Formal Methods in practice

When the standard meets the experience

Eduardo Giménez
At the edge of three axes

1. Trusted Labs’ experience on the use of Formal Methods

2. Application of the AIS34 standard (on Formal Methods)

3. Current EAL5+ evaluation of a Java Card Platform:
   - Open smart card compliant with Java Card & GlobalPlatform
   - Common Criteria v2.3 + AIS34
   - Augmentation with ALC_DVS.2 and AVA_VLA.4
   - ST drawn from Sun’s Java Card Protection Profile
Summary

1. FM applied to smart cards software: Trusted Labs’ experience
2. The AIS34 approach to FM
3. Preparing a certification based on AIS34
4. Keynotes for addressing hard issues with Formal Methods
5. The AIS34 standard in CC v3.1
What does Formal Methods mean?

- A scientific approach of programming:

- Developing a mathematical model of a piece of software in order to predict (anticipate) its behavior.

- Rigorous reasoning: it is a deductive approach close to Logic and Mathematics which complement empiric approaches (testing).
Customer motivations for using FM

- Acquiring a better understanding of a given technology.
  - Java Card Technology
  - Open smart card devices (GlobalPlatform)

- Improving confidence on new, high-tech solutions.
  - Bytecode verification as a security shield
  - Designing new versions of public standards
Formal Methods: the usual picture

- Informal Specifications of a public standard
- Formalize behavior
- Functional Model
- Consistency Proof
- Correspondence Proof
- Formal Low Level Design of a particular implementation
- Expected Properties
- Formalize statements
- Security or Functional Objectives
The scheme applied to GlobalPlatform

GlobalPlatform’s Card Implementation Requirements

An abstract state machine \((S,R,I)\) processing each APDU command

A function \(f(cmd, s_1) = s_2\) computing the state after executing \(cmd\) on \(s_1\)

\[ \forall s \in S, R(s,f(s)) \]

Formalize behavior

Consistency Proof

\[
\forall s \in \text{reachable } s \rightarrow \exists a, a \in \text{applets } s, \quad a.\text{SelectDefault} = \text{true} \land \forall a' \neq a \quad a'.\text{DefaultSelected} = \text{false}
\]

Correspondence Proof

\[ \forall s \in S, s \rightarrow \exists a, a \in \text{applets } s, \quad a.\text{SelectDefault} = \text{true} \land \forall a' \neq a \quad a'.\text{DefaultSelected} = \text{false} \]

Formalize statements

“There is always one and only one applet instance that has the Default Select Privilege”

or

“Any command sent through a secure channel of level MAC open in a previous session cannot be replayed in the current session”
Design and implementation errors

Informal Specifications → Formalize behavior

Informal Specifications → Correspondence Proof

Functional Model → Consistency Proof

Expected Properties → Design error!

Implementation error!

Formal Low Level Design

Security or Functional Objectives → Formalize statements
Formalization errors

Misunderstanding!

Informal Specifications → Formalize behavior

Formalize behavior → Correspondence Proof

Correspondence Proof → Functional Model

Functional Model → Consistency Proof

Consistency Proof → Expected Properties

Expected Properties → Formalize statements

Formalize statements → Security or Functional Objectives

Informal Specifications → Formal Low Level Design

Formal Low Level Design → Misunderstanding!
FM applied to smart card software: state of the art

• Currently at hand: validating specifications
  – Technically feasible for present state of the art in formal methods.
  – Required for further validations (implementation errors).

• Going further: searching for implementation errors:
  – A much more expensive task (needs to go into all the details).
  – A gap between the code model and the real embedded code.
  – Frequently, code evolves too fast for keeping proofs up-to-date.
  – Very few complex algorithms in smart card’s world
    • Counter-example: on-card bytecode verification
Applications Notes and Interpretation
Scheme: AIS34

- CEM: provides methodology for EAL1 to EAL4 (no FM)

- AIS34: extends the CEM to EAL5 (formal SPM)

- Defines minimal evaluation effort for achieving an EAL5 evaluation and provides guidance on ways and means of accomplishing the evaluation.
EAL5 in Common Criteria specs (ADV_SPM)

ADV_SPM.3.3C
Formalize behavior

ADV_SPM.3.4C
“Consistent and complete”
(weak sense)

ADV_SPM.3.2C
“Characteristics”

ADV_SPM.3.2C
“Rules”

EAL5 = there is a formal model
(no ambiguity)

Informal Specifications,
Security Policies in the ST

Functional Specification
EAL5 in the AIS34 approach

EAL5 = validating design errors in the ST
When the standard meets the experience…

AIS34 approach matches our past experience and the current state of the art in Formal Methods

But this approach has consequences…
Consequences of the AIS34 interpretation

Significant impact on evaluation schedule and workload

- Developing a formal model of a smart card has become technically feasible in a reasonable delay (a few month/men).

- Developing formal proofs still requires a significant amount of work!
  - Proving the 32 security objectives of the JCSPP mean several year/men
  - Could be incompatible with time-to-market and product lifetime constraints.

- Possible improvements: assign the security objectives to be proven as part of ADV_SPM requirements
Formal Methods in practice...

Preparing a CC evaluation based on AIS34
Defining the scope of the formal model

Trusted Labs’ claim: from a technical point of view, there is no limit on what can be modeled or proven.

- AIS34: abstention from formal modeling of relevant TSP requires argumentation.

- Only true parameters limiting the scope:
  - The proof environment chosen for developing the model
  - Reasonable timeframe and workload for a CC certification

- Trusted Labs’ proposal: let us focus on challenging properties!
  - Example: complicated collection of positive and negative access control rules.
  - Counterexample: PIN verification prevents more than 3 unsuccessful authentication attempts.
Security Target writing

Think about FM from the very beginning of the project

- Each security objective in the ST has to be proven as a formal property.

- State them as high-level demonstrable assertions:
  - 😃 Each TOE sample shall be uniquely identified by a serial number.
  - 😞 The IC serial number shall be stored in EEPROM.

- Concentrate on the essential, be precise, avoid introducing superfluous objectives or low level details that could complicate the rational with the SPM.

- An issue: sometimes the statement comes from an (unadaptable) PP.
Selecting the proof environment

- It is not possible to do formal methods without the support of a proof environment.

- Several proof environments exist:
  - Proof assistants: Coq, HOL, Isabelle, PVS, ... (logical frameworks)
  - Static analysis tools: ESC/Java, VeriSoft, ... (code analysis)
  - Hore Logics: Atelier B, ... (code refinement)
  - Model checkers: Kronos, SPIN, ... (finite state models)
  [See for instance the 95 references at http://vl.fmnet.info/]

- Some of them are better suited for preventing design errors (proof assistants), other are better suited for preventing implementation ones (static analysis tools, Hore Logics).
Trusted Labs choice: Coq proof assistant

- Strong specification language that enables to:
  - Quickly modeling any kind of system and properties;
  - Easily tuning the required level of abstraction.

- Proof strength:
  - Important library of lemmas and theorems.
  - User driven, extensible language for tactics.
  - May be connected to automatic proof engines, if necessary

- Provides a trace that explains why the theorem is true (i.e., its proof):
  - Model and proofs soundness only depends on a small kernel (Coq’s proof-checker) that has been (itself) formally verified.
  - Internal consistency for free
  - Maintenance: proofs are self-documented and can be reviewed
Selecting the FM team

Not only Logicians, not only engineers...

- Developing high quality formal models still require FM experts.
- Most of them come from research, and are not always available.
- Very few combines both FM, software engineering and domain-specific skills.

- Setting up a good communication channel with the dev team is a crucial point.
  - Prevents getting stuck or missing important points because you lack of information about the actual behavior of the TOE.
  - Requires providing them with some feedback (FM is not useless!)
Placing FM in the TOE life cycle

Use Formal Methods as (yet another) validation tool.

- Specification
- Design
- Implementation
- Testing
- Modeling Specs
- Proving objectives
- Review
- Feedback

Ambiguities:
- Hypotheses
- Limit cases
- Design errors
Formal Methods in practice...

Some hard issues with Formal Methods
Find the right abstraction level

- Too high: dangerous situations could be ejected from the model.
  - Example: representing an operand stack as a potentially infinite list amounts to neglect overflow attacks.

- Too low: if a given implementation is cabled in the specifications, proofs become too complicated because of uninteresting details.
  - Example: detailing the error code returned under each situation is useless if the TOE does not further process them.

- Good compromise usually result from good communication between development and formal methods teams.
The SPM is an abstraction of the real TOE.

It frequently helps to makes explicit some hidden hypotheses required for the security properties to hold.

But could also contain implicit ones...

Example:
- Smart card = abstract state machine that processes APDU commands.
- Transition relation $s_1 \rightarrow^c s_2$.
- The model implicitly assumes atomic command processing.
- What happens if the card is withdrawn during command processing?
Formal Methods require a sustained effort

- Maintaining the models across new versions of the product.
  - Mathematics: prove it and forget it.
  - Industry: prove it and step ahead to the next TOE version.

- Proofs has to be maintained too! (proof engineering)
  - Organize the formal model into separate modules
  - Define interfaces for each module
  - Implement tactic libraries to keep as much abstract as possible
  - Document definitions and proofs
  - Prove the most general theorems for maximal reuse
  - Etcetera...

- This requires software engineering (and FM) skills!
AIS34 and CC V3.1

- CEM v3.1 still leaves ADV_SPM guidance to the schemes.

- Does not contradict the idea of EAL5 as verifying the ST design (proving security properties from characteristics).

- Drawback: implicitly considers that the security properties are always "invariants" of a state machine (too restrictive).
  - Example: *a command cannot be replayed in a future card session*

- Mentions internal consistency as an application note, but does not require it as an action element.
AIS34 for CC V3.1: a possible improvement

- Assign the security objectives to be proven as part of ADV_SPM requirements

  - **ADV_SPM.1.1D** The developer shall provide a formal security policy model for the [assignment: *list of policies that are formally modeled, identifying the relevant portions of the statement of SFRs that comprise each of the modeled policies*].

  - **ADV_SPM.1.xD** The developer shall provide a formal proof for the [assignment: *list of security properties that are formally proven, identifying the relevant portions of the statement of security objectives that comprise each of the proven properties*].
The End
Questions?