Universidad Zaragoza

1542

DTU

Master Thesis

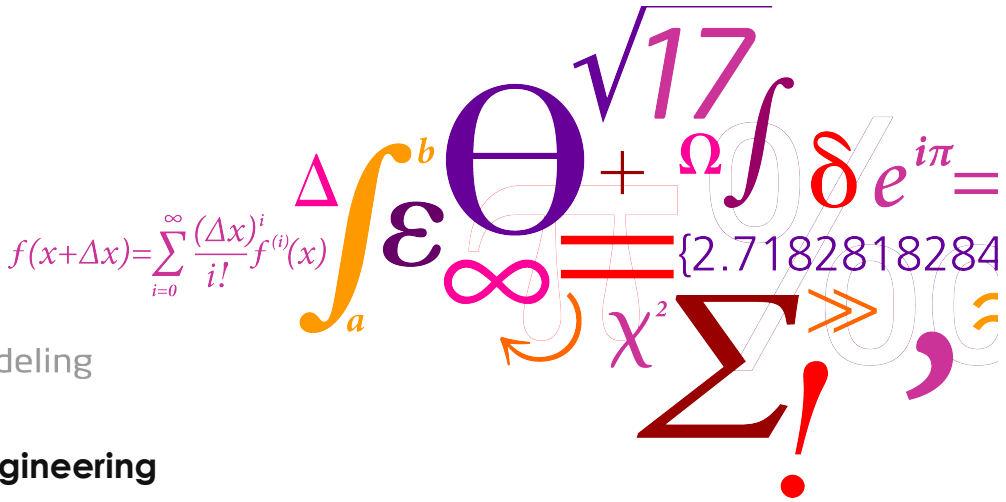
Securing Multi-Application Smart Cards by Security-by-Contract

Author: Eduardo Lostal Lanza

Supervisor: Nicola Dragoni

Tutor: Fco. Javier Fabra Caro

September 17th, 2010



DTU Informatics

Department of Informatics and Mathematical Modeling

Dep. of Computer Science and Systems Engineering

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**
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

- Extending SxC Off-Card
- Implementation
- Evaluation

Solution

- Conclusion

Conclusion

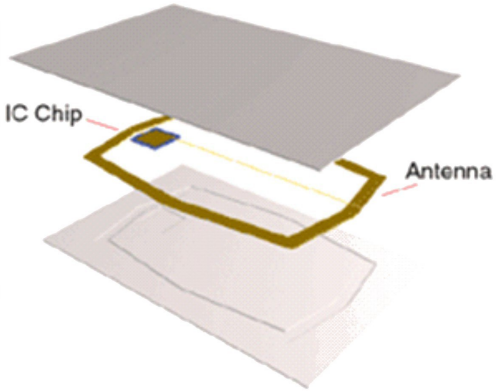
Smart Card

- 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!

Interfaces { Contact / Contactless

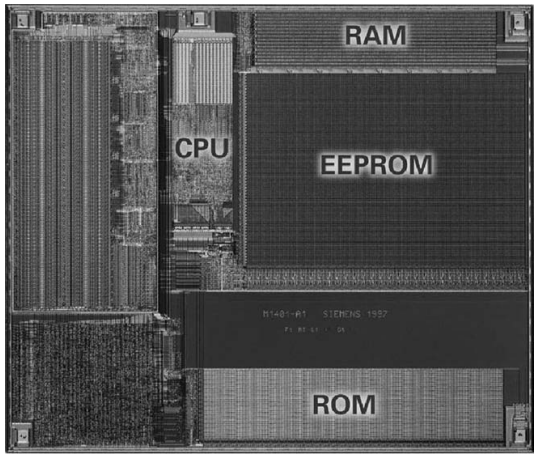


Microprocessor

- Not specifically built: Money and security

Memory

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



Coprocessors

- Commonly used for: Cryptography algorithms, Random-Number generator


Grid computing
Banking
Telecommunication
Pocket-gamming
Authentication
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
ELECTRONIC PASSPORT

Contact Smart cards -> Physical attacks

- Invasive, Side-Channel Attacks
- 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

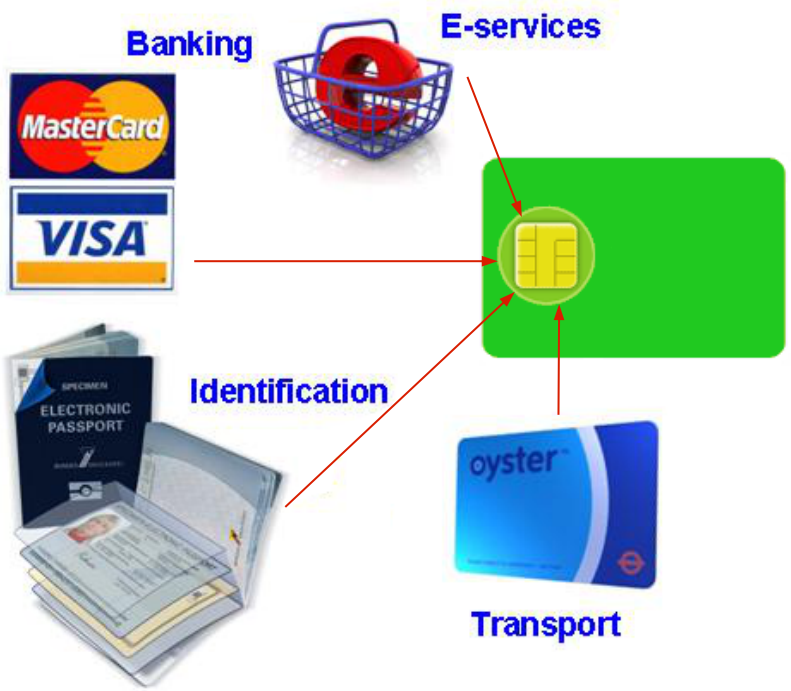
- Smart Card Technology
- **Multi-Application Smart Cards**
- Security-by-Contract
- Extending SxC Off-Card
- Implementation
- Evaluation
- Conclusion

Background

Problem

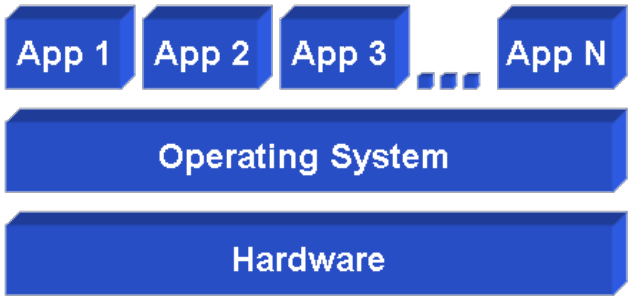
Solution

Conclusion



Dynamic load post-issuance!

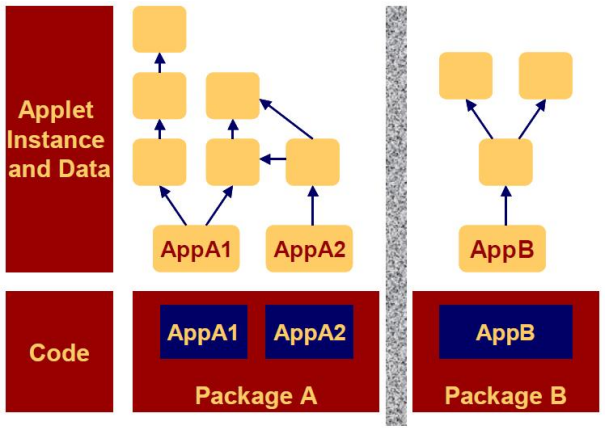
Flexible, open and secure platform to load dynamically applications after card issuance



Global Platform

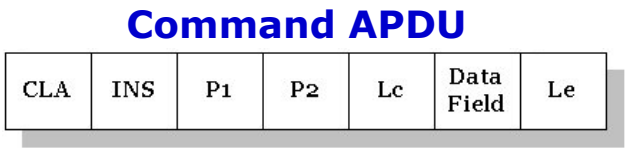
- Set of specifications to create a standard card management
- Security Domains

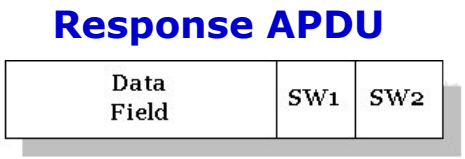
Java Card



- Subset Java
- Object oriented, interoperable, portable
- Context isolation
- Enforced by firewall
- Shareable Interface Objects to allow accessing through firewall

- 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

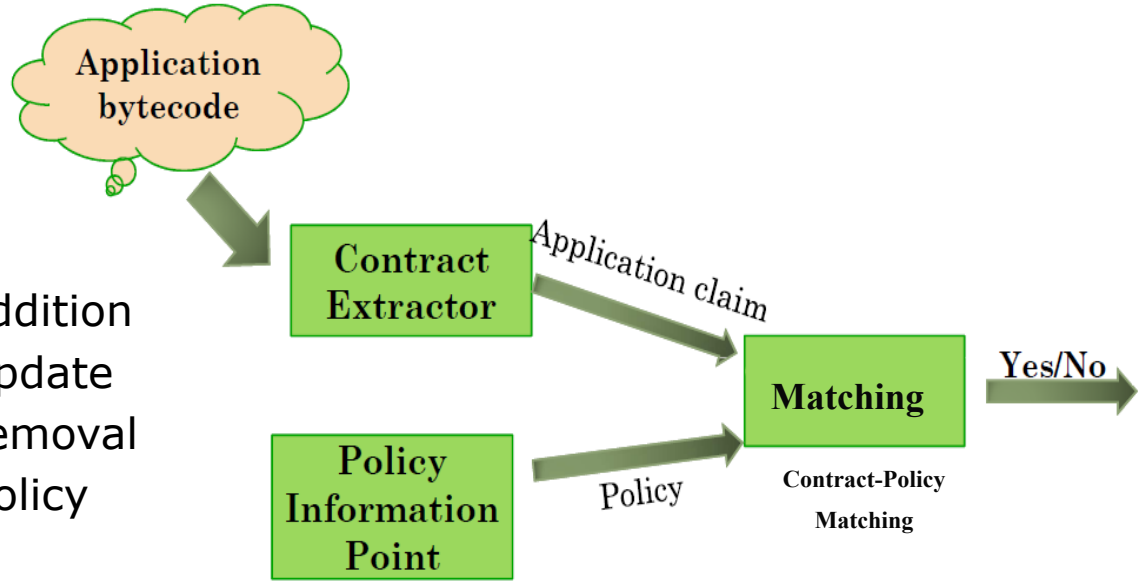
Solution

- Conclusion

Conclusion

Security-by-Contract

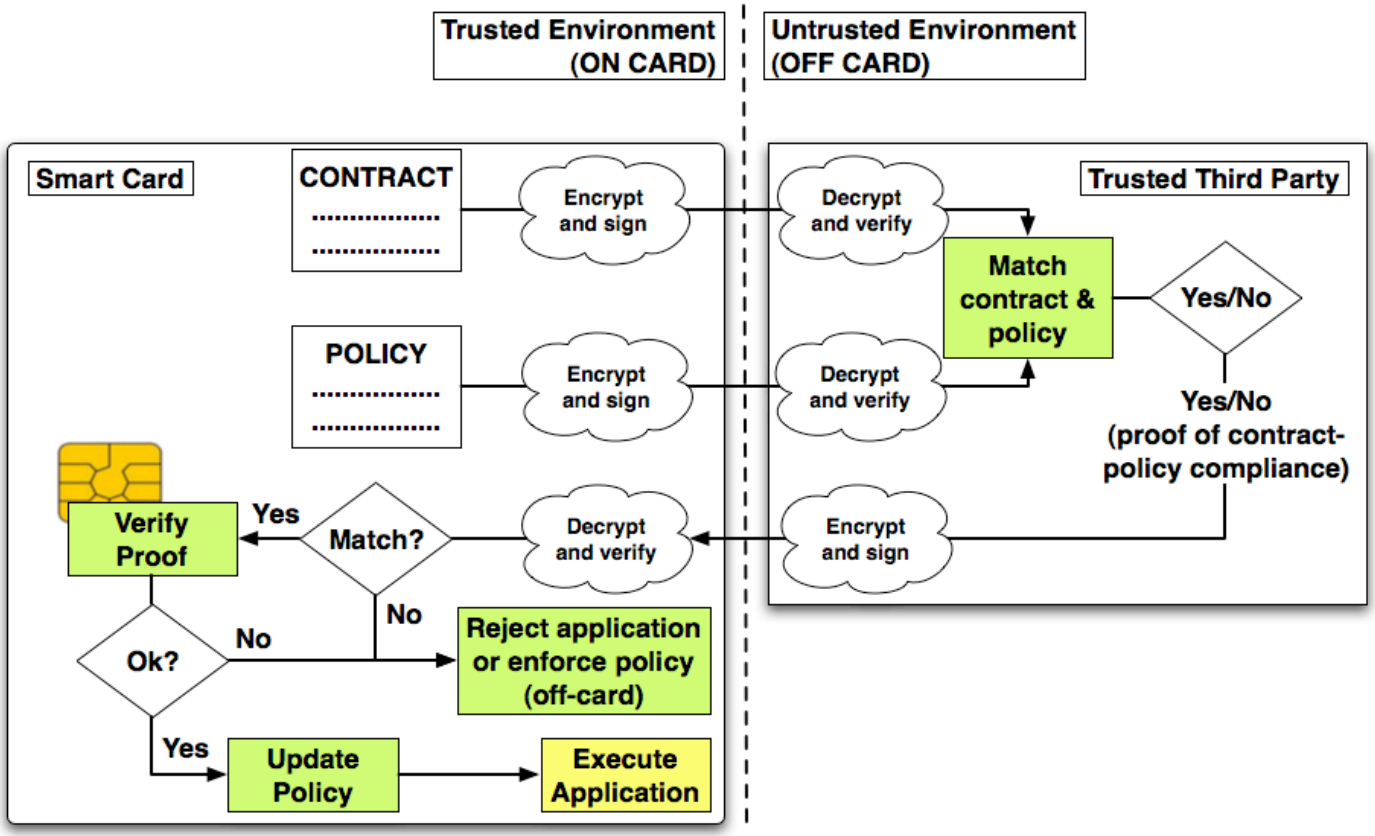
- Key point: Contract-Policy Matching (at load time)
- To solve { New application will not interact with forbidden ones on-card
  Dynamic change will not affect the correct work
- Dynamic change { addition
  update
  removal
  policy



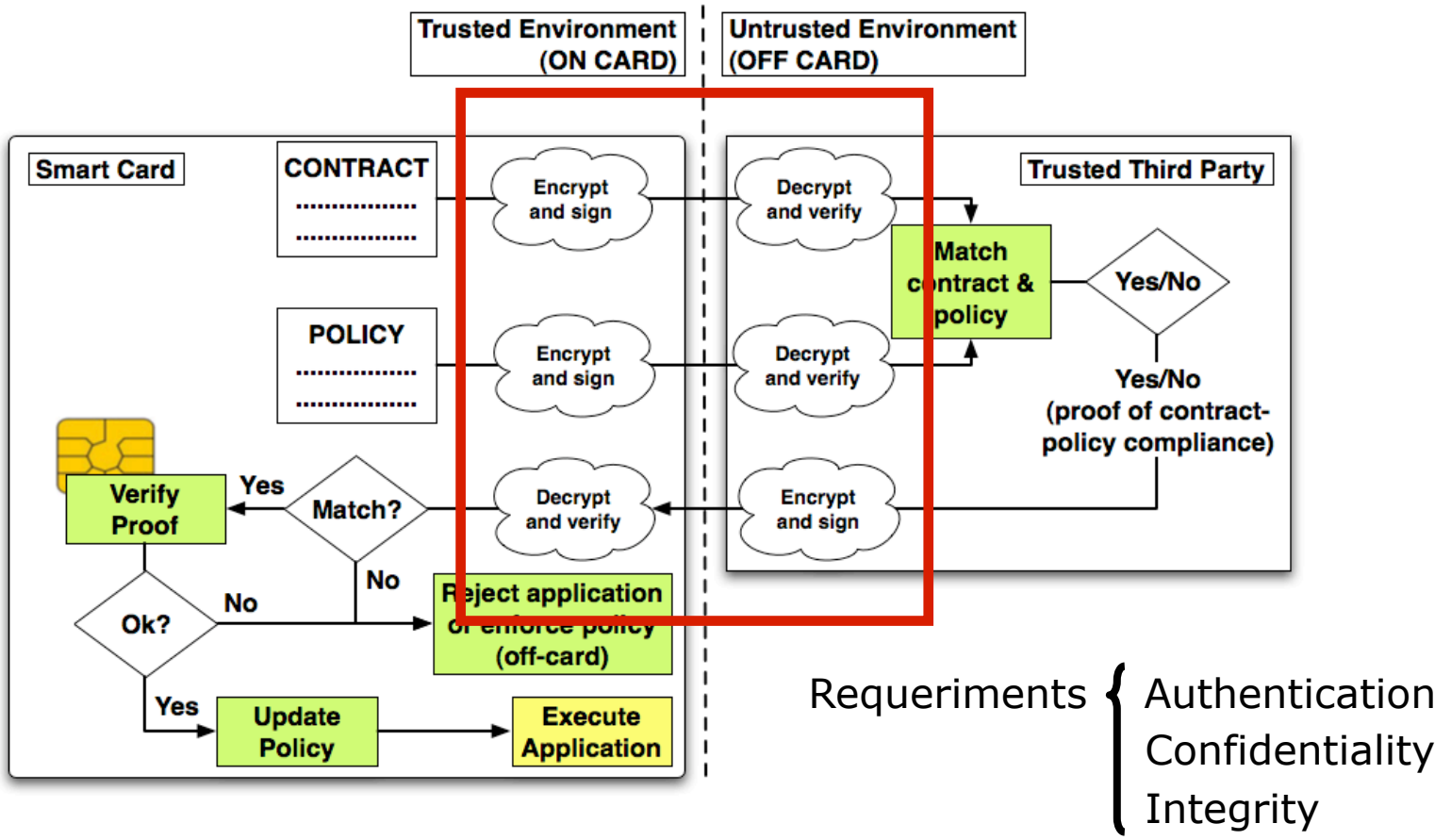
Hierarchy of Models

- 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
- Why do not use Level 3 always (specification most complete)?

Problem: Securing Off-Card Contract-Policy Matching



Problem: Securing Off-Card Contract-Policy Matching



- Smart Card Technology
- Multi-Application Smart Cards

Background

- Security-by-Contract

Problem

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

Solution

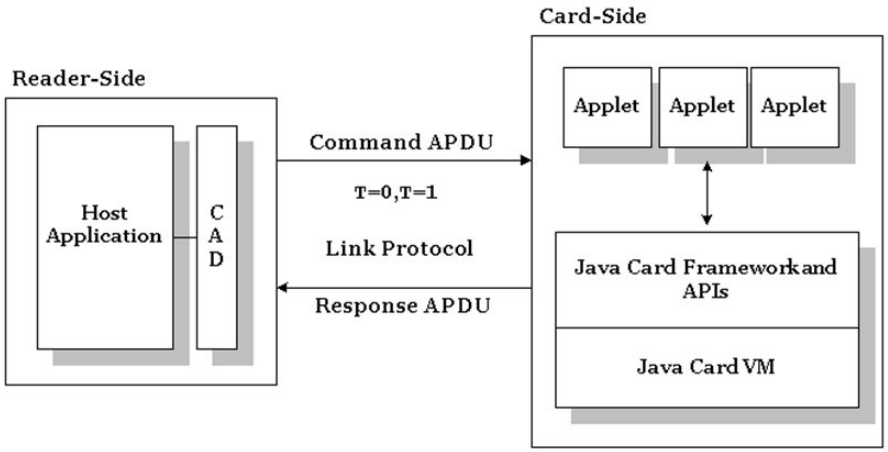
- Conclusion

Conclusion

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

} Initialization needed

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



Installation Phase

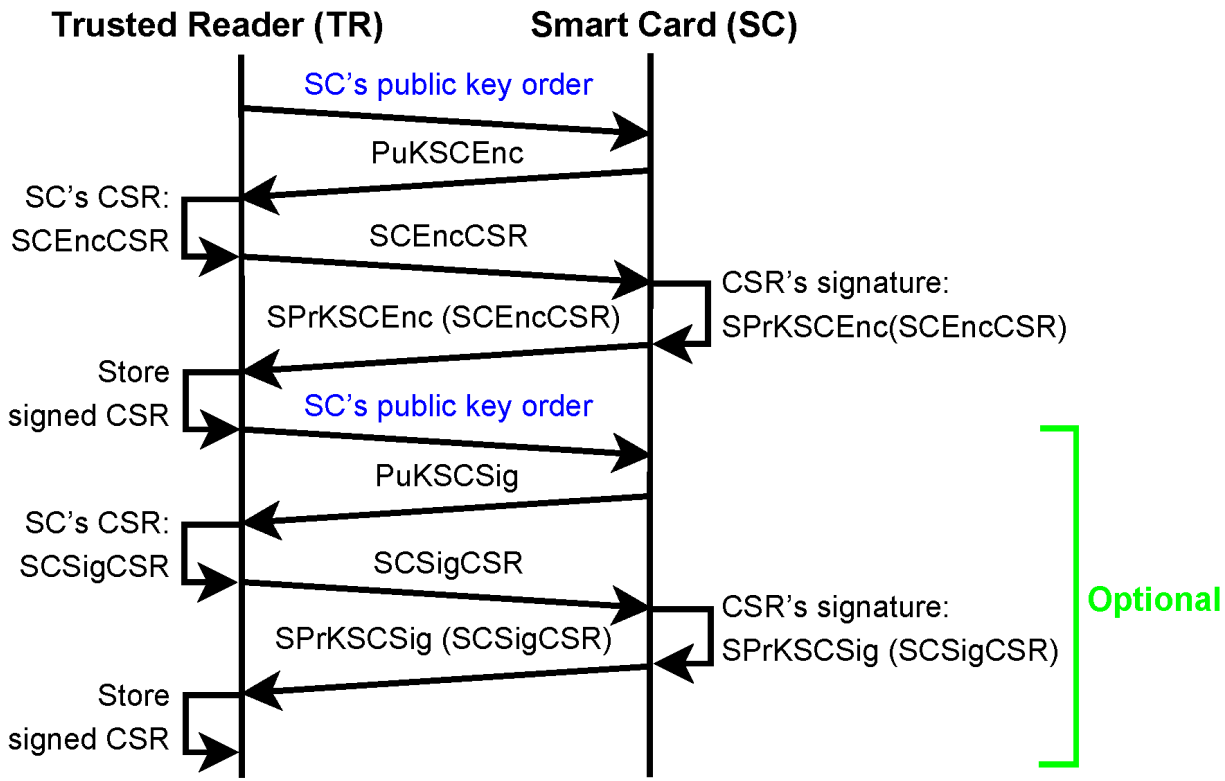
- After deploying application on the card
- Install the application on the card
- Generate the keys
- Security highlight of the system: Private keys do not leave the card simply because there is no reason to do that

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

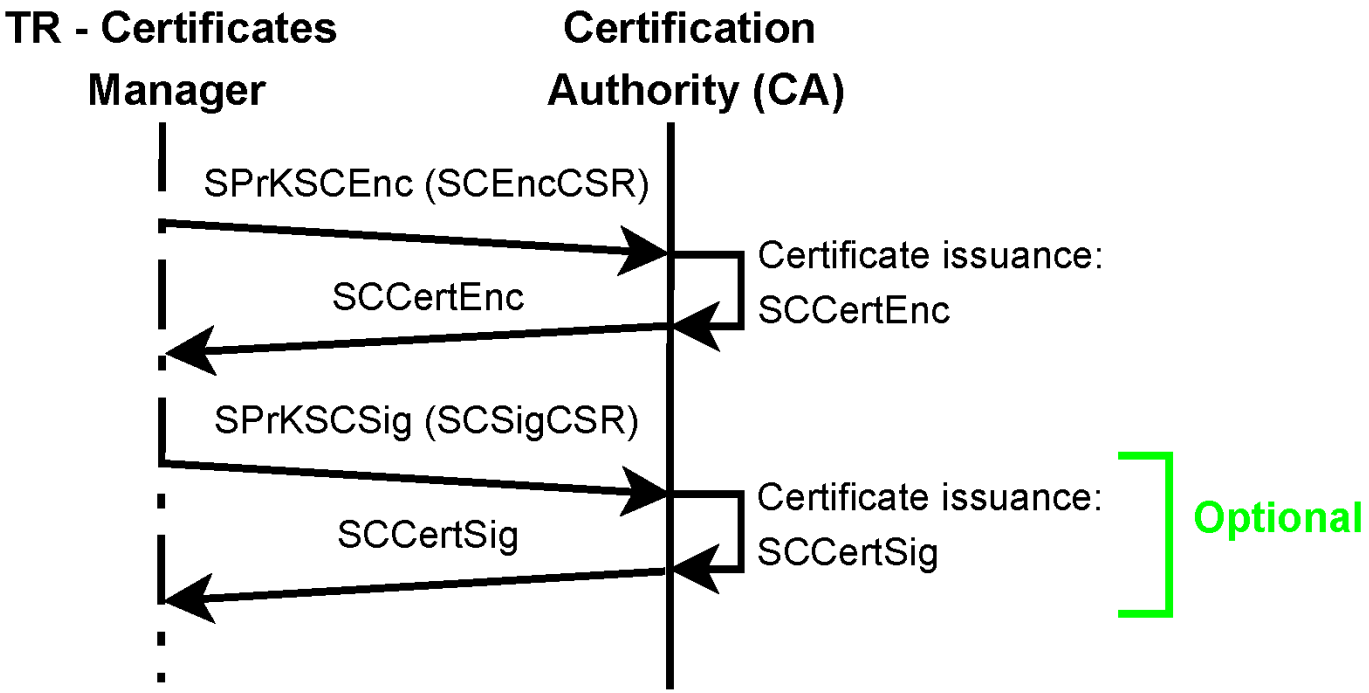
Initialization Phase

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

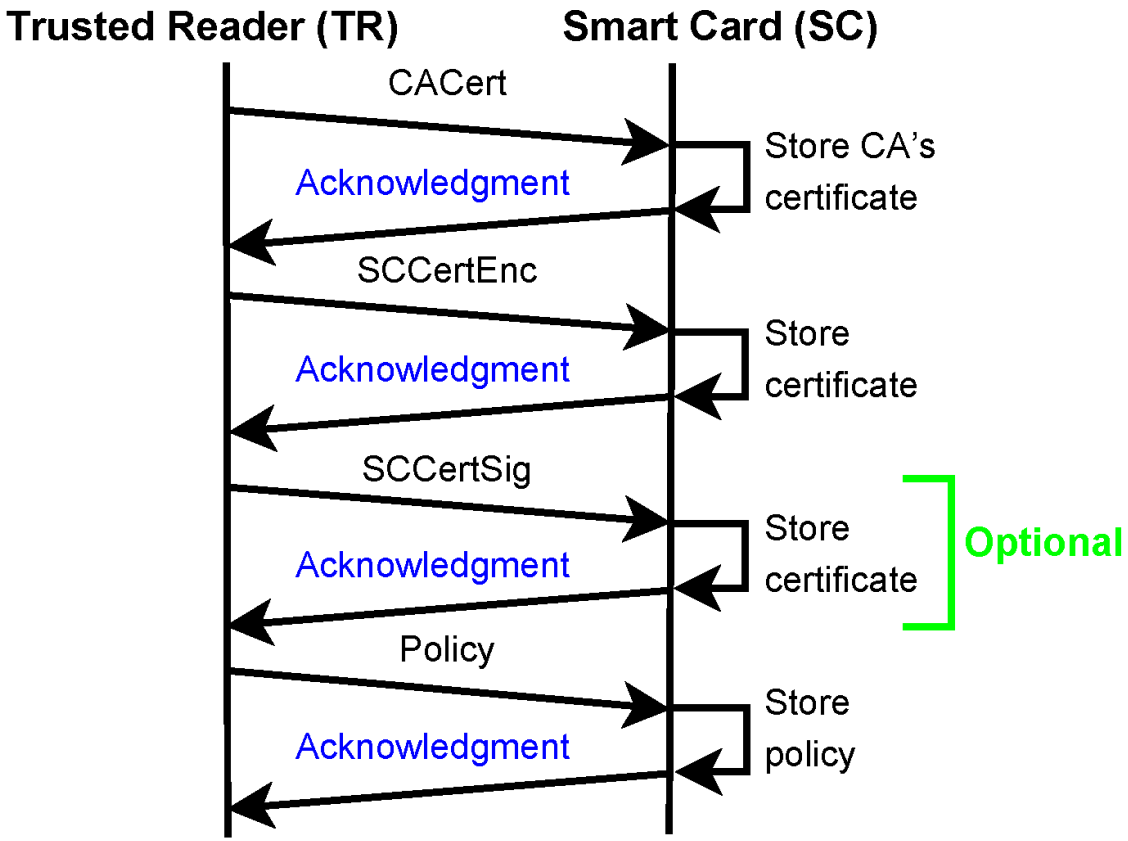
Initialization Phase: Certificate Signing Requests Building



Initialization Phase: Certificates Issuance

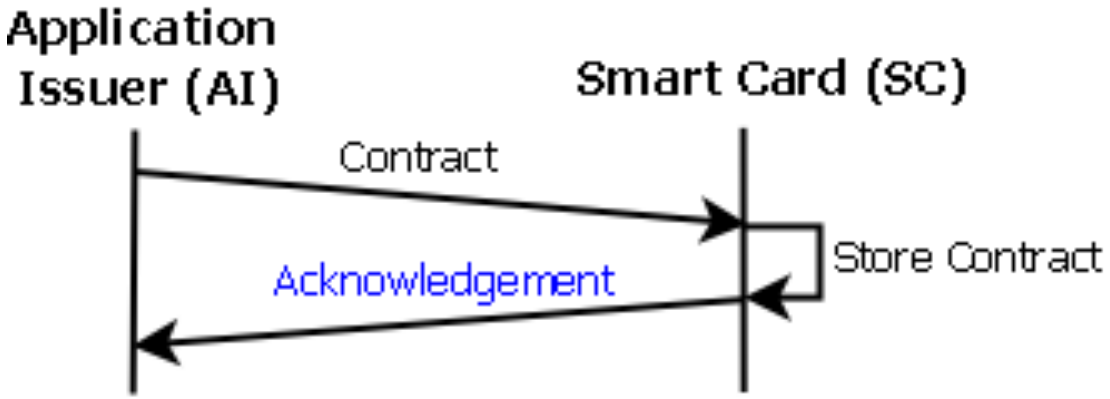


Initialization Phase: Certificates and Policy Storage

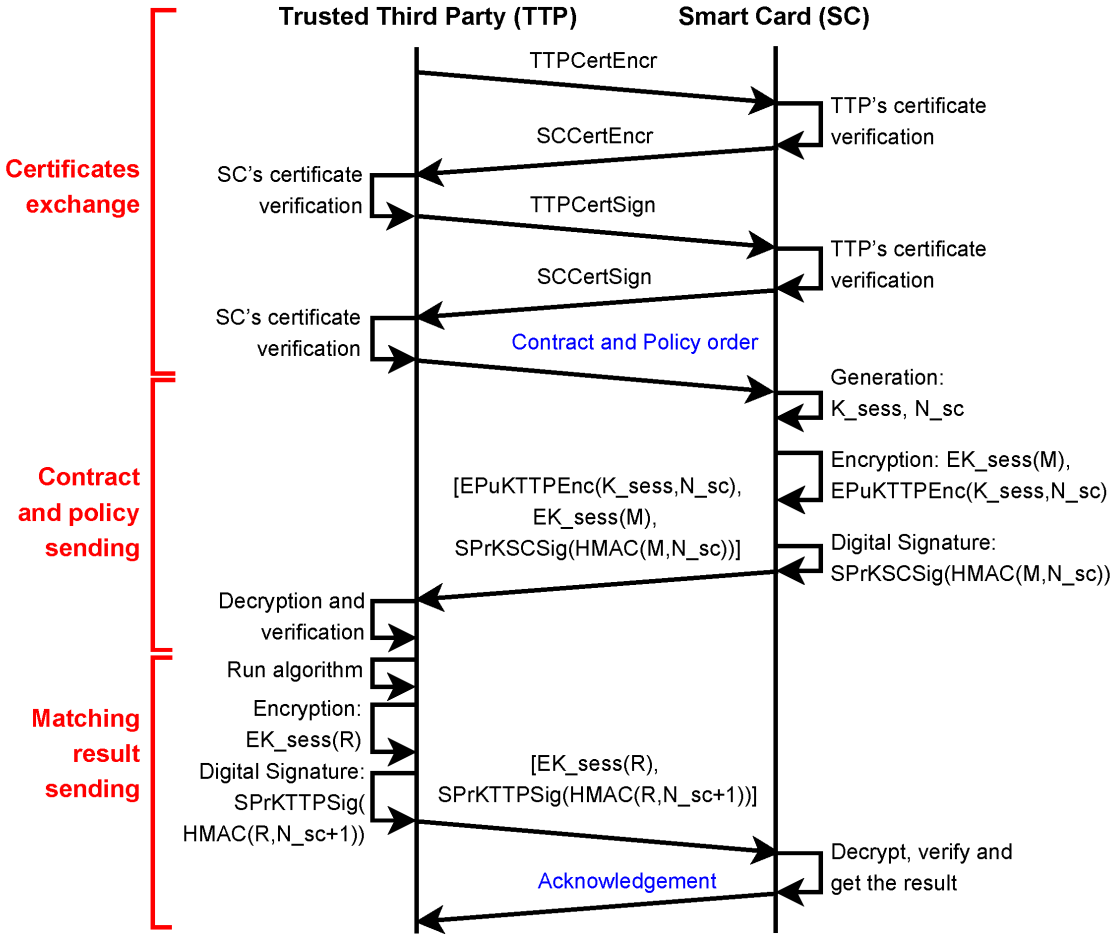


Contract Storage Phase

- 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** (sun.* packages, jdk 1.6.0.18)
- Smart Card (SC) → **Java Card 2.2.2** (Why not Java Card 3?)

- IDE → Eclipse SDK 3.5.2

- Simulators { JCWDE, CREF } Both provide a subset of the cryptographic classes

- 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 generated by means of OpenSSL
- CA certificate is self-signed

On-Card → Byte arrays, DER encoded

Parser on-card

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

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)

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
- 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

Certificates **Encryption** **Digital Signature**

- 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)
  - ”Securing Off-Card Contract-Policy Matching in Security-By-Contract for Multi-Application Smart Cards”
    N. Dragoni, E. Lostal, D. Papini and J. Fabra

- 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

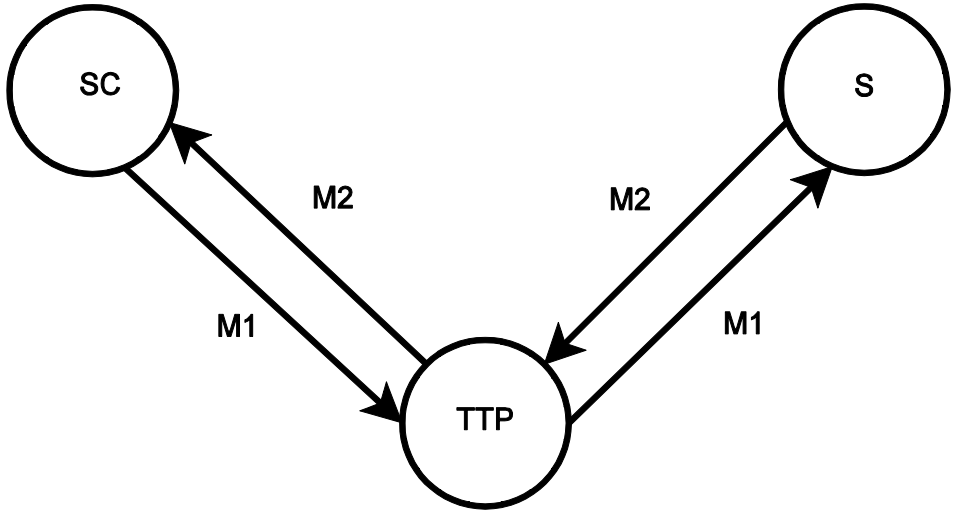
- More stable version of the system and independently of Sun's code updates (still working for the specified jdk version)
- Building classes to work with DER encoding
- Goal is to get a CSR which format was compliant with the standard
- Even it is possible to do on-card
- Initialization phase would be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



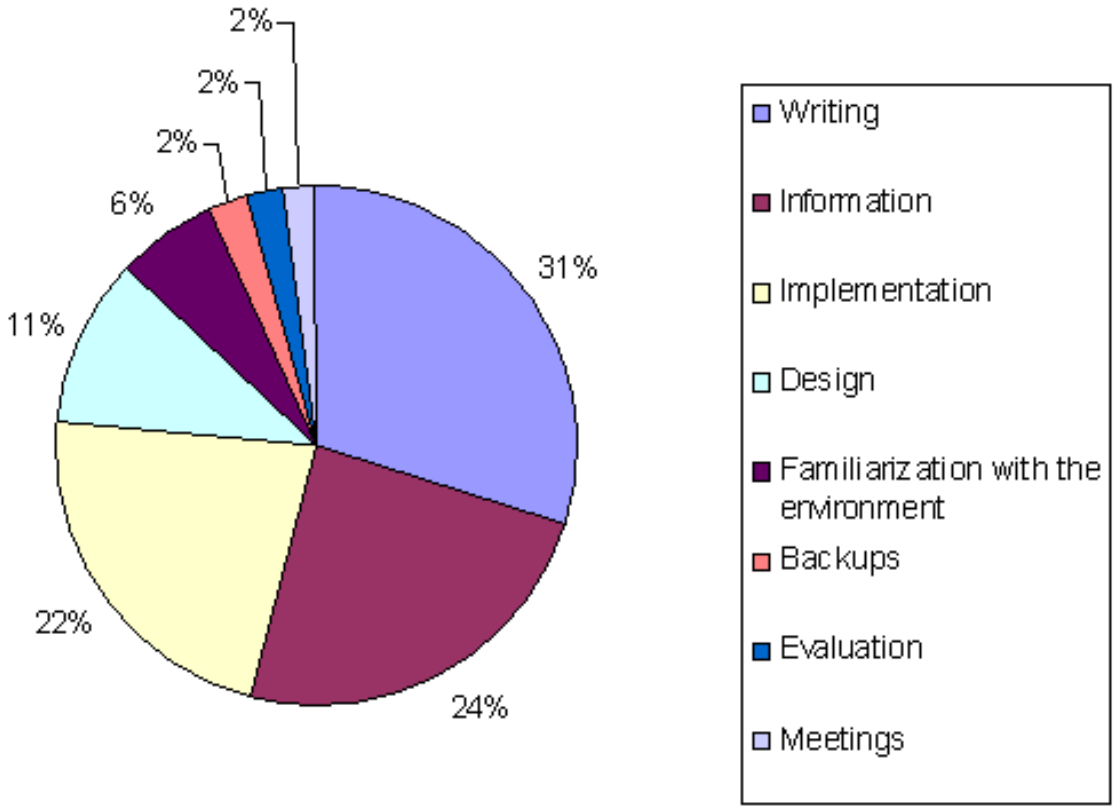
Certificate Revocation Check

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

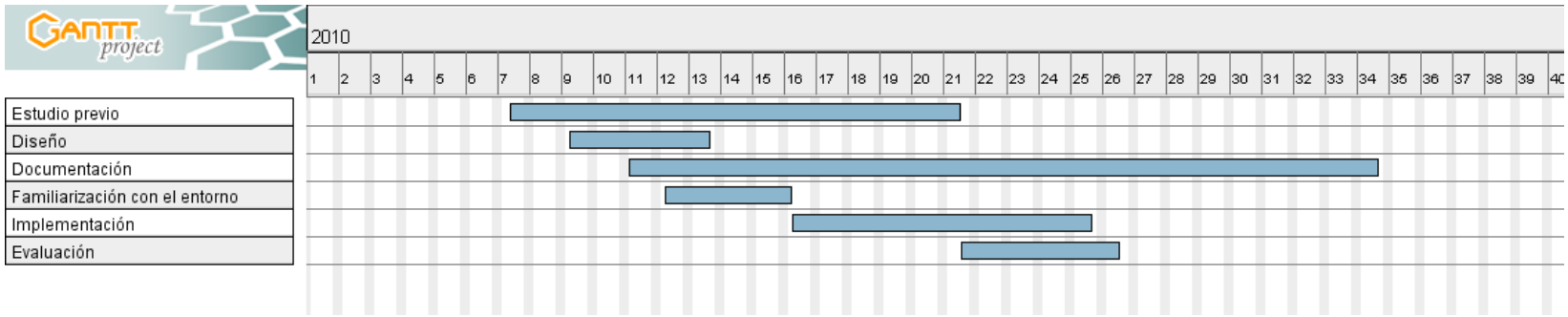
Certificates Renewal

- 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

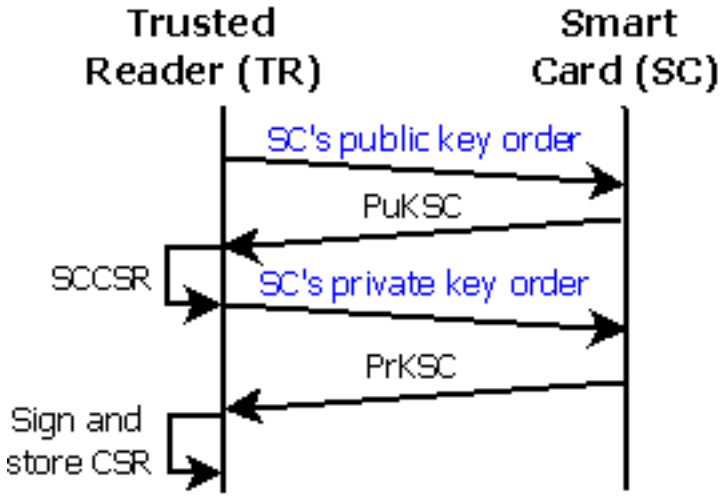
Time and Effort Management



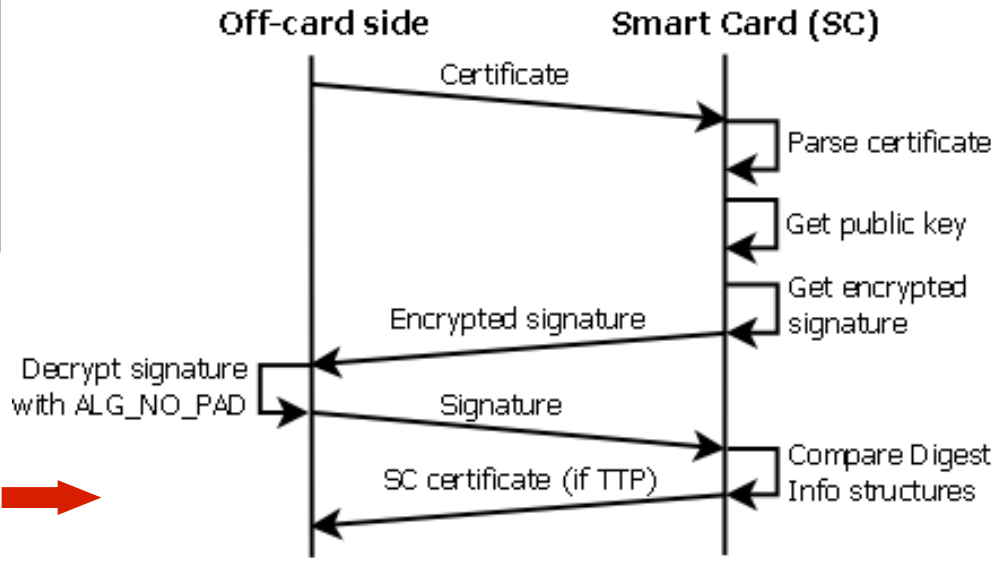
Time and Effort Management



Changes because of simulator



CSR generation modified



Signature verification modified