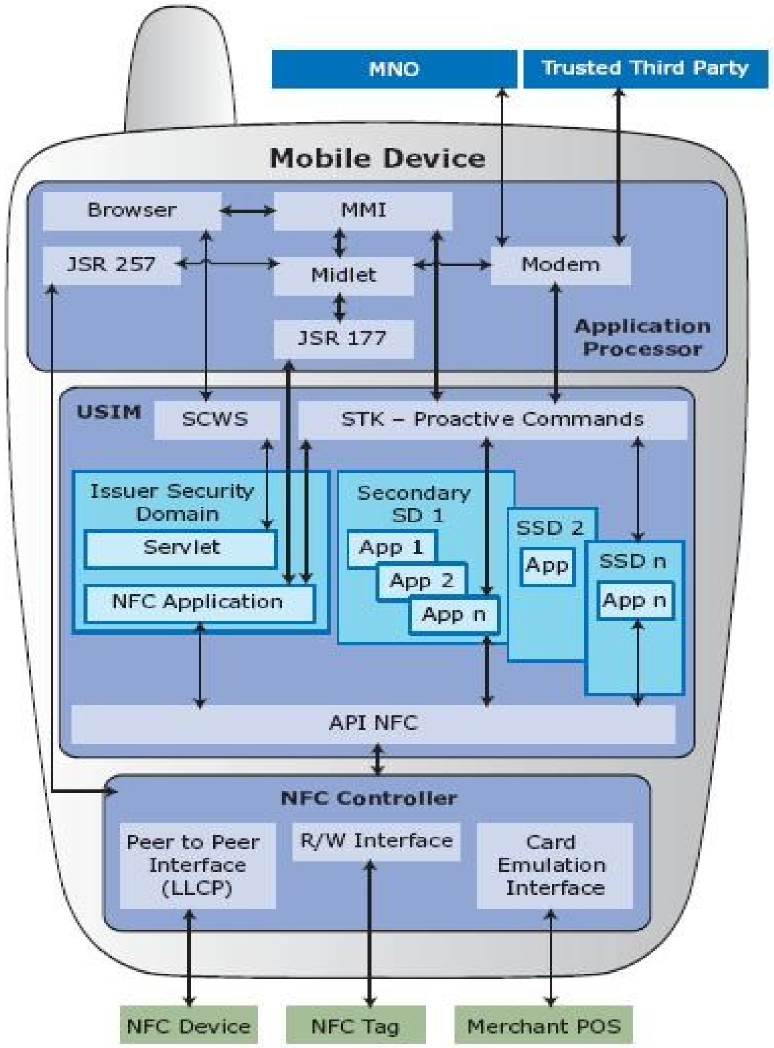
IDEA

Use the smart card as the secure element!

Already deployed infrastructure

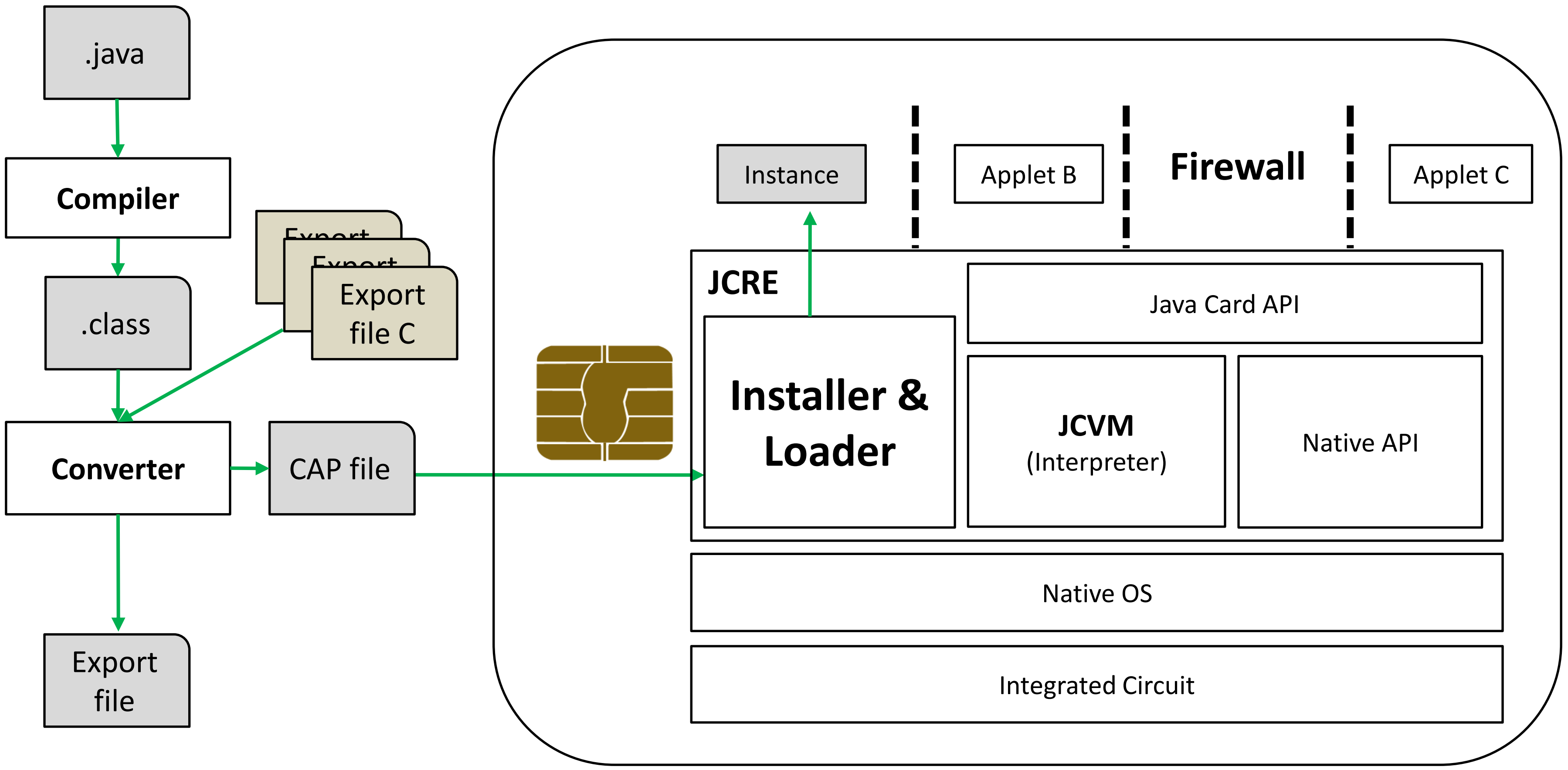
BUT

Application interactions need to be controlled



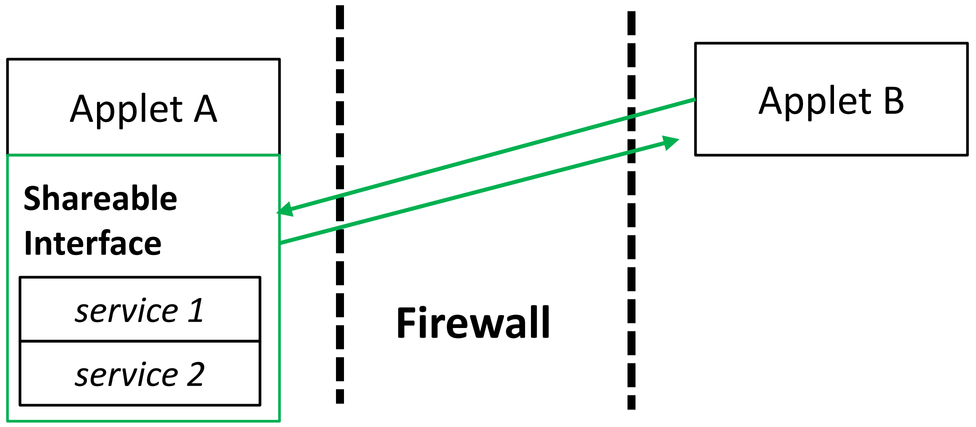
The Platform

[redacted]



How does JC really work?

Run-time



Access control is embedded into functional code

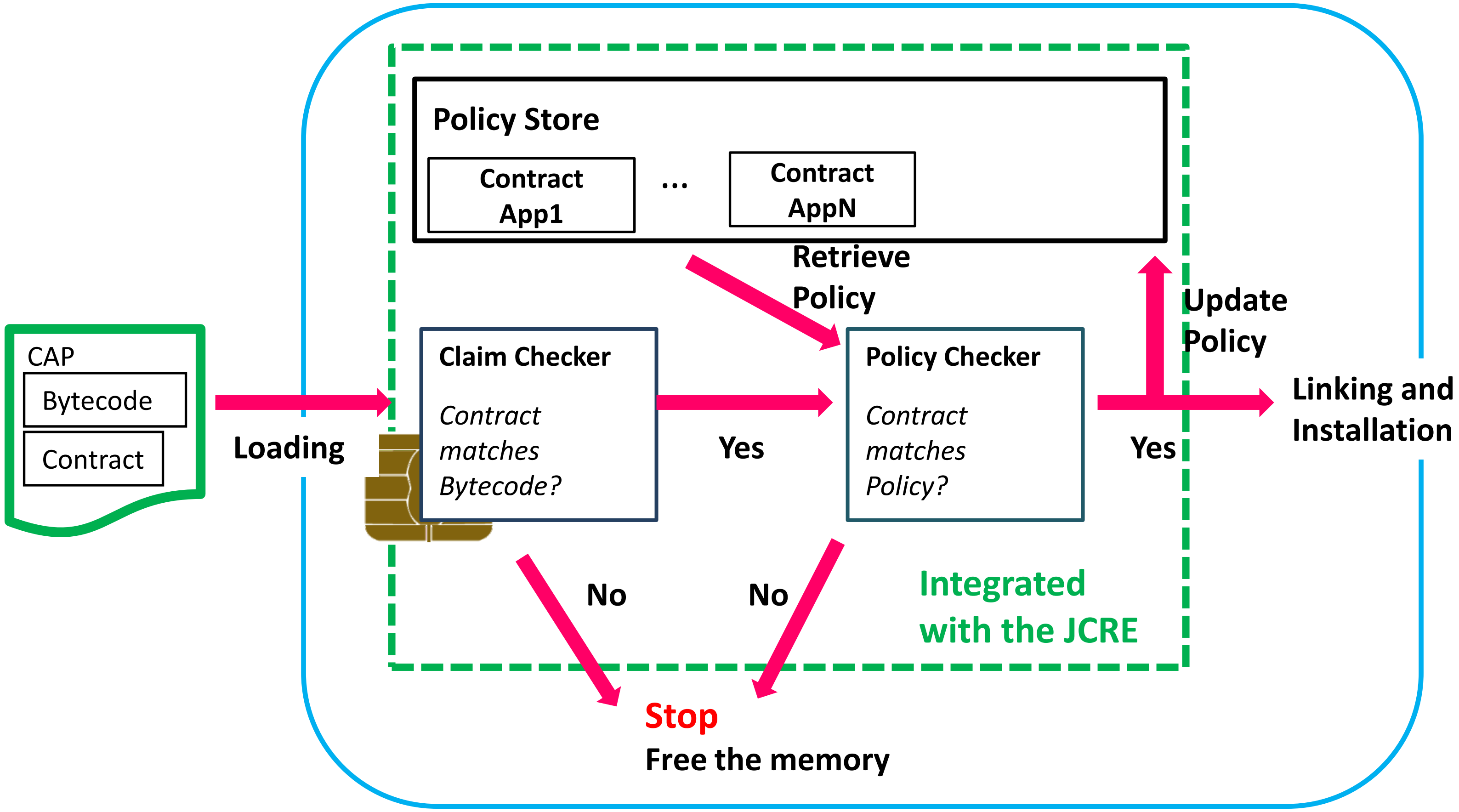
- **Technical Consequence 1 → If A checks who calls it, the access control policy cannot be updated unless the code is updated**
  - **sometimes code updates are not even possible**
- **Technical Consequence 2 → If A does not check, then everybody can use it**

Design Targets

- Same security of interacting smart cards with access control embedded in the code
  - Apps can arbitrarily restrict who calls their services
- Enabling security policy updates
  - without code update
- On a challenging hardware platform
  - RAM footprint <1KB, NVM footprint <20KB
  - Small time overhead
- No changes to external loading protocols

Loading time verification with the Security-by-Contract scheme

[redacted]



[redacted]

Contract I

- **Apps come equipped with a contract**
  - **Claims**
    - I may provide these shareable interfaces with these services
    - I may call those methods from those interfaces
  - **Security Rules**
    - This service can only be called by this application
  - **Functional Rules**
    - I need these services from those applications
- **When new app arrives platform will check**
  - contract complies with bytecode
  - contract acceptable to other applets

Contract II

Contract of an applet

AppClaim

Provided services

<Interface token, method token>

Called services

<Provider application AID, Interface token, method token>

AppPolicy

Security rules

<Interface token, method token, Authorized application AID>

Functional rules

<Provider application AID, Interface token, method token>

[redacted]

How do we get the tokens?

Source code of an applet

```
public interface CoopPointsInterface
extends Shareable {
    byte sharePoints (byte points);}

public class CoopPointsClass
implements CoopPointsInterface {
  public byte sharePoints(byte
points) {
    return (byte) (points + 2);}}

private void askForCharge() {
    final AID Purse_AID =
JCSystem.lookupAID(PurseAID,(short)0,
(byte)PurseAID.length);

    CreditObject = (CreditInterface)
(JCSystem.getAppletShareableInterface
Object(Purse_AID, CreditDetails));

points = CreditObject.charge(points);
}
```

// Actual service invocation

Export file of the same applet

```
export_classes {
  class_info {        //Shareable interface token
    token 0
    name_index 3 //coop/CoopPointsInterface
    export_methods_count  1
    methods {
      method_info {       //shared method token
          token    0
          name_index  0 // sharePoints
```

CAP file of the same applet

```
package_info[2] { …
   AID_length   6
   AID (1,2,3,4,5,0)  }
```

Import component

```
constant_pool[18] { ...
     External PackageToken: 2,
     ClassToken: 0
     ...}
```

//Called interface token

Constant Pool component

//Bytecodes of **askForCharge()**

```
...
getstatic_b 4
invokeinterface 2, 18, 0
putstatic_b 4
return
```

//Called method token

Method component

Security Policy on the card

[redacted]

We can have arbitrary number of applets mentioned in the policy

Policy on the card

Small size and (frequent) efficient operations

`Policy` (fixed size)

All loaded contracts in an internal bit-arrays format

`MayCall`

Possible future authorizations for applets not yet on the card

Big size and (rare) slow operations

`Mapping`

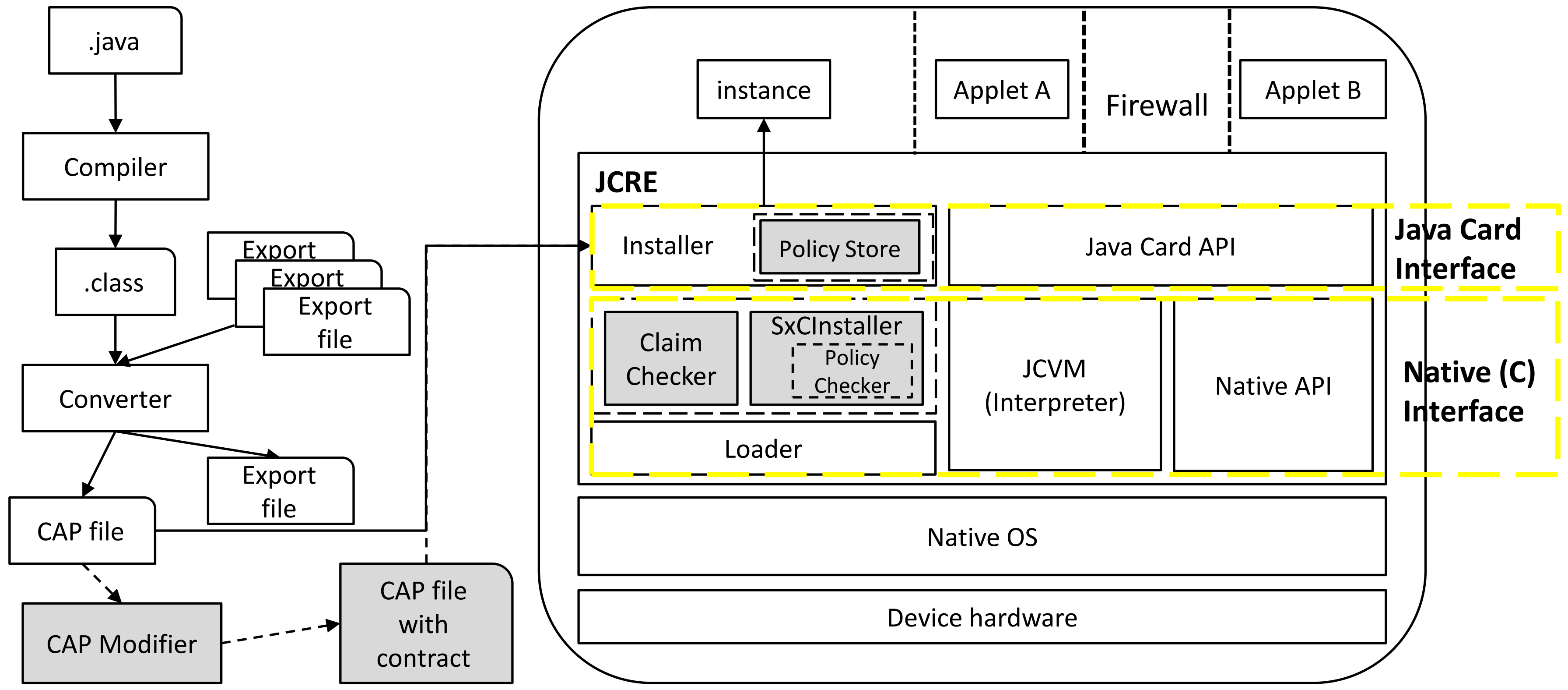
Maintains correspondence between on-card IDs and AIDs

`WishList`

Called services from ap
not yet on the card

Big size and (rare) slow operations

SxC Architecture

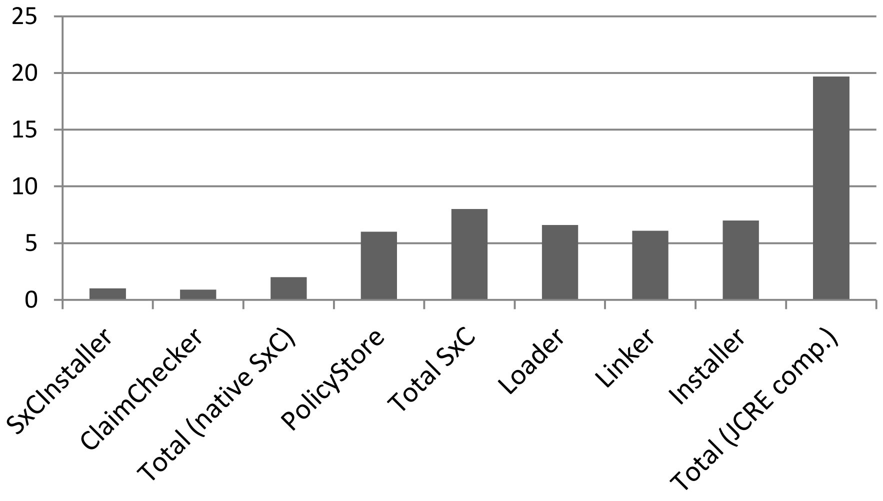


The SxC deployment process does not modify the standard Java Card tools

[redacted]

It really works on a card

- **Developer’s Version (run on PC Win32 simulator)**
  - ClaimChecker →10KB
  - PolicyChecker+SxCInstaller →10KB
  - PolicyStore → 6KB
- **JavaCard’s version (on Gemalto’s card)**
  - ClaimChecker → 1KB
  - PolicyChecker +SxCInstaller→ 0.9KB
  - Total SxC components → 8KB of NVM
- **To put numbers in perspective**
  - Installer → 6KB
  - JCRE (Loader+Linker+Installer) → 20KB



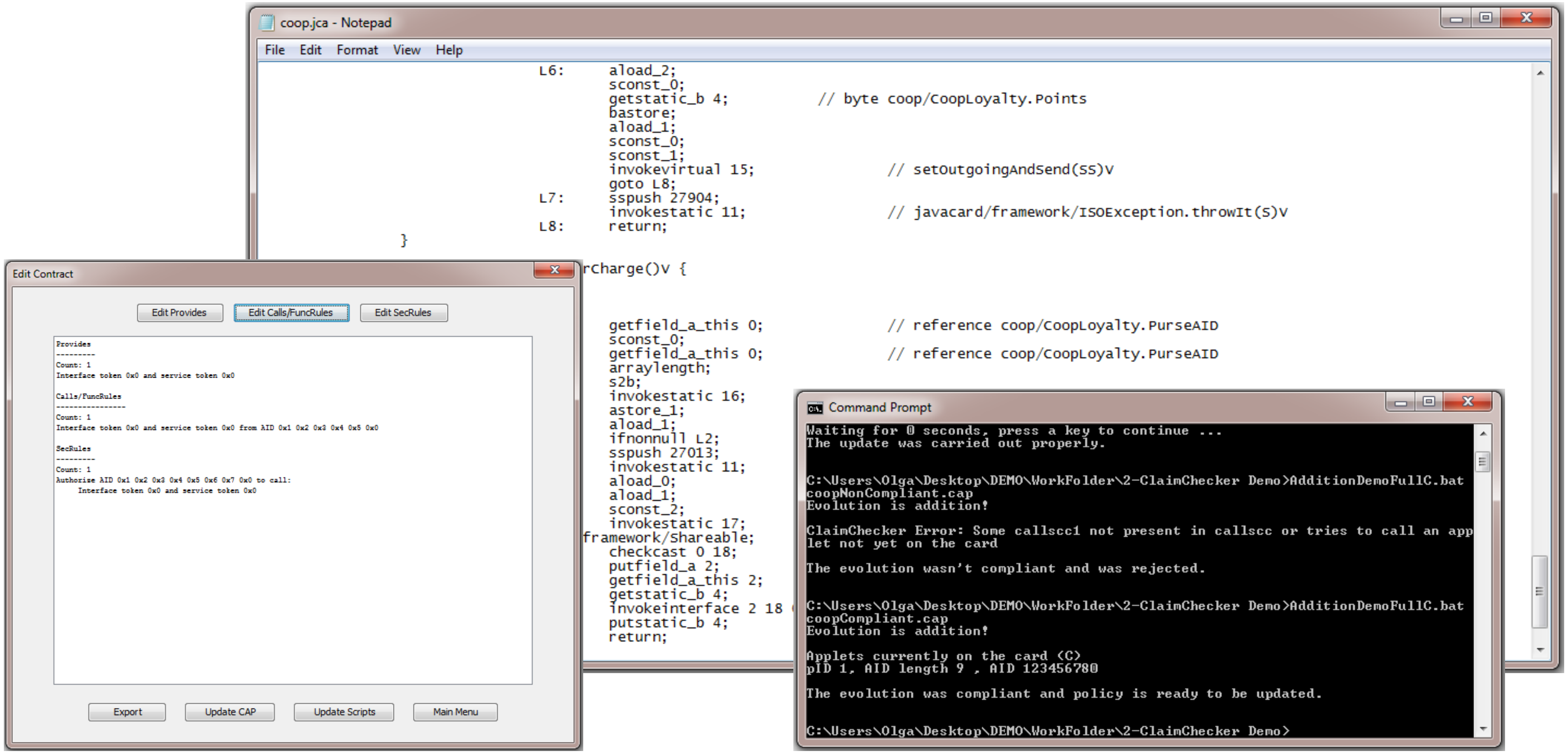
On-card components

[redacted]

Works on real applets

Quick overview of the real applets used for testing:

- **Electronic purse application from Gemalto:** 4.7KB CAP file, 16 methods
- **Ticketing app from Gemalto** : 3KB CAP file, 7methods
- **Belgian electronic identity app:** 11.2 KB CAP file, 81 method
- **Another electronic purse app from Gemalto:** 4.5 KB CAP file, 18 methods



```
                          L6:     aload_2;
                                  sconst_0;
                                  getstatic_b 4;          // byte coop/CoopLoyalty.Points
                                  bastore;
                                  aload_1;
                                  sconst_0;
                                  sconst_1;
                                  invokevirtual 15;               // setOutgoingAndSend(SS)V
                                  goto L8;
                          L7:     sspush 27904;
                                  invokestatic 11;                // javacard/framework/ISOException.throwIt(S)V
                          L8:     return;
                 }

rCharge()V {


                                  getfield_a_this 0;              // reference coop/CoopLoyalty.PurseAID
                                  sconst_0;
                                  getfield_a_this 0;              // reference coop/CoopLoyalty.PurseAID
                                  arraylength;
                                  s2b;
                                  invokestatic 16;
                                  astore_1;
                                  aload_1;
                                  ifnonnull L2;
                                  sspush 27013;
                                  invokestatic 11;
                                  aload_0;
                                  aload_1;
                                  sconst_2;
                                  invokestatic 17;
framework/Shareable;
                                  checkcast 0 18;
                                  putfield_a 2;
                                  getfield_a_this 2;
                                  getstatic_b 4;
                                  invokeinterface 2 18 (
                                  putstatic_b 4;
                                  return;
```

```
Provides
---------
Count: 1
Interface token 0x0 and service token 0x0

Calls/FuncRules
----------------
Count: 1
Interface token 0x0 and service token 0x0 from AID 0x1 0x2 0x3 0x4 0x5 0x0

SecRules
---------
Count: 1
Authorise AID 0x1 0x2 0x3 0x4 0x5 0x6 0x7 0x0 to call:
     Interface token 0x0 and service token 0x0
```

```
Waiting for 0 seconds, press a key to continue ...
The update was carried out properly.

C:\Users\Olga\Desktop\DEMO\WorkFolder\2-ClaimChecker Demo>AdditionDemoFullC.bat
coopNonCompliant.cap
Evolution is addition!

ClaimChecker Error: Some callscc1 not present in callscc or tries to call an app
let not yet on the card

The evolution wasn't compliant and was rejected.


C:\Users\Olga\Desktop\DEMO\WorkFolder\2-ClaimChecker Demo>AdditionDemoFullC.bat
coopCompliant.cap
Evolution is addition!

Applets currently on the card (C)
pID 1, AID length 9 , AID 123456780

The evolution was compliant and policy is ready to be updated.


C:\Users\Olga\Desktop\DEMO\WorkFolder\2-ClaimChecker Demo>
```

DEMO?

Just ask me at the coffee break!

Conclusions

[redacted]

- **The SxC embedded verifier performs the loading time application certification**
  - Ensuring that an applet is accepted only if it respects policies of the applets already on the card
- **The security code is separated from the functional code**
- **The policy management is centralized**
  - Important for the platform owner
- **It really works on a smart card with real industrial applets**
  - The framework is a non-invasive addition to the standard Java Card deployment process

UNIVERSITY
OF TRENTO - Italy

secure CHANGE

NESSoS

Questions?

olga.gadyatskaya@unitn.it

more info at
www.disi.unitn.it/~gadyatskaya/sxc.html