Current Trends in Smart Card Research

Finnish Data Processing Week 2005
Petrozavodsk State University
May 17-19, 2005

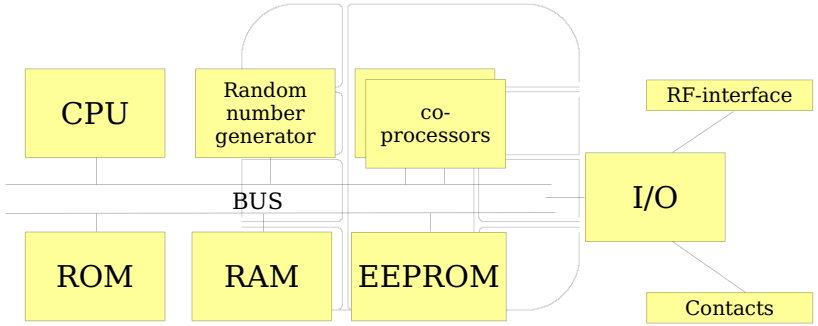
Olli Vertanen
Researcher, M.Sc.
University of Kuopio
vertanen@cs.uku.fi

Presentation outline

- Introduction
  - What is smart card
  - Smart card application environment
  - Current trends in the industry
  - Java Card
- Are smart cards secure?
- Smart Card [RE|E]volution – some possible paths

What is smart card? (1)

- Smart card: a chip on plastic



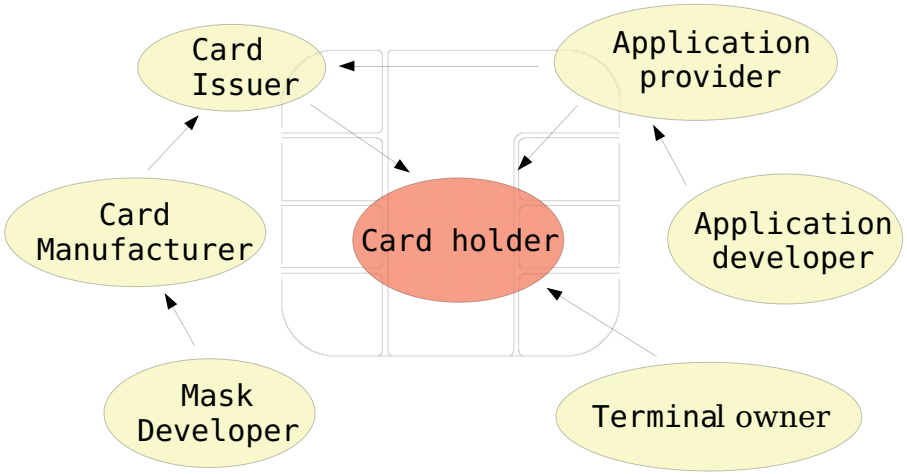
What is smart card? (2)

- Smart card is a “handicapped” computer – it lacks peripherals for interaction. Processing power is relatively low. So why do we use them?
- Emphasis on the security!
  - The existence of smart cards is justified in their ability to keep secrets and process secrets confidentially.
- Other computing tasks can be implemented more efficiently by other means (mobile phones, PDAs, ...).

Smart Card application areas

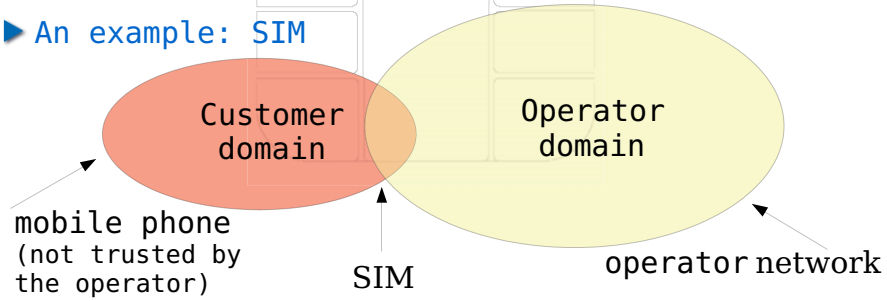
- Smart card are used in:
  - Mobile phone industry (SIM, USIM)
  - Banking (Credit cards, Electronic purse)
  - Identification (electronic id cards, ePassport, access control cards)
  - Health care
  - Transportation
  - Pay-TV
  - Digital Rights Management
  - ...

Smart card application environment (1)



Smart card application environment (2)

- Smart card contains data that is controlled by another party than the card holder.
- In the other words: smart card is used to propagate trust from one authority domain to another.
- An example: SIM



Trends in the industry

- Processor: 8 bit → 16 bit → 32 bit
- Memory: KB → MB, EEPROM → Flash
- Communications: Kb → Mb (USB, Wireless)
- Applications:
  - Assembler/C → Java Card!
  - Single → multiple applications!
- Cryptography: DES → AES, RSA/DSA → ECC
- HW random number generators on card
- New formats in addition to traditional ISO 7816-1/2 (e.g. Memory Card)

Java Card (1)

- Introduced 1996 (Schlumberger's initiative)
- What was revolutionary:
  - Object oriented language for smart cards (applets),
  - multi-application capable platform,
  - application loading after issuance (post-issuance).
- Current version 2.2.1
  - next: 3.0

Java Card (2)

- A tiny implementation of Java. But not exactly a subset of the big brother:
  - Transactions and atomic operation,
  - persistent objects,
  - applet firewall and controlled data sharing,
  - no dynamic class loading, object cloning, threads, ...
  - lacks most of the standard Java APIs,
  - own binary format (.cap, different from Java class file format).

Java Card (3)

- After introduction of Java Card a great deal of smart card research has been computer language research!
- Java Card was a perfect match for formal method scientists:
  - Small language,
  - simple programs,
  - need for provably secure platform and provably correct applications.
- Java Card specification has been carefully scrutinized by the academia.

Java Card (4)

- Java Card is and will be dominating card platform in the near future.
- The current version is considered too limited.
- What next?
  - Industry is soon possible to provide cards comparable to todays hand-helds (in terms of memory size and processor power).
  - These cards can embed full Java (e.g. J2ME).

Java Card (5)

- But, smart card standards are lagging behind:
  - Archaic communication protocol,
  - archaic and arcane file system,
  - cumbersome application model,
  - ...
- There is very little (open) discussion going on what the next generation card platform should look like.

Java Card (6)

- Possible future directions?
  - J2ME? not applicable directly!
  - Communications: TCP/IP, RPC,...
  - OS: threads, resource control,
- Security vs. complexity!
  - Simple system are easier to make secure.
  - The smart card world is seemingly going to another direction...

Presentation outline

- Introduction
  - What is smart card
  - Smart card application environment
  - Current trends in the industry
  - Java Card
- → Are smart cards secure?
- Smart Card [RE|E]volution – some possible paths

Are smart cards secure?

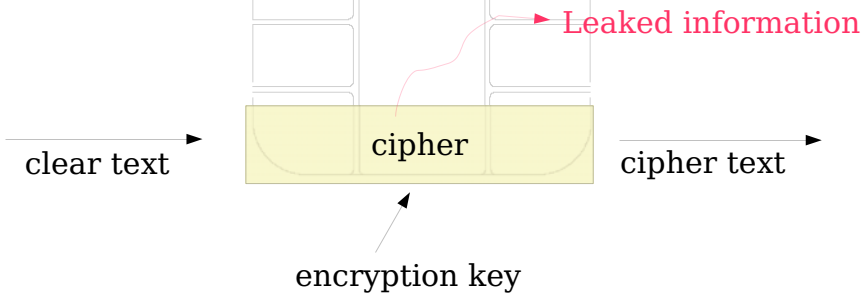
- Card holder ≠ data owner → mistrust!
- Threat: extraction of secret/private information
  - cryptographic keys
  - program code etc.
- Attack scenarios has been a very intensively studied area during the last years.
- How to do it?
  - Physical attacks,
  - side-channel attacks,
  - fault attacks,
  - malicious code.

Physical attacks

- How:
  - Chip delayering (remove glue-top and passivation layer) mechanically and/or chemically,
  - block localization.
- Attacks:
  - Bus probing,
  - optical memory read out (microscope).
- Getting harder when feature size shrink.
- Today secure devices are called "tamper resistant" rather than "tamper proof".

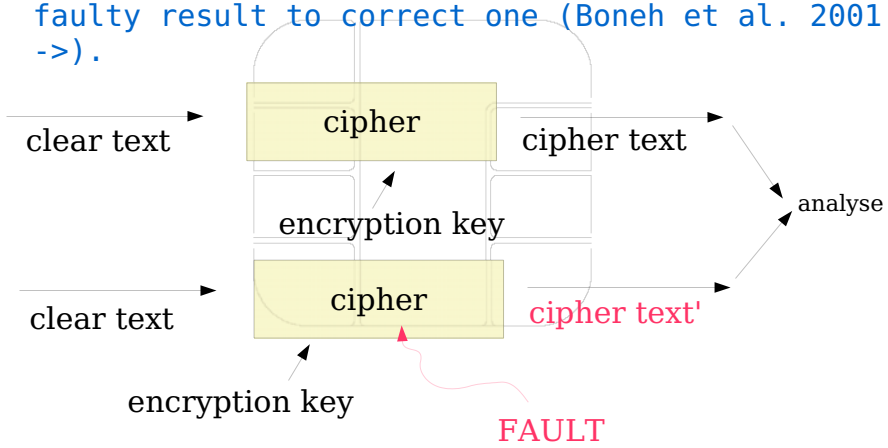
Side-channel attacks

- Analyze information leaked through an abnormal channel (P. Kocher, 1996).
- Observe power consumption, electromagnetic radiation, magnetic field, timing, ...



Fault attacks (1)

- Introduce a fault during computation. Compare faulty result to correct one (Boneh et al. 2001 ->).



Fault attacks (2)

- How to inject a fault:
  - Glitches: voltage, clock,
  - radiation: white light, laser, ionizing radiation ...
- Fault attacks have proved very effective in breaking cryptographic systems.
- What else can be done:
  - Nullify/modify instructions,
  - change data on bus...
- Fault attacks are a good example, how theoretically sound systems fail in real life.

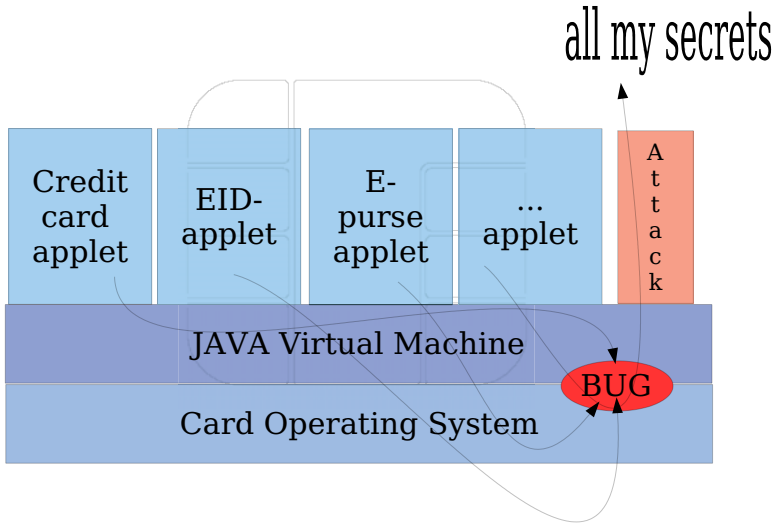
Malicious code (1)

- Java type system
  - Strong typing is the cornerstone of Java security.
  - Java compiled to JVML (Java Virtual Machine Language = byte-code).
  - Type consistency of JVML is checked in byte-code verification.
    - verification: static type level abstract interpretation of the code
  - Some checks done run time (array bounds, null reference, casts...)

Malicious code (2)

- Type system is easy to circumvent if verification cannot be enforced (e.g. an illegal cast from an integer to pointer).
- Combined with fault attacks!!
  - A single bit error can be exploited to compromise the whole VM, if the attacker can choose the program to run! (demonstrated by Govindavajhala & Appel, 2003.).
- Other attacks: exploiting holes in the implementation (like with any other OS).
- Cards with multiple application has been introduced with caution. (see next slide).

Multi-application nightmare scenario!



Presentation outline

- Introduction
  - What is smart card
  - Smart card application environment
  - Current trends in the industry
  - Java Card
- Are smart cards secure?
- Smart Card [RE|E]volution – some possible paths

Counter-measures (1)

- What is there to be done?
- Evolutionary approaches: add counter-measures when problems arise
  - The “normal” way to fight the beast.
  - Build new protection mechanisms on the top of existing architecture.
    - Hardware sensors, software checks, ...
  - Modify (crypto)algorithms; hacks specific to some vulnerability.

Counter-measures (2)

- Revolutionary approach
  - New way of thinking and doing things.
  - Try to enhance the overall security of the system.
- Some examples:
  - Applying asynchronous logic design,
  - ROM-less smart cards,
  - Defensive Java Virtual Machine (our approach).

Asynchronous design (1)

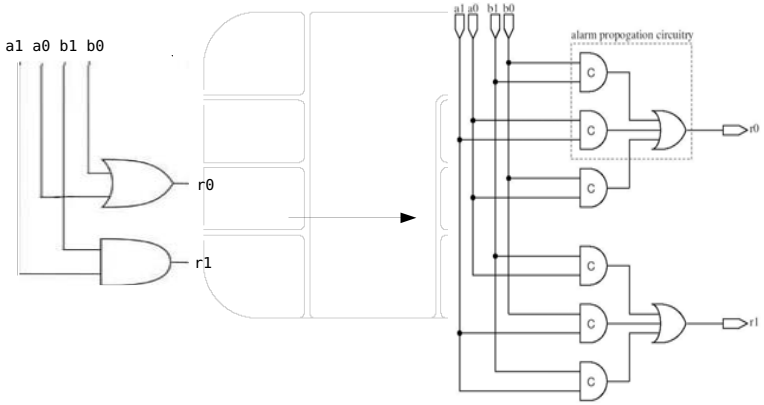
- Not a new idea (used already 1950's)
- New is to apply asynchronous logic to enhance security of the overall system:
  - Circuit level counter-measures!
- Asynchronous logic:
  - No global clock,
  - subsystems are 'self-timed',
  - subsystems use handshake protocol to exchange data,
  - dual-rail encoding.

Asynchronous design (2)

- Dual-rail encoding: two lines per bit
  - 00 – initial state, 10 – logical 0, 01 – logical 0, 11 – ALARM (fault).
- Some motivations:
  - Fault tolerance/detection by redundant data encoding.
  - Data independent power consumption:
    - 1 and 0 present equal amount of transitions in gates thanks to dual rail encoding and return-to-zero switching.
  - No global clock, no clock glitch attacks.

Asynchronous design (3)

An example: dual-rail encoded OR-gate



(Moore et.al. 2003)

Asynchronous design (4)

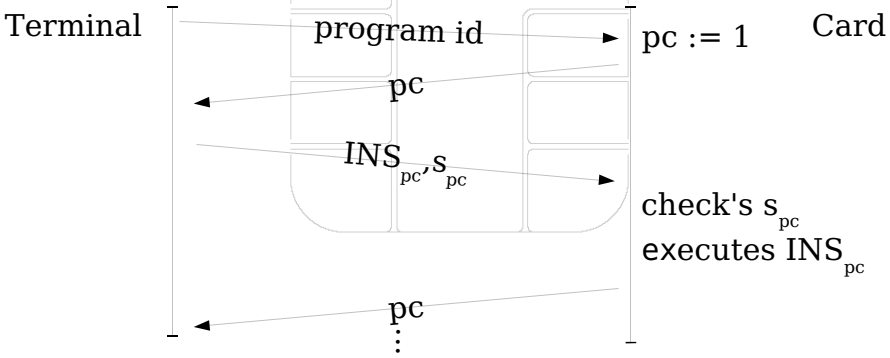
- Springbank SC-XAP processor (Moore et al., 2003).
- Lessons learned: (Fournier et al., 2003)
  - Extremely careful design needed (equal wire lengths etc.),
  - time domain and electromagnetic emission is difficult to handle.

ROM-less smart cards (1)

- The idea (Chevallier-Mames et al. 2004)
  - Terminal is needed for interaction with card.
  - Terminals have more resources than cards.
  - Why couldn't the terminal provide the code for cards?
  - 'Externalized microprocessor', no program memory!
- The objective:
  - To fight the 'complexity explosion' of smart cards

ROM-less smart cards (2)

- Programs authenticity must be ensured:
  - signed instruction: $s(id,pc,INS_{pc})$.



(The inefficient version of the protocol, Chevallier-Manes et.al. 2004)

ROM-less smart cards (3)

- Some reasoning:
  - It is hard to attack an algorithm, if there is no way to interfere with it.
    - power analysis? fault attack?
  - Program updates are made easier: nbr of terminals << nbr of cards.
  - How to reach acceptable performance?
    - High speed I/O,
    - efficient signature checks.
  - How to store objects (code+data)?
- The development is still at very early stage.

Defensive Java Virtual Machine dJVM (1)

- The idea: Defensive Java Virtual Machine enforces Java security/safety rules at run-time – dynamically.

Now:

Our approach:

dJVM (2)

- Motivation:
  - Static verification is prone to 'time-to-check-time-to-use' vulnerability. (Not resistant to fault attacks.)
  - Type rules are easier to check at run-time than during abstract interpretation (no ambiguities).
  - Implementation generally considered infeasible.
- Also:
  - Increase of 'natural' upsets (faults).
  - Radical increase of embedded VM technology.

dJVM (3)

- Already formally defined to some extent (Cohen 1997, Hartel et al., 1999).
- Goals:
  - To examine applicability of run-time type checking.
  - Non-compromising execution of JVML code under fault attacks.
  - Fault detection.
  - Side-channel attacks not addressed.

dJVM (4)

- Challenges
  - Performance, performance, performance...
- Ingredients:
  - Logic design,
  - computer architecture,
  - memory hierarchies,
  - fault detection techniques,
  - fault tolerance techniques,
  - virtual machine technology,
  - cryptography.

Conclusions

- Smart cards propagate trust between authority domains.
- Advances in semiconductor technology is about to break current resource limitations.
- Attacks on cards are getting more sophisticated.
- Open questions:
  - Do we need more complex cards?
  - How to make use of increased capabilities?
  - What does the next generation Java Card look like?

Thank you!

Questions?

Comments?