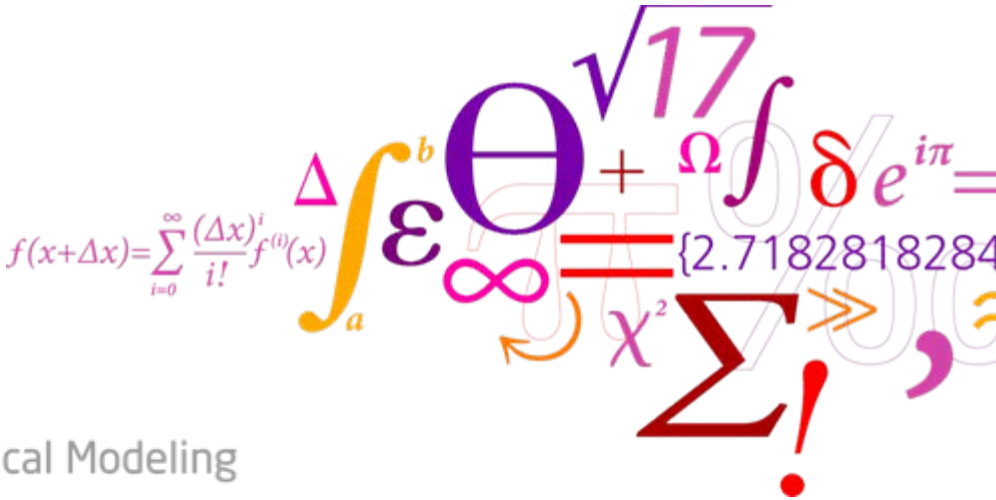
Master Thesis

Securing Multi-Application Smart Cards by Security-by-Contract

Author: Eduardo Lostal Lanza

Supervisor: Nicola Dragoni

Kongens Lyngby, 2010



DTU Informatics
Department of Informatics and Mathematical Modeling

Structure of this Presentation

- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- Extending SxC Off-Card
- Implementation
- Evaluation

Solution

- Conclusion

Conclusion

- **Smart Card Technology** — Background
- Multi-Application Smart Cards — Background
- Security-by-Contract — Problem
- Extending SxC Off-Card — Solution
- Implementation — Solution
- Evaluation — Solution
- Conclusion — Conclusion

Smart Card

- Device able to:
  - o Store data
  - o Carry out functions
  - o Interact with a external reader
- Why so widespread?
  - o Easiness of use
  - o Portability
  - o Cheap price
- Tamper-Resistant and Security Features

Secure and Trusted device

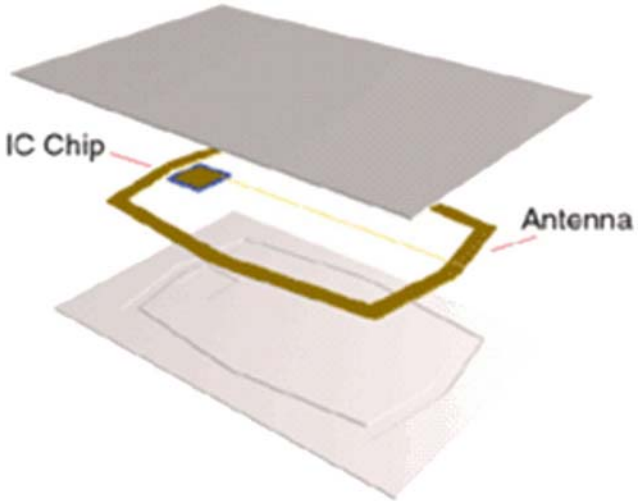
High security at a reasonable cost!

Contact Smart Cards

- Inserted into the CAD to be powered on
- Gold-Plated pads in physical contact with reader
- Drawback ➡ Fail because of being worn out

Contactless Smart Cards

- Antenna glued inside the plastic card
- Communication and power supply over-the-air
- More reliable, but more expensive

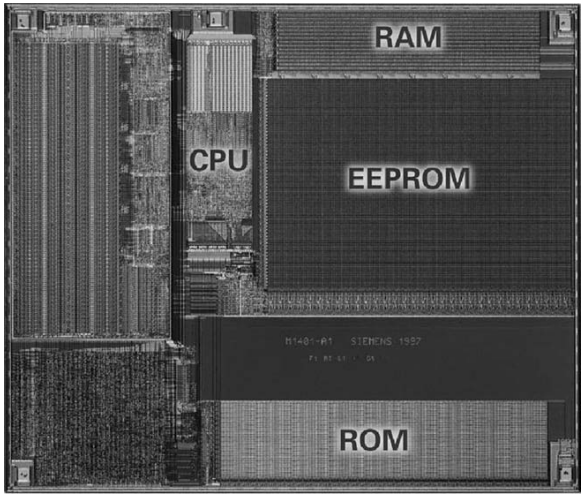


Dual Interface Cards

- Used to work in more than one application

Microprocessor

- Not specifically built: Money and security
- Older CISC, nowadays RISC



Memory

- RAM: Volatile,
- EEPROM: Persistent, erasable
- ROM: Persistent, non-erasable

Coprocessors

ROM → EEPROM → RAM

- Carrying out particular tasks
- Commonly used for: o For cryptography algorithms
  o Random-Number generator


Grid computing
Banking
Telecommunication
Authentication
Pocket-gamming
Loyalty applications
E-services
Health care
Identification
Audiovisual industry
Transport
Access control
MasterCard
VISA
$10
Phone Card
TRAVEL
carte d'assurance maladie
vitale
ORF DIGITAL
oyster
SPECIMEN
ELECTRONIC PASSPORT

Contact Smart cards -> Physical attacks

- Invasive attacks: Probe on bus lines and EEPROM, reverse engineering
- Side-Channel Attacks: Electromagnetic, power and timing analysis, etc.

- Solutions: Scrambling, metal shield, glue and obfuscated logic, ANG

Contactless Smart Cards

- Eavesdropping, denial of service, radio frequency analysis

- Solutions: Cryptography, current stabilizer

Anomaly Monitors -> Voltage, frequency, temperature, etc.

Software Attacks -> Communication, verification of the bytecode

- Smart Card Technology
- **Multi-Application Smart Cards**
- Security-by-Contract
- Extending SxC Off-Card
- Implementation
- Evaluation
- Conclusion

Background

Problem

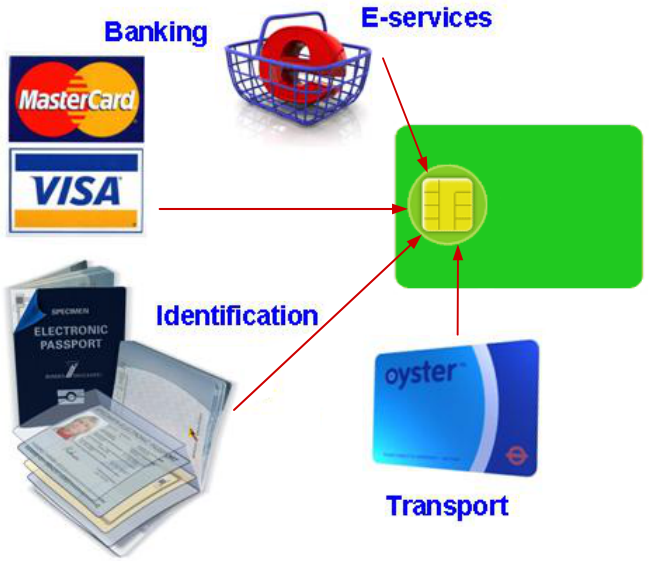
Solution

Conclusion

- Too many cards in our wallets → Users want to reduce them
- Issuers want to decrease → Time-to-market
  Development, infrastructure and deployment costs or to update applications after card issuance

Dynamic load post-issuance!

Flexible, open and secure platform to load dynamically applications after card issuance



Global Platform

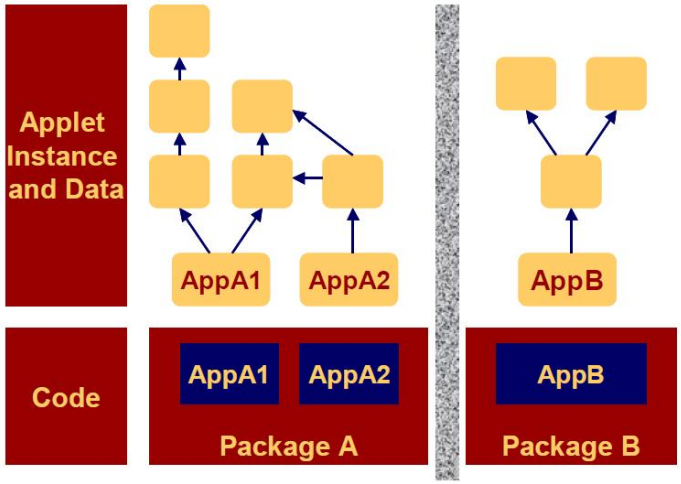
- Set of specifications to create a standard card management
- Security Domains

MULTOS

- Virtual machine, STEP, may program with several languages, MEL bytecodes

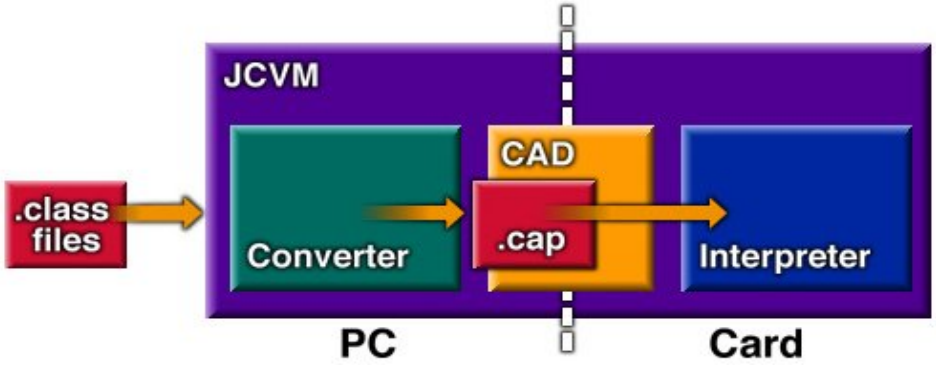
Java Card

- Subset Java
- Object oriented, interoperable, portable
- Context isolation → JCRE
- Enforced by firewall

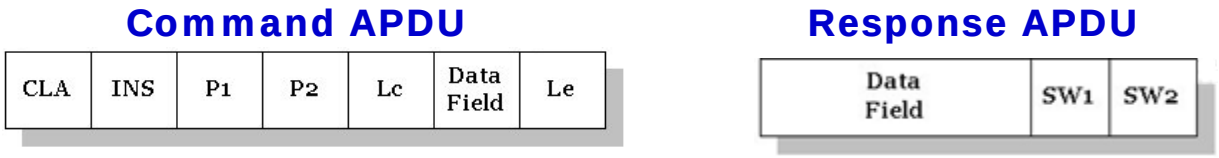


Java Card

- Shareable Interface Objects to allow accessing through firewall
- Virtual Machine



- APDU



Open Multi-Application Smart Cards

- Open policy which allows anybody to load, update and remove any application on-card
- Risk: software to be installed might not be trustworthy
- Problem: interactions among applications on-card
- Firewall-SIO do not solve it
- Semantic of the modification is not checked
- Should verify the behavior of the application → Security-by-Contract

- Smart Card Technology
- Multi-Application Smart Cards

Background

- **Security-by-Contract**

Problem

- Extending SxC Off-Card
  - Implementation
    - Evaluation

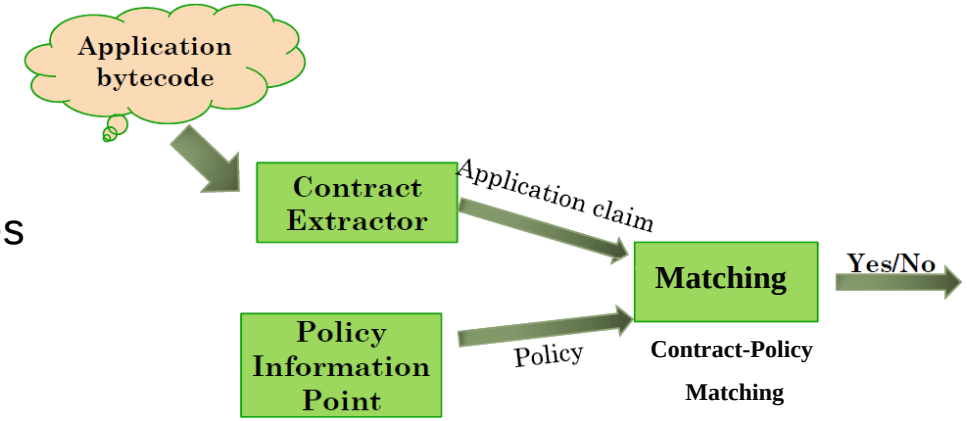
Solution

- Conclusion

Conclusion

Security-by-Contract

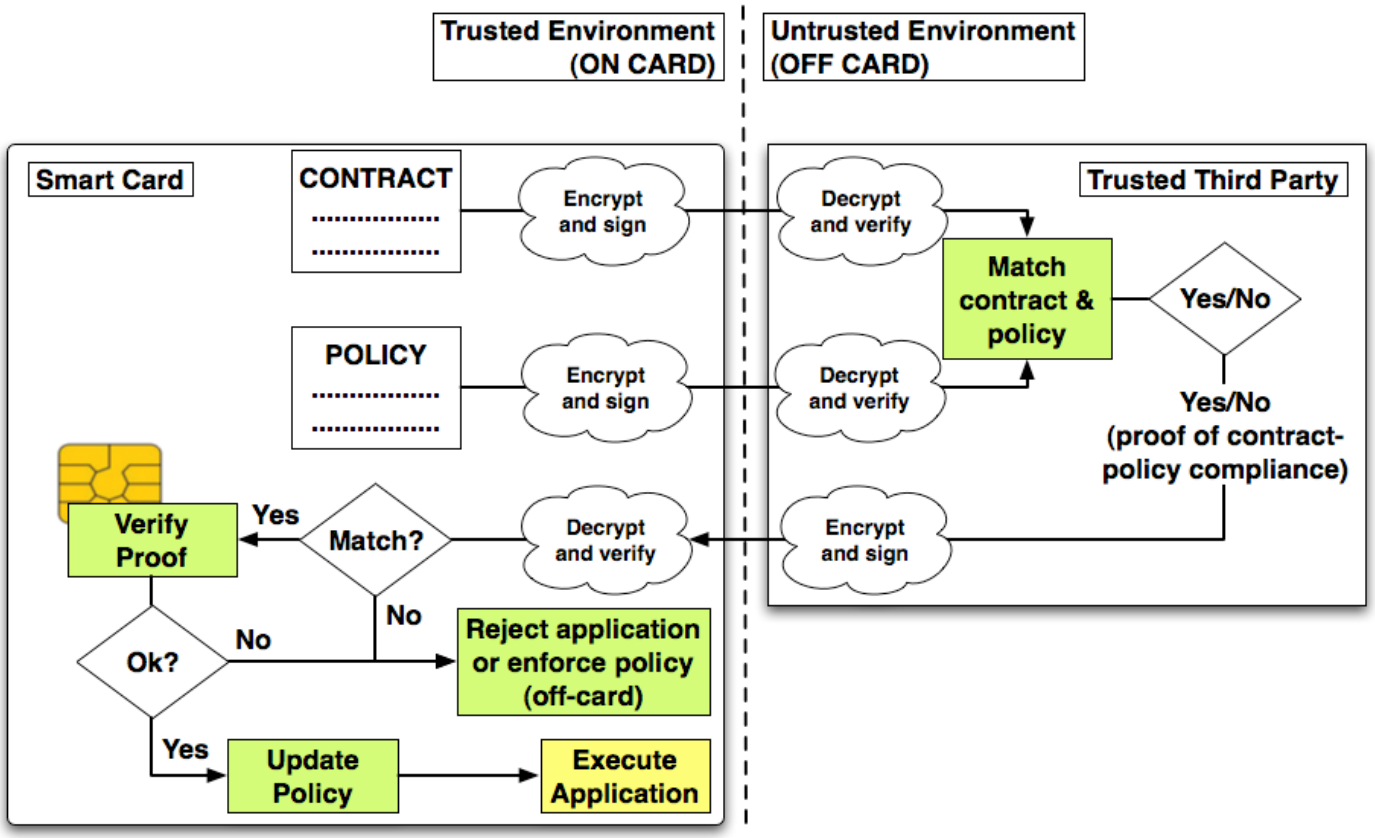
- Built upon the notion of MCC and successfully developed for mobile code
- Key point: Contract-Policy Matching (at load time)
- To solve { New application will not interact with forbidden ones on-card
  Dynamic change will not affect the correct work
- Dynamic change { addition
  update
  removal
  policy changes



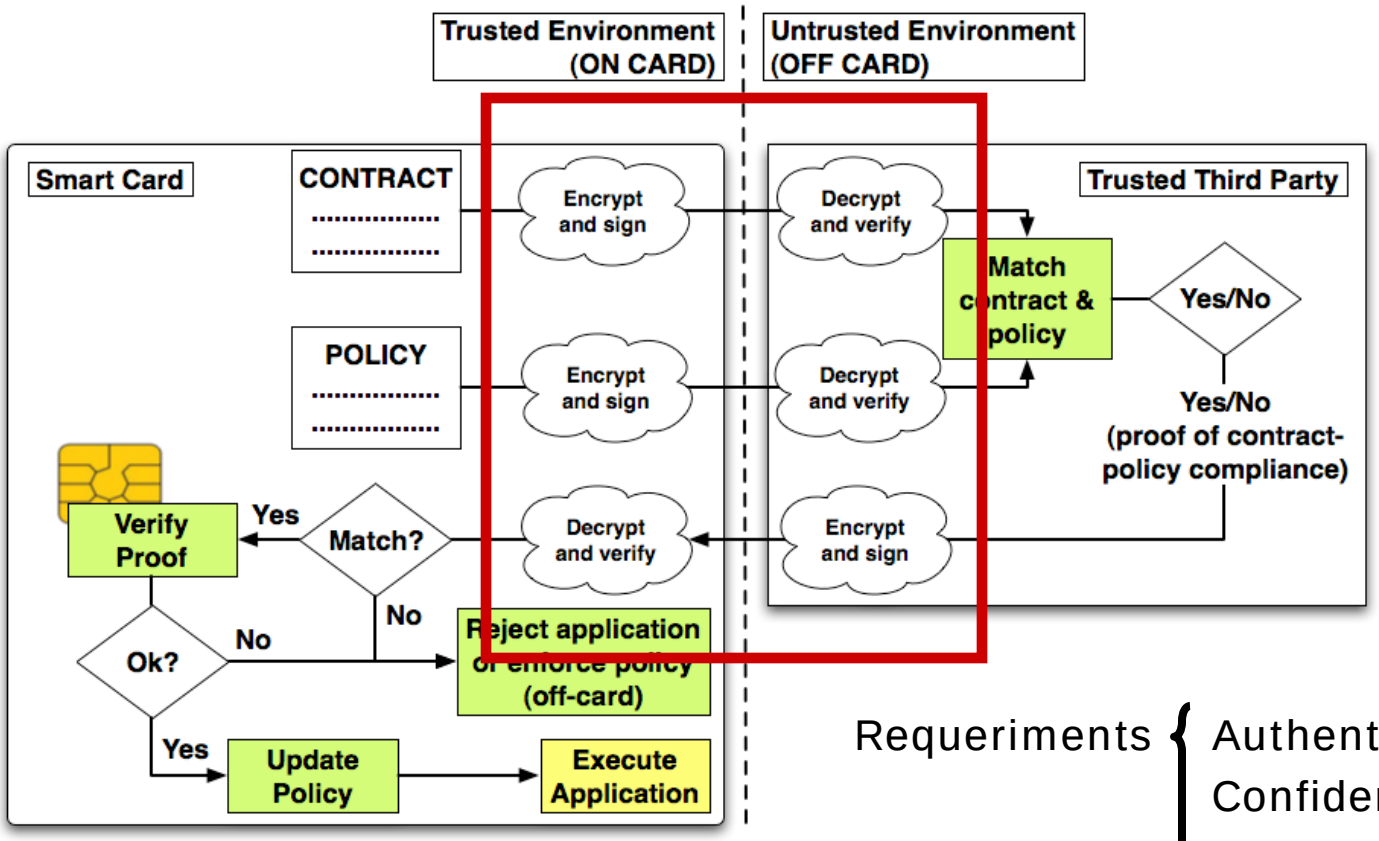
Hierarchy of Models

- Because of constraint resources
- Benefits in terms of computational efforts and expressivity according to level
- Levels
  - L0: Application as Services
  - L1: Allowed Control Flow
  - L2: Allowed and Desired Control Flow
  - L3: Full Information Flow
- Limitation of Level 0: Captures the possible information exchange, instead of the actual exchange and cannot capture the indirect communication between applications
- Why do not use Level 3 always (specification most complete)?

Problem: Securing Off-Card Contract-Policy Matching



Problem: Securing Off-Card Contract-Policy Matching



Requeriments { Authentication
Confidentiality
Integrity

- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- **Extending SxC Off-Card**
- Implementation
- Evaluation

Solution

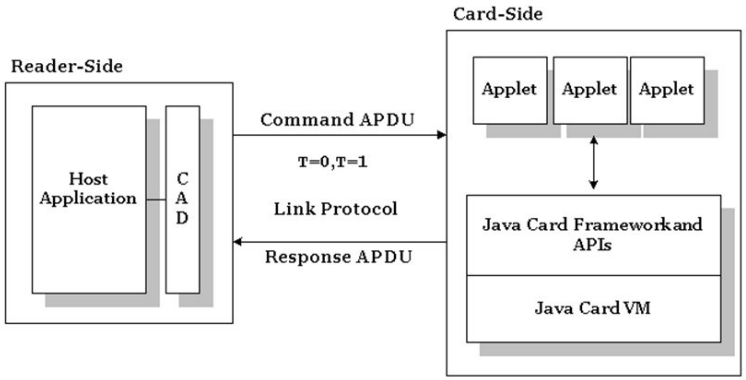
- Conclusion

Conclusion

- To fulfil requirements system is based on a PKI
- Identities are handled through certificates

} Initialization needed

- Initialization different to Installation
- One key pair for encryption and other one for signature
- Design focuses on Java Card, card works always as a server



Why confidentiality?

- Preventing information to be got from spoofing attacks
- Avoid an attacker to get information of the application's behavior
- Could be required by application issuers
- Commonly recommended, if constrained resources afford it
- Benefits bigger than disadvantages

Installation Phase

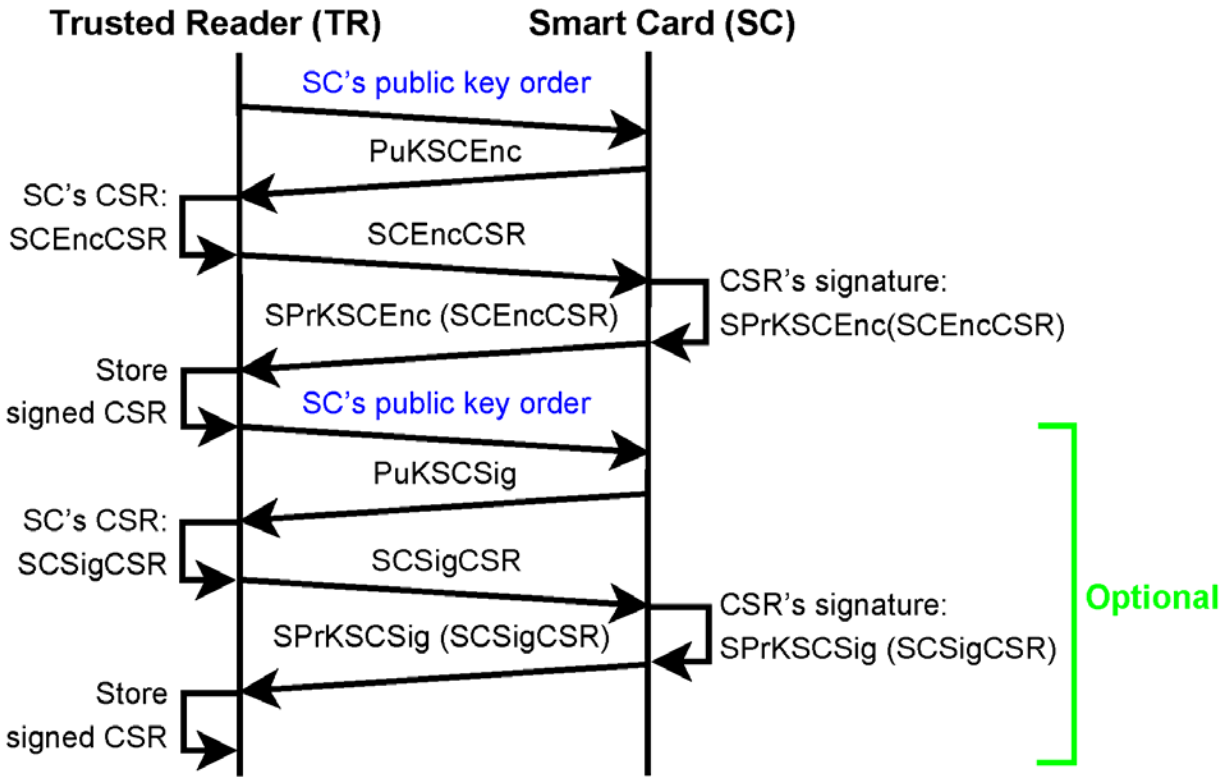
- After deploying application on the card
- Install the application on the card
- Generate the keys
- Security highlight of the system: Private keys do not leave the card simply because there is no reason to do that

- No one apart from the card can either get or use these keys

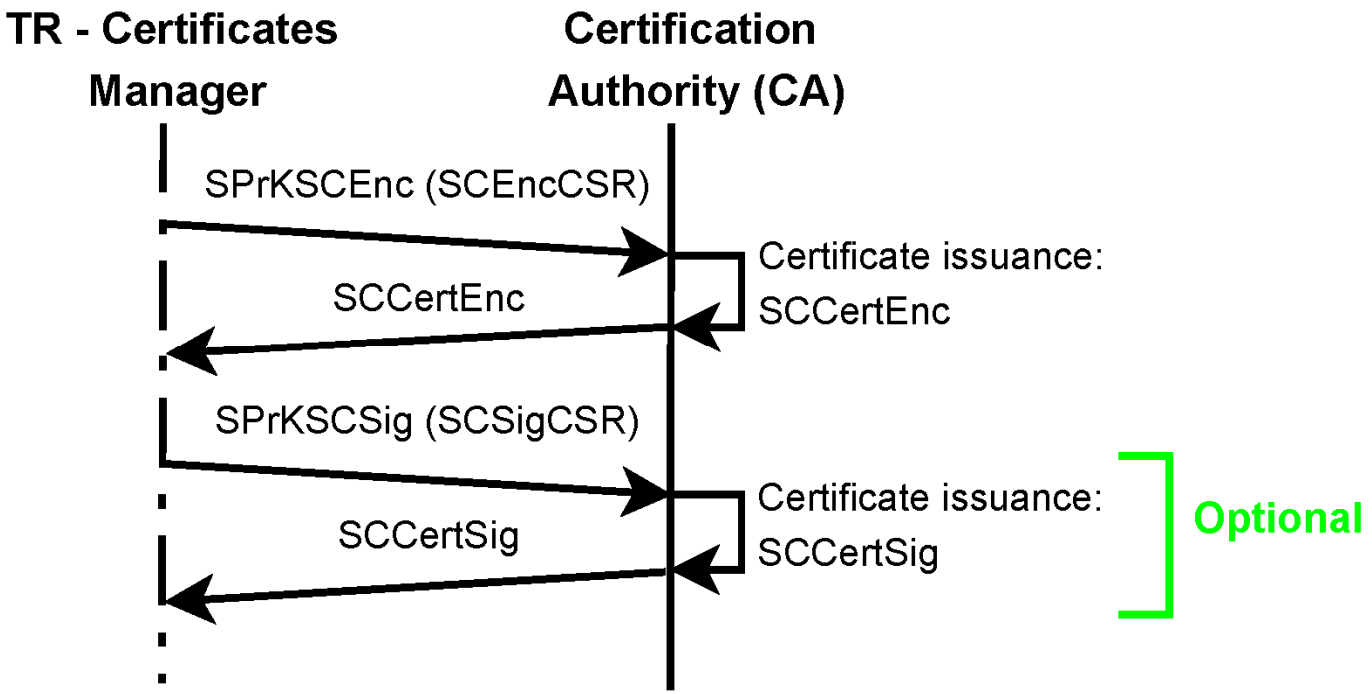
Initialization Phase

- Responsible for generating and storing the certificates and the policy
- Environment is completely trusted and secure
- Three stages:
  - Certificate Signing Requests Building
  - Certificates Issuing
  - Certificates and Policy Storage
- TR changes in the second stage to TR-Certificates Manager

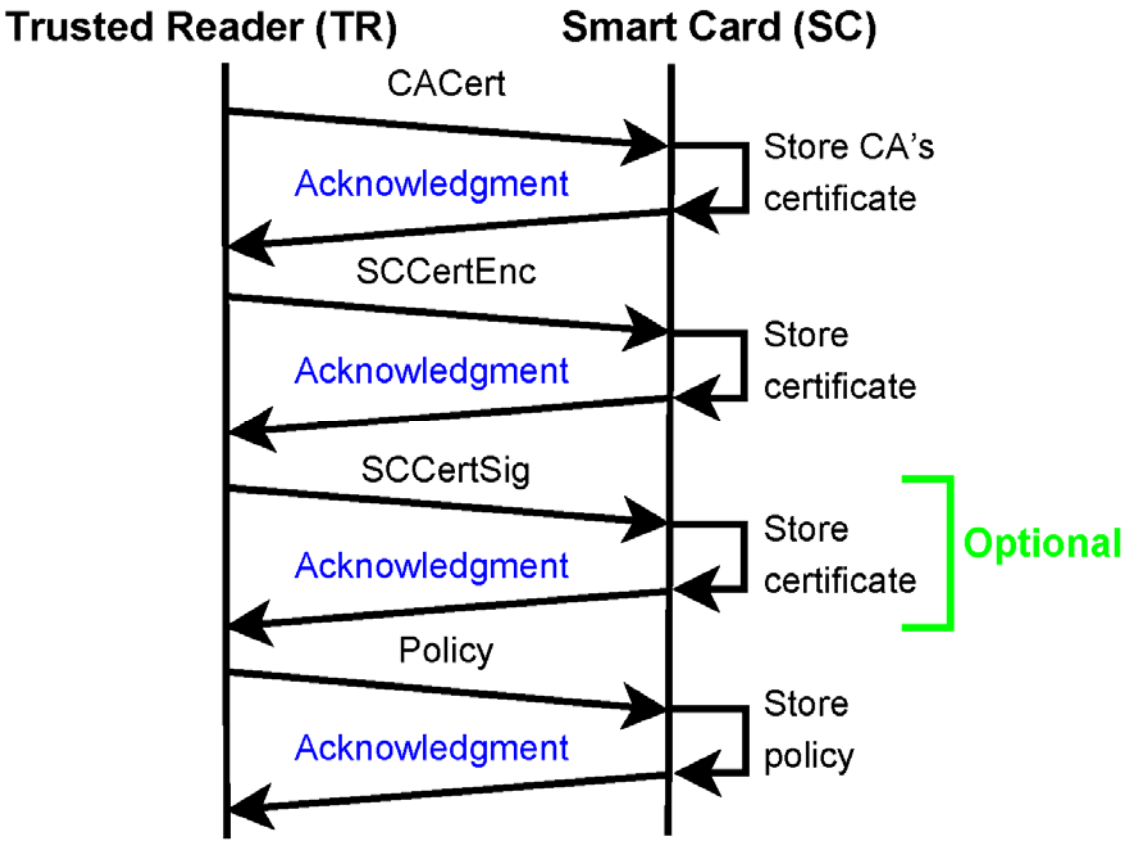
Initialization Phase: Certificate Signing Requests Building



Initialization Phase: Certificates Issuing

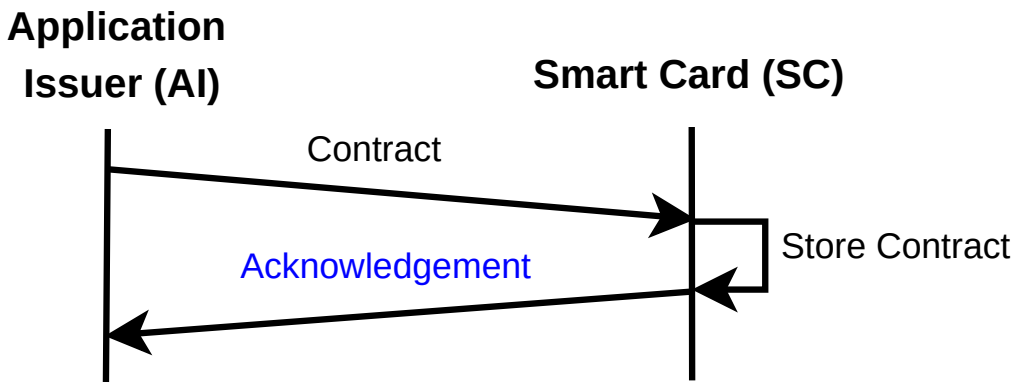


Initialization Phase: Certificates and Policy Storage

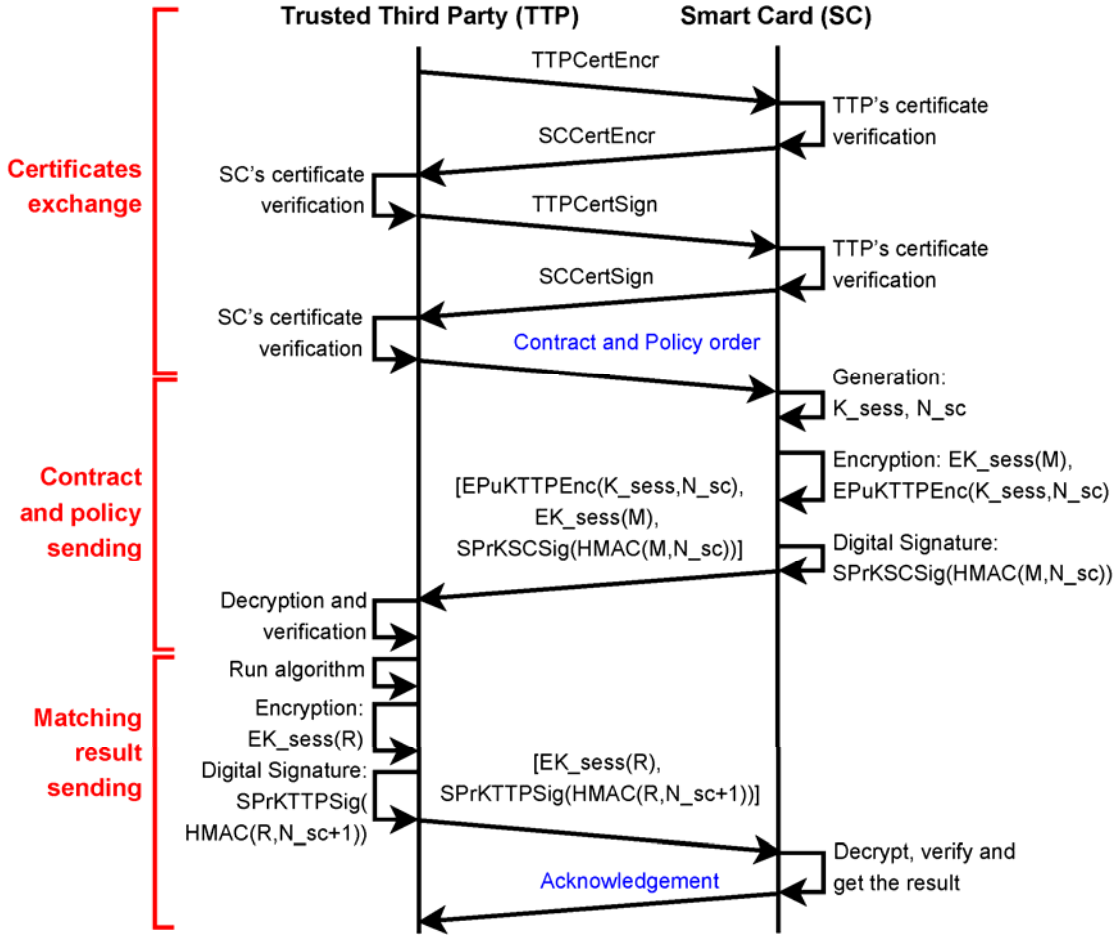


Contract Storage Phase

- Storage of the application's contract
- Carried out by AI



Contract-Policy Matching Phase



Contract-Policy Matching Phase – Why ...?

- **Nonce:**
  - Protecting card to ”Replay” attacks
  - Should be random to insure the freshness

- **Block symmetric Cryptography:**
  - Symmetric encryption provides higher speed (decryption is the same process; hence, it is also fast) than asymmetric for big amount of data
  - High security due to their no linearity

- **HMAC:**
  - Adds a shared key (salt) which increases the randomness
  - Assures that only who has the salt has been able to build the digest
  - Use of Nonce as salt assures freshness

- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- Extending SxC Off-Card
- **Implementation**
- Evaluation

Solution

- Conclusion

Conclusion

Off-Card (AI, TR, TTP) → Java

- Object-oriented, robust and simple
- Why? Multi-Platform and set of classes to deal with security issues
- Use of sun.* packages
- JDK version: jdk 1.6.0.18

Smart Card (SC) → Java Card 2.2.2

- APDU Extended length
- Why? Widespread and multi-application
- Why not Java Card 3?
  - Lack of real cards (only used by researchers)
  - Lack of wide information or manuals
  - Big overhead over the card response time
  - Expected cost

- IDE → Eclipse SDK 3.5.2
- Simulators
  - JCWDE
  - CREF
- Both provide only a subset of the cryptographic classes available at Java Card 2.2.2
- Restrictions in terms of
  - Key lengths
  - Cryptographic algorithms
    - ALG_NO_PAD RSA
    - Secure Random
- As a result
  - Prototype built with limitations
  - Another implementation prepared for a real card

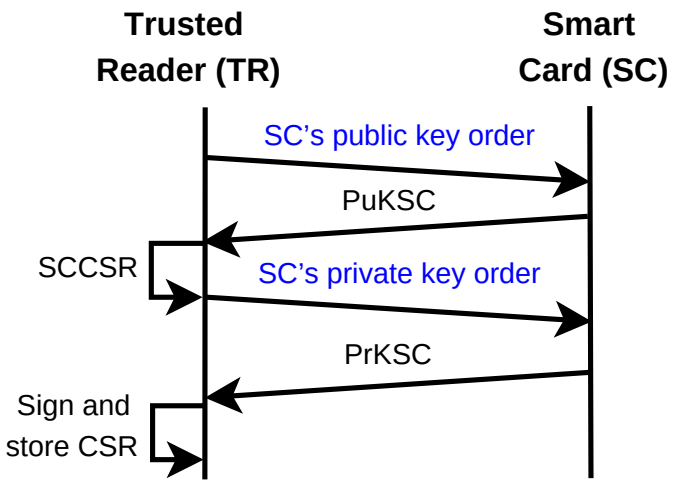
- X.509 Certificates
- Certificate generation by means of OpenSSL
- CA certificate is self-signed

Off-Card

- Stored in files, Base64 encoded and PEM extension
- X509Certificate data structure

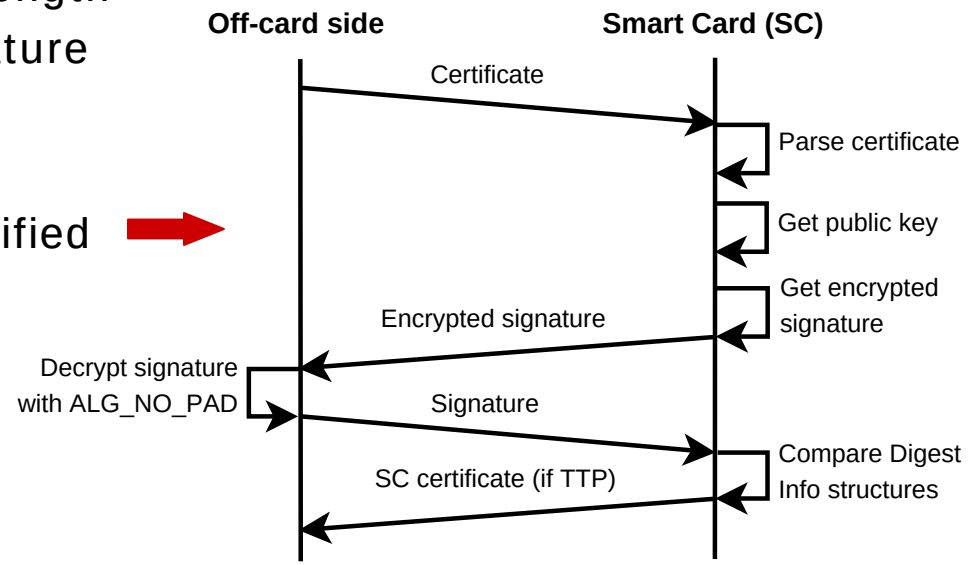
On-Card

- Byte arrays, DER encoded
- CSR generation modified



Parser on-card

- Verification checks
  - Compliance with DER
  - Compliance with ASN.1
  - Key algorithm
  - Signature algorithm
  - Key length
  - Signature
- Signature verification modified



Symmetric Block Cryptography

- AES algorithm with CBC mode and 128 bits of key
- Nonce is used as initialization vector

Asymmetric Cryptography

- RSA algorithm with padding according to PKCS#1 (v1.5)
- RSA key length 512 bits (for a real card 2048 bits)

Pseudo-Random Number Generator

- For a real card, Secure Random

- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- Extending SxC Off-Card
- Implementation
- **Evaluation**

Solution

- Conclusion

Conclusion

Memory Analysis

- By means of CREF simulator (data in bytes)
- Interesting data: EEPROM

Stage	Consumed before	Consumed after	Available before	Available after
Deployment	6994	12837	58510	52667
Installation	12837	14322	52667	51182
Initialization	14322	17919	51182	47585
Running	18298	18135	47206	47369

- Prototype needs a rough memory space of 11 KB → **Upper-limit!**
- Goal was to get a functional prototype, not optimal was expected
- Heaviest issue is the bytecode download

Memory Analysis

Considerations:

- Considering a size per application of 4-5 KB (usually smaller), still more than eleven applications can be stored
- Expected better on a real card (less bytecode)
- Memory hardware evolves, memory needs keeps the same
- An optimization would reduce the application weigth

System is considered suitable and could fit properly in a card

It does not reduce available memory considerably, allowing to store a large Amount of applications in a secure way

- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- Extending SxC Off-Card
- Implementation
- Evaluation

Solution

- **Conclusion**

Conclusion

- Problem with interactions among the applications stored on-card in open-multi-application smart cards
- Security-by-Contract framework proposed as a solution
- Constraint resources of smart cards → Hierarchy levels models
- Highest level of expressivity needs a computational effort bigger than the provided by smart cards
- Needed to outsource Contract-Policy Matching to a Trusted Third Party over an untrusted environment
- Communication must be secured
- Requirements of authentication, confidentiality and integrity

- Design to address the issue of making the communication to outsource the Contract-Policy Matching secure
- Implementation of a prototype as a proof-of-concept working on a simulator
- Implementation ready to work in a real card
- Test results which show the design is viable, suitable and define an upper-limit for the memory needs
- Preliminary results accepted to publication in Ubicomm 2010 (October - Florence, Italy)

- Exhaustive study of the code in order to carry out its optimization in terms of memory
  - Should cover: Use of garbage collector, instances' creation, reuse of variables, extend use of global variables and attributes, etc.
- Build the necessaries classes to avoid the use of sun.* packages (more stable version of the prototype)
- Multi-CA support
- Improve the parser: Validity period and revocation checks
- Smart Card certificate renewal
- Extend the project with Global Platform SD concerning to the installation of applications

Thank you for your attention!

- Exhaustive study of the code in order to carry out its optimization in terms of memory
  - Should cover: Use of garbage collector, instances' creation, reuse of variables, extend use of global variables and attributes, etc.
- Build the necessaries classes to avoid the use of sun.* packages (more stable version of the prototype)
- Multi-CA support
- Improve the parser: Validity period and revocation checks
- Smart Card certificate renewal
- Extend the project with Global Platform SD concerning to the installation of applications

Why to have a certificate for encryption if it is not used?

- It kept from previous designs where it was necessary (key session per communication)
- Although the design was modified and it was no longer needed, it was decided to hold it
- **Future uses**
- For instance, solution proposed to validity period issue

Why not to use a time stamp instead of Nonce?

- Card has not any clock since it has not a uninterrumped supply of power
- Requeriments for the ”stamp” { Randomness, Freshness
- Nonce accomplished these requirements perfectly

Why firewall is not enough?

- Firewall controls accesses to any method on the card
- It avoids any application being able to access any method of other application
- However, it may be necessary an application accessed to other
- SIO allows that
- To sum up, every application is not able to access any method on the card, but any method in the SIO

Why not to make the contract storage's communication secure?

- Time limitation of the project
- Contract storage was not really necessary for the problem addressed in the thesis, it was added in order to try to make the system closer to the reality
- If necessary, it could be done in a future, it should be applied in a similar way as it has been done for the contract and policy matching

Avoid the use of classes from sun.* packages

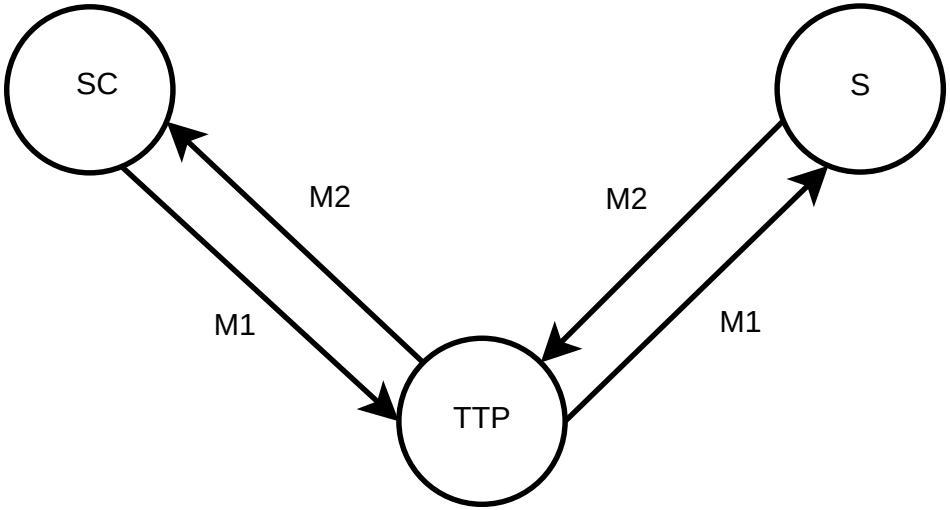
- More stable version of the system and independently of Sun's code updates (still working for the specified jdk version)
- Building classes to work with DER encoding
- Goal is to get a CSR which format was compliant with the standard
- Even it is possible to do on-card
- Initialization phase would be less complicated and faster (two communications less), but more space on the memory

Multi-CA Support

- In real world, card should store some CA's certicates in order to be able to verify certicates signed from distinct CAs.
- Think about how to store the certificate { constrained memory resources; inefficient to parse it every time
- Notice that Java Card does not allow to create any new data structure

Validity Period Check

- Not possible on-card (there is no clock; hence, no current time)
- Solution proposed



M1 = [SCCert, ECSPuK(Nonce), SSCPrK (Nonce)]

M2 = [ESCPuK(Nonce+1), ESCPuK(Date), SCSPrK (Date,Nonce+1)]

Certificate Revocation Check

- Not possible on-card, it is not possible to connect to the revocation server
- Solution proposed close to the validity period issue

Certificates Renewal

- Smart card cannot check who sends every message; hence once it does not know if it is a TTP or a TR who tries to store a certificate or a policy
- Solution in the prototype, everything is stored only once
- Problem: when the certificates expire, card is useless
- Solution: to store TR certificate during initialization avoiding anyone could replace it
- Send the certificate with a signature to insure authentication and integrity
- In such a way, the card could be sure that is the TR who is sending the message

Time and Effort Management