# *Tamper resistant devices*

*Levente Buttyán*
Laboratory of Cryptography and System Security (CrySyS)
Budapest University of Technology and Economics
**buttyan@crysys.hu**

# *Why are they needed in SeVeCom?*

- implementing security services in vehicular networks requires cars to store sensitive data [RayaH05sasn]
  - cryptographic keys (secret keys, private keys), event logs, …
- sensitive data needs to be protected from unauthorized access
- cars operate in a *hostile environment*
  - unsupervised access to all parts of a car by potentially malicious parties (car owners and maintenance service providers) is possible
  - there may be incentives to compromise the data (e.g., to modify event logs by the car owner)
- detection of an attack seems to be impossible in "real-time"
  - system is fully distributed, there's no central entity which can collect and analyze transaction logs, and identify anomalies
    (such an entity may not even be desirable for other reasons such as privacy)
  - tampering with cars may be detected by authorities at regular inspections, but only months (or perhaps years) later
- attacks should be *prevented*
  - tamper resistant hardware in cars could be useful !

# Outline and objective

- outline
  - FIPS 140
  - smart cards
  - the IBM 4758 coprocessor
  - conclusions

- the objective is to understand
  - what tamper resistance means ?
  - what kind of tamper resistant devices exist ?
  - what they are capable for ?

- this understanding is necessary to base the design of the SeVeCom security architecture on the right assumptions

# FIPS 140

- benchmark standard that specifies the security requirements for cryptographic modules

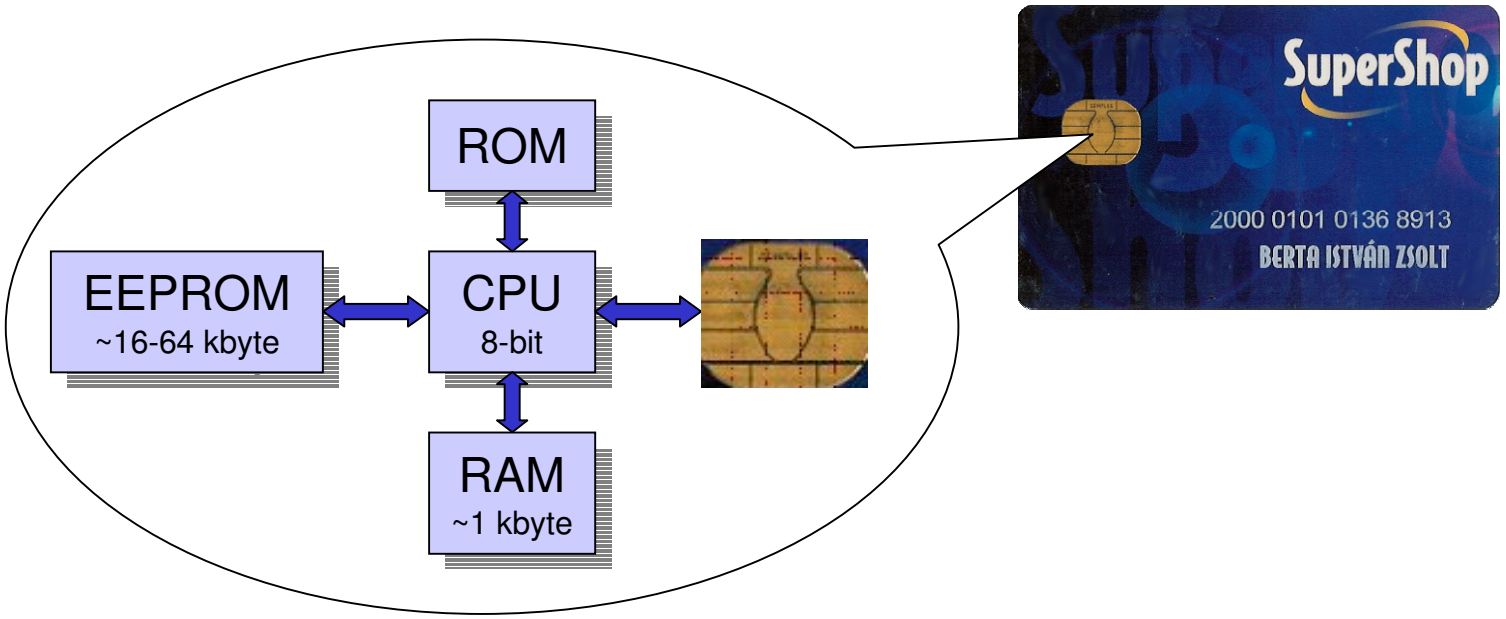
- types of requirements considered:
  - cryptographic module specification
    - algorithms, modes of operation, description of HW, SW, FW, security policy
  - ports and interfaces
  - roles, services, and operator authentication
  - finite state model
    - states and state transitions
  - physical security
    - locks, seals, coatings, covers, tamper detection and response
  - operational environment
    - OS requirements
  - key management
    - random number generation, key generation and storage, key erasure
  - EMI/EMC
  - self-tests
  - design assurances
    - configuration management, delivery and operation, development

# FIPS 140 security levels

- level 1
  - basic requirements on cryptographic algorithms
  - no physical security mechanisms are required in the module
  - examples: encryption software on a PC
- level 2
  - needs tamper evident coating or seals
  - requires role based access control
  - OS evaluated at CC level EAL2 (or higher)
- level 3
  - enhanced physical security preventing unauthorized access to stored sensitive data
  - requires identity based access control
  - data ports used for critical security parameters must be physically separated from other data ports
  - parameters must either be entered into or output from the module in encrypted form or be directly entered into or output from the module using split knowledge procedures
  - OS evaluated at CC level EAL3 (or higher)
- level 4
  - highly reliable tamper detection and response (immediately erasing all secret data)
  - protection against a compromise due to environmental conditions or fluctuations outside of the normal operating ranges (e.g., voltage, temperature, …)
  - OS evaluated at CC level EAL4 (or higher)

# Smart card basics

- a smart card is microcomputer embedded in a plastic card (credit card size or smaller)



# Smart card basics

- smart cards are used in a wide range of applications
  - bank cards
  - GSM SIM cards
  - electronic tickets for mass transport systems
  - payTV applications
  - access control to buildings
  - electronic ID cards, e-passports
- in all these applications, smart cards store sensitive data
  - crypto keys (even system master keys), access codes, account balance, …
- smart cards are intended to protect sensitive data in hostile environments, but …
  - their use is usually extended with other security measures (e.g., video surveillance, transaction log analysis and blacklisting, …)
  - when such additional measures are not applied, smart cards become less efficient and fraud pervades (see e.g., payTV systems)
- many smart cards support cryptographic operations
  - custom hardware for DES and modular arithmetics

# Security of smart cards

- access control
  - smart cards have a single interface
  - access to sensitive data through the interface is controlled by the smart card (OS)
  - authorization of access is based on PIN codes
  - after a certain number of unsuccessful attempts, the card blocks itself
- however, physical security is not very strong
  - smart cards do not resist tampering by a determined attacker with slightly above-average knowledge

# Classification of attackers

- clever outsider
  - intelligent but may have limited knowledge about the system
  - access to moderately sophisticated equipment
  - takes advantage of known weaknesses rather than create new ones
- knowledgeable insider
  - specialized technical education and experience
  - varying degrees of understanding of parts of the system
  - highly sophisticated tools and instruments for analysis
- funded organization
  - able to assemble teams of specialists with complementary skills
  - advanced analysis tools
  - backed by great funding resources
- most attacks against smart cards can be carried out by “clever outsiders” with few hundred dollars resources

# Non-invasive attacks

- non-invasive attacks do not destroy the card
- side-channel attacks
  - careful observation of the interaction of the card with its environment during critical operations may reveal some amount of information about the sensitive data stored in the card
  - examples: timing attacks and power analysis
- unusual operating conditions may have undocumented effects
  - unusual temperatures or voltages can affect EEPROM write operations
  - power and clock glitches may affect the execution of individual instructions

# Example

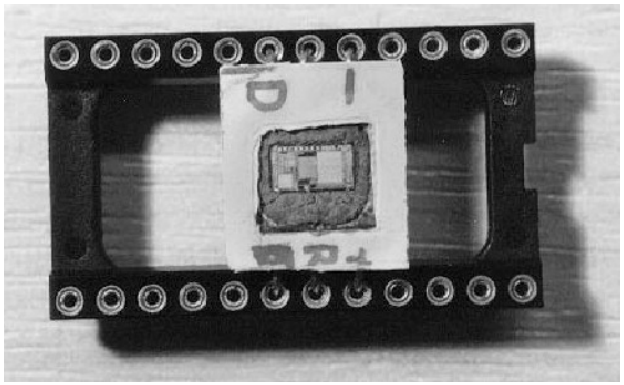
- a typical subroutine (writes the content of a limited memory range to the serial port):

```
1   a = answer_address
•   b = answer_length
•   if (b == 0) goto 8
•   transmit(*a)
•   a = a + 1
•   b = b – 1
•   goto 3
•   …
```

- if we can find a glitch that transforms the loop variable decrement in line 6 into something else, then the card will dump the content of the whole memory

# Physical attacks

- removing the chip from the plastic cover is rather easy
  - nitric acid dissolves epoxy without damaging silicon

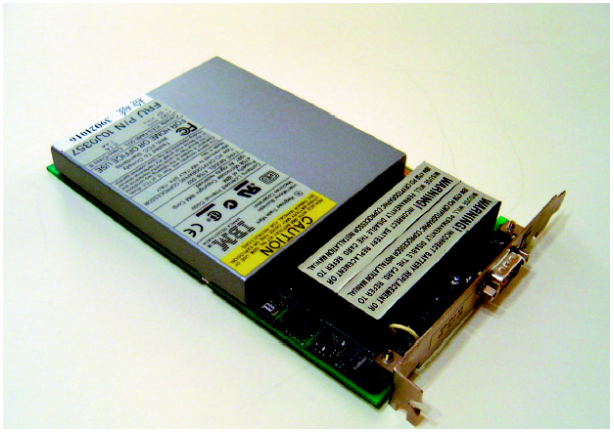


- once the chip is visible, one can use microprobing needles or electron beam testers to access on-chip signals and extract data from the chip

# Conclusion on smart cards

- the main advantage of smart cards is their *low cost*
- disadvantages:
  - no tamper resistance against a determined attacker
    - although there exist smart cards already with FIPS 140 level 3 evaluation
  - additional security measures (surveillance, blacklisting) are not feasible in the SeVeCom scenario
  - no battery, no on-board clock
  - limited support for multiple players
    - car manufacturers, national authorities, third party software providers
    - who would personalize the smart cards ?
    - who could put code on it ?

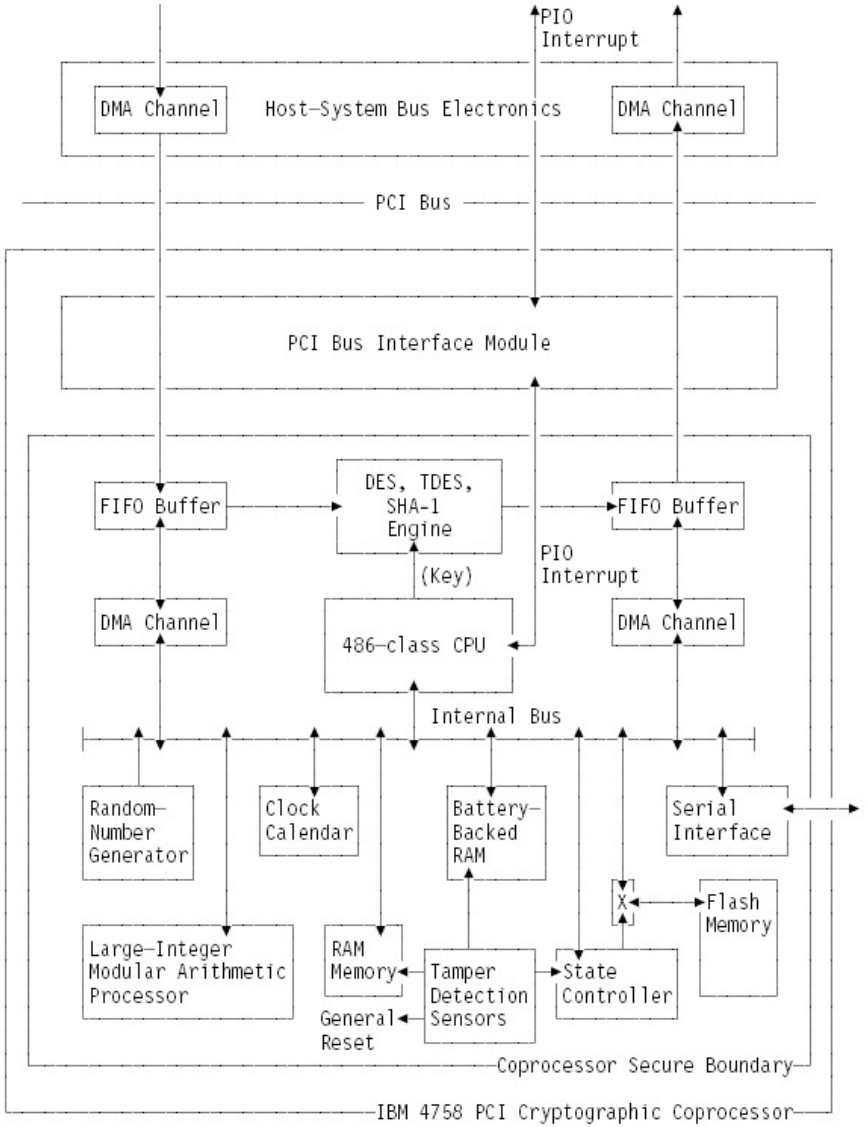
# The IBM 4758 cryptographic coprocessor



- programmable PCI board with custom hardware to support cryptography and tamper resistant packaging
- main features:
  - pipelined DES encryption engine
  - pipelined SHA-1 hash engine
  - 1024-bit and 2048-bit modular math hardware to support RSA and DSA
  - hardware noise source to seed random number generation
  - pseudo-random number generator
  - support for RSA key pair generation, encryption, and decryption
  - support for key management
    - DES based, RSA based, key diversification, PIN generation
  - secure clock-calendar
  - support for PKCS#11 and IBM Common Cryptographic Architecture (CCA)
  - battery backed RAM (BBRAM) to store secrets persistently
  - steel house with tamper detecting sensors and circuitry to erase the sensitive memory

# IBM 4758 hardware

- Intel 80486 CPU (66 OR 99 MHz)
- ROM for bootstrapping code
- 1-2 MB Flash memory to store bootstrapping code, OS, and applicatior code
- 4 MB RAM for applications data
- 32 KB BBRAM for secrets
- hardware support for common cryptographic operations
- random number generator
- on-board clock
- tamper detection sensors (with own battery)
- state controller
  - controls access to portions of the BBRAM and Flash memory
  - functions as a hardware lock separated from the CPU
  - denies unauthorized access to secrets even if the CPU runs a malicious application



# Defending against physical attacks

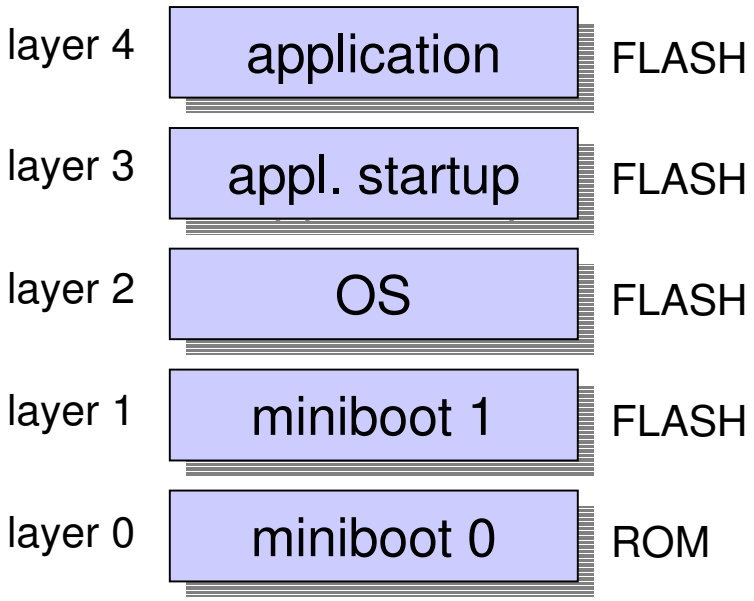
- the board is wrapped in a grid of conductors, which is monitored by a circuit that can detect changes in the properties of these conductors
- conductors are non-metallic and resemble the material in which they are embedded
- the grid is arranged in several layers
- entire package is enclosed in a grounded shield to reduce detectable electromagnetic emanations
- additional sensors:
  - temperature
  - humidity
  - pressure
  - ionizing radiation
  - changes in supply voltage and clock frequency
- reaction to tamper: erase BBRAM and reset the whole device
- physical security is certified at FIPS 140 level 4

# Device initialization

- the primary secret of the device is its RSA private key
- factory initialization
  - each device generates its on RSA key pair (using its own random number generator)
  - private key is kept in the BBRAM, public key is exported
  - external CA (manufacturer) certifies the public key by adding identifying information about the device and its software configuration and signing the certificate
  - device certificate is loaded back in the device
- regeneration of key pairs
  - the device generates a new key pair
  - signs a transition certificate with the old private key for the new public key
  - atomically deletes the old private key and activates the new one

# Code layers and organization of the BBRAM

- software is organized into layers
- device is shipped with miniboot 0 and 1
- OS and applications are loaded into the Flash memory by miniboot 1
- each layer has its own page in the BBRAM, where it can store its own secrets
  - device private key is stored in page 1
- the state controller ensures that code running at layer N cannot access pages that belong to lower layers



# Secure bootstrapping

bootstrapping sequence:

miniboot 0 → miniboot 1 → OS startup → appl. startup → application

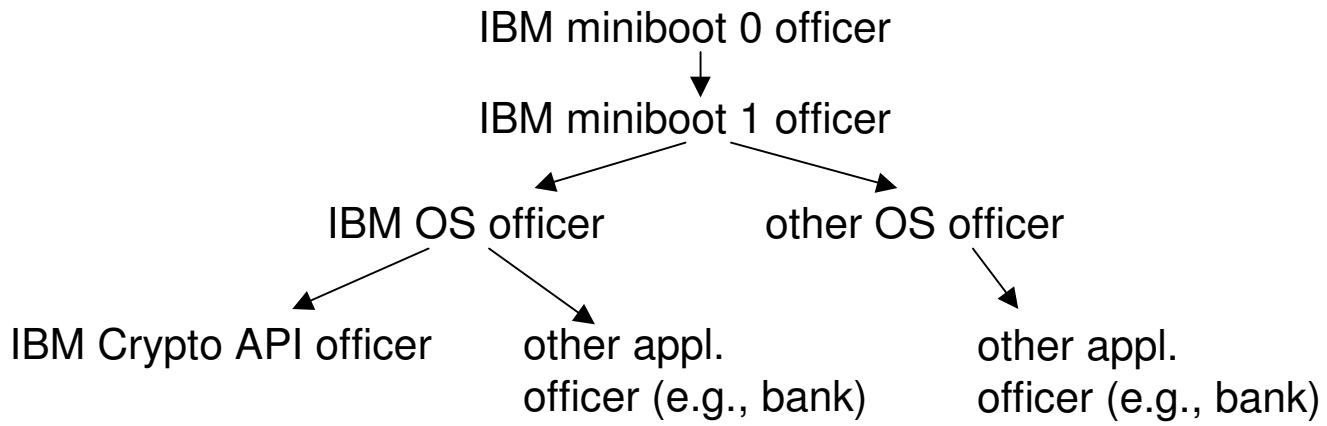
- after HW reset, the CPU starts miniboot 0 from ROM
- miniboot 0
  - runs self-test and evaluates the hardware needed to continue execution
  - checks the integrity of miniboot 1
  - advances the state controller from 0 to 1
  - and starts miniboot 1
- miniboot 1
  - runs self-test and evaluates the rest of the hardware
  - checks the integrity of OS and applications
  - advances the state controller from 1 to 2
  - and starts the OS
- OS
  - starts up
  - if needs to hide data from applications, then advances the state controller from 2 to 3 before invoking layer 3 code

# Code integrity

- problem: how to ensure that a malicious application cannot change the code of the OS and the miniboots?
  - the application can remove the integrity checks and the instruction to advance the state controller
  - next time the device is booted, miniboot 1 will not check the integrity of itself and upper layer codes, and will not advance the state controller
  - then, the application can read secrets of lower layers
- solution: the state controller prevents writing access to the Flash by the OS and the applications
  - all write accesses are denied when the state is greater than 1
  - only miniboot 1 can update software in the Flash !

# Code authorities

- loading of new software into a given layer is authorized by code authorities
- code authorities are organized into a tree
- each authority has a key pair
- parent certifies public key of its children
- miniboot 1 knows the public key of the miniboot 1 authority

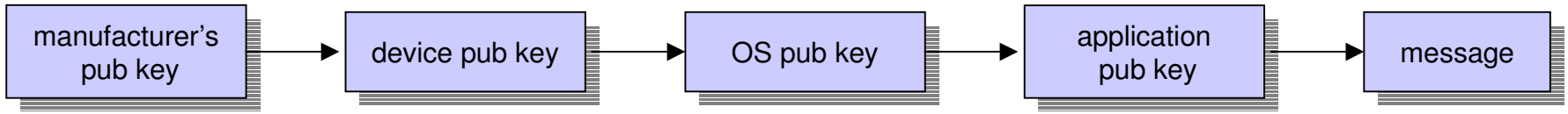


# Code loading

- code to be loaded is signed by the appropriate authority
- necessary certificate chain is also attached to the signed code
- miniboot 1 can verify the chain and the signature on the code
- if everything is correct, it loads the new code into the Flash

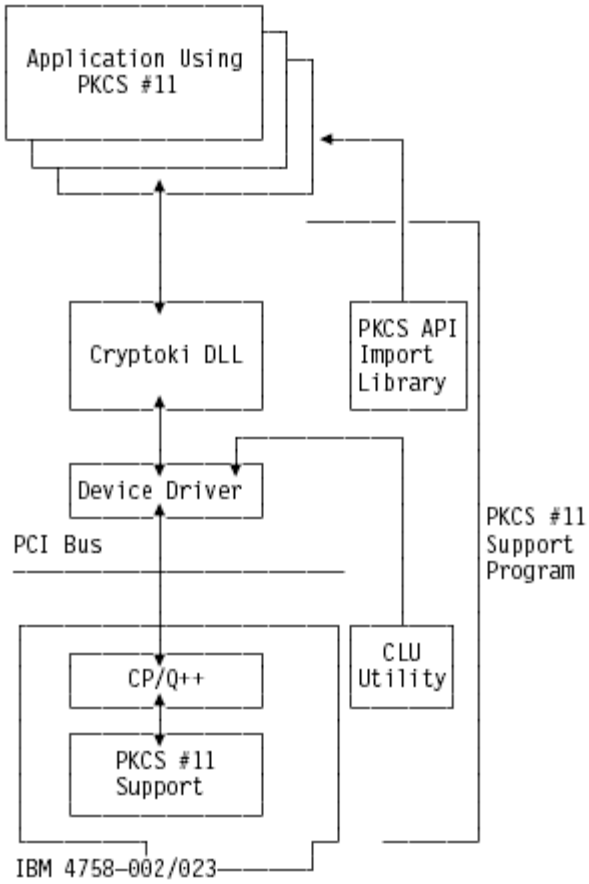
# Authenticating the execution

- problem:
  - how to distinguish between a message from an untampered device and a message from a clever adversary
- naïve approach:
  - device should use the device private key to sign messages
- problem with the naïve approach:
  - applications have no access to the device private key (stored in page 1 in the BBRAM)
- solution:
  - each layer N has a key pair, and its public key is signed by the private key of layer N-1
  - public key of layer 2 is signed by the device private key
  - application signs its messages with its own private key
  - signature can be verified with a chain of certificates starting from the device certificate



# CCA and PKCS#11 support

- the CCA and PKCS#11 support programs provides host-system software and coprocessor software that together allow the full exploitation of a 4758 device
- both CCA and PKCS#11 provide a convenient API to the cryptographic functions for application programmers



# Conclusion on high-end secure coprocessors

- advantages:
  - very high level of security
  - high performance
  - flexibility (loading and updating the OS and the applications)
  - layered trust model and code authorities may be conveniently mapped to the SeVeCom scenario
    - miniboot officer → security module manufacturer
    - OS officers → car manufacturers
    - application officers → car manufacturers, national authorities, third party software providers
- disadvantages:
  - price
    - the 4758 costs around 4000 dollars
  - battery lifetime
    - batteries need to be changed after 3 years
  - robustness
    - e.g., operating temperature: 10 – 40 C

# Conclusions

- we have presented two extremes of the spectrum of tamper resistant devices
- the solution for SeVeCom may lie between this two extremes
- how to find it ?
  - we must better understand the threats and the security requirements
  - we must determine the decision criteria
    - level of security provided
    - cost
    - …
  - we must look at what other people did in this field (e.g., GST)