Model-Based Security Verification and Testing for Smart-Cards

M. Ochoa

F. Bouquet

J. Bottela

E.Fourneret

J.Jurjens

P. Yousefi

ARES 2011

technische universität dortmund

UFC UNIVERSITÉ DE FRANCHE-COMTÉ

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

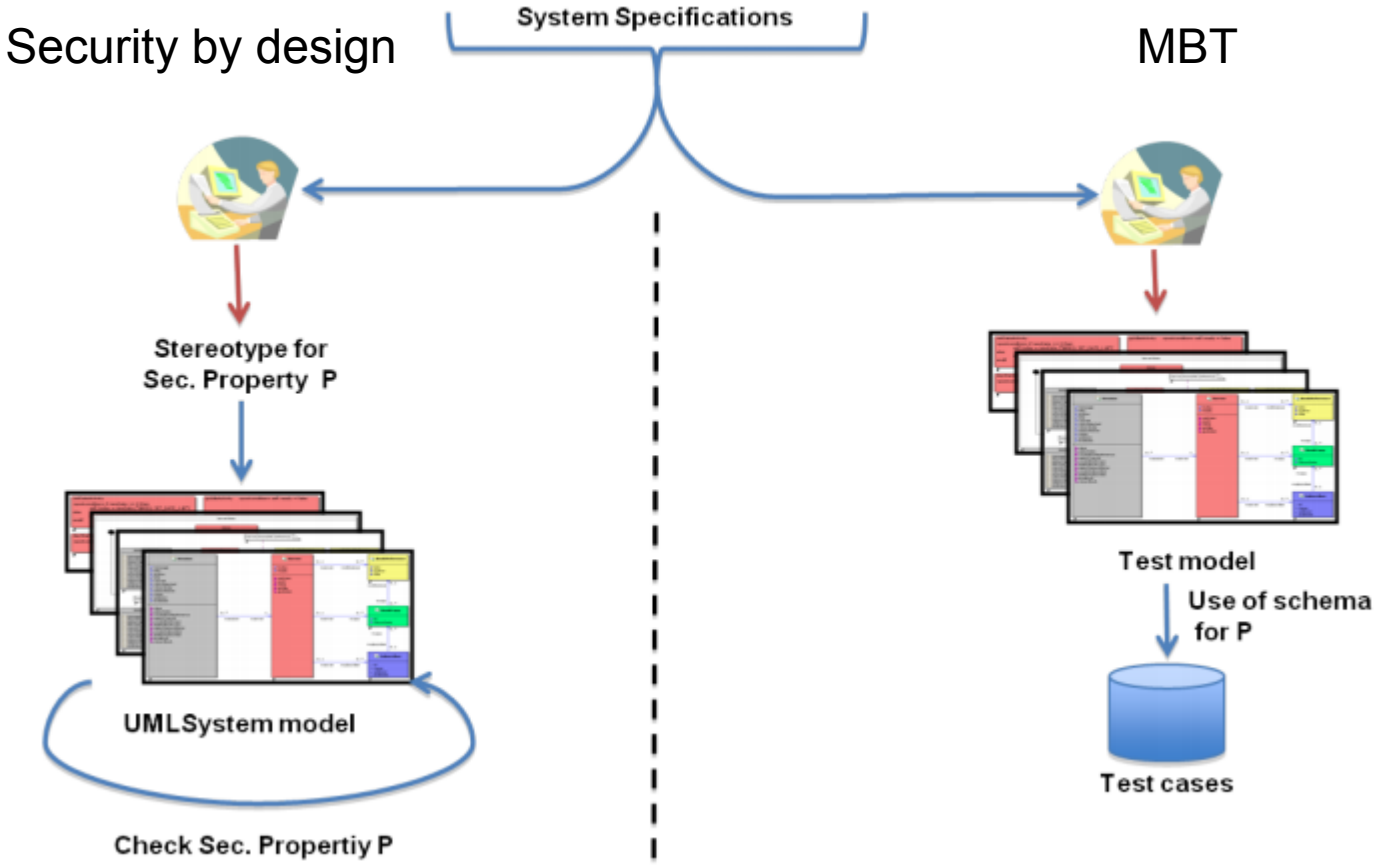
INRIA centre de recherche NANCY - GRAND EST

smartesting® Optimize your Test Center

OUTLINE

- **Introduction**
- **Background**
  - UMLsec
  - Model-based Testing
- **Security in smart-card life-cycles**
- **Correct security testing**
- **Validation**

Research question?



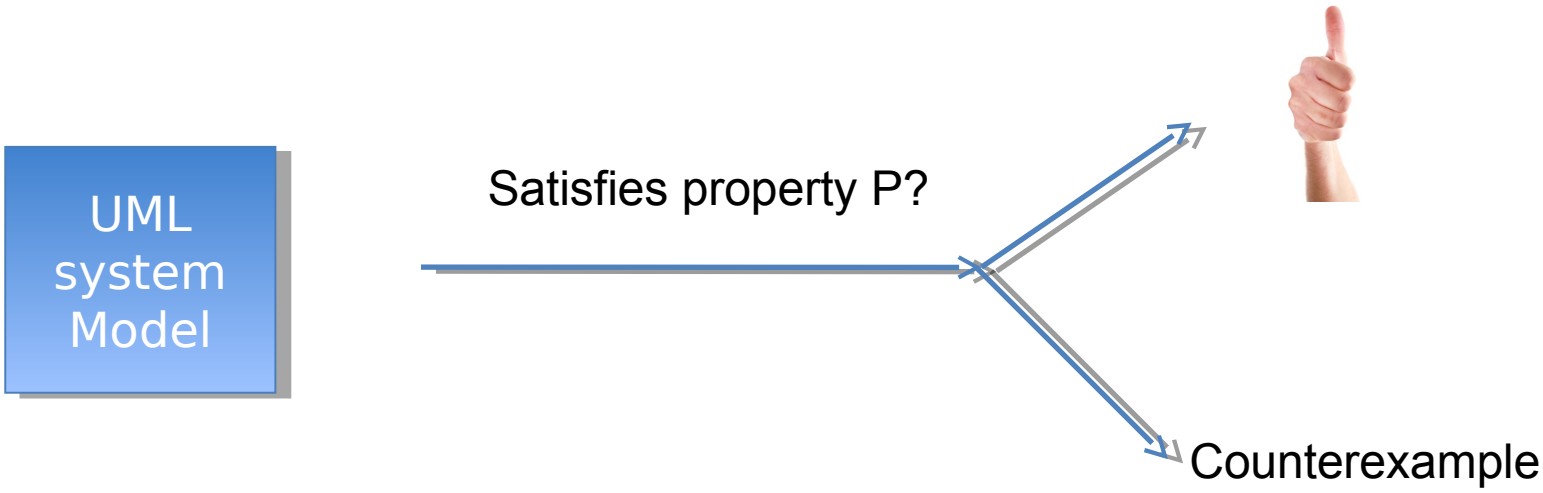
Can we unify these two approaches? (to some extent)

Background: UMLsec

UMLsec is a lightweight extension of UML by means of stereotypes and tagged values.

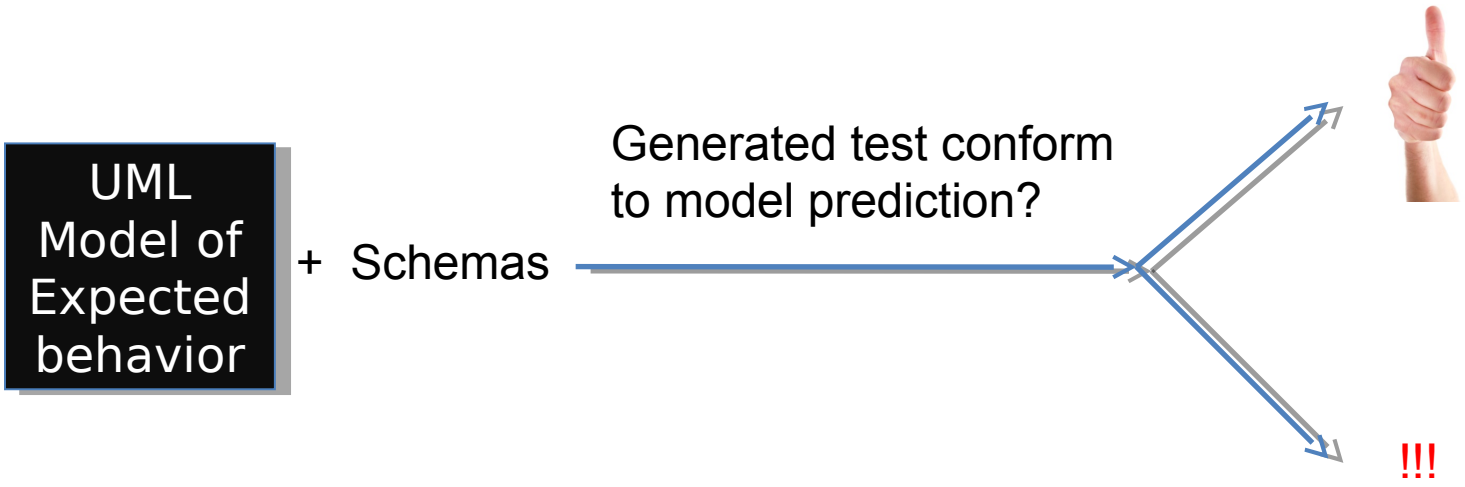
- Formally well-founded (based on a formalization of a fragment of UML).
- Supports a collection of different security verification techniques across UML system views.
- Tool supported.

Background: UMLsec



UMLsec extends Class, Sequence, Activity, Statechart and Deployment diagrams and allows to verify Dolev-Yao cryptography, Non-interference and RBAC among others. There is tool support for most of the UMLsec stereotypes.

Background: MBT with Schemas



- Automated test generation from security properties to testing needs using schemas
- Ensure security property coverage
- Traceability between generated tests and security property

Background: MBT with Schemas

Schema Language: Allows a straightforward, imperative-programming-like definition of Test Schemas, from which automatic test sequences can be generated. For example:

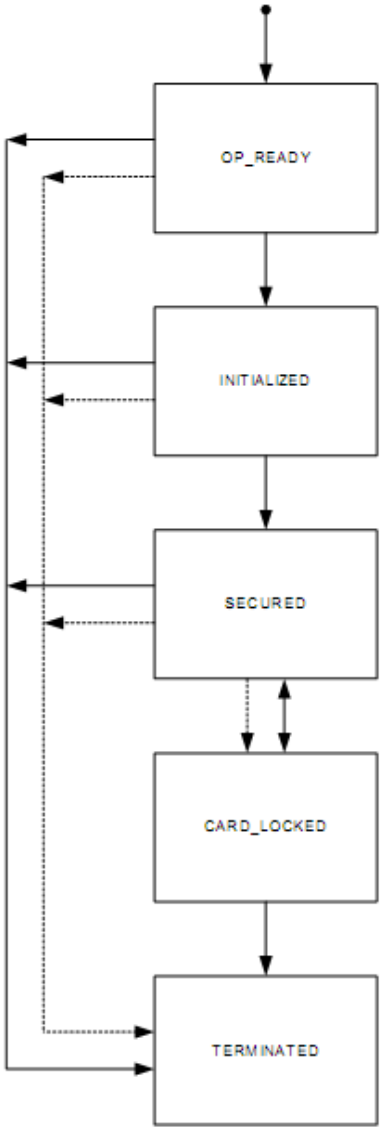
```
for_each $X from S,
use any_operation any_number_of_times
to_reach state_respecting T
on_instance 'chosen_instance' then
use $X at_least_once to_reach state_respecting Q
on_instance 'chosen_instance'
```

Security in Smart-card Life-cycle

A smart-card has typically a well-defined life-cycle, that ranges from pre-deployment to active and eventually to a locked-status or a terminated status where is not possible to use the card any more.

Security in Smart-card Life-cycle

Example: Global Platform Specification v 2.1.1 on the Card Life Cycle Scope



Security in Smart-card Life-cycle

Natural security requirements on the life-cycle to prevent D.O.S attacks:

Security Property 1: *For any execution, whenever the card is set to the state TERMINATED by means of a operation performed by a privileged application, then it should not be possible to revert to another state.*

Security Property 2: *It should not be possible for an application that does not have the given privilege to set the card into a given state TERMINATED.*

Correct security testing

Main ideas:

- Is the expected behaviour, as described by the models in MBT already trivially violating the security properties to be tested?
- Can we improve the quality of the MBT by using the UMLsec philosophy?
- Can we also automatically generate schemas from this analysis?

Correct security testing

UMLsec new stereotypes for Security Properties 1 and 2 on statecharts:

<<locked-status>> together with tag {status} specifies a status (node) in the statechart that should not have outgoing transitions to other nodes.

<<authorized>> together with tags {status} and {permission} checks that all transitions to a given node contain a given permission check in their guard.

Correct security testing

From the testing perspective, we can express the security properties as hoare triples {P} S {Q} where P and Q are FOL formulas quantifying over system variables and S is a set of system commands. For example:

Locked-status: $\{\text{state} = \{\texttt{status}\}\}\ \text{set\_status}\ \{\text{state} = \{\texttt{status}\}\}$

Authorized: $T := \text{state} \neq \{\texttt{status}\} \wedge \{\texttt{permission}\} \notin A.\text{permissions}$

$$Q := \text{statusWord} = \text{Error\_not\_Privilege\_}\{\texttt{permission}\}.$$

$$\{T\}\ \text{set\_status}\ \{Q\}$$

Validation

Automatically verified some violating models of the GP v 2.1.1 card life-cycle

Validation

Generated schema:

```
for_each $X from APDU_Set_status,
use any_operation any_number_of_times to_reach
state_respecting (self.lcs->exists(lc : LogicalChannel|
lc.selectedApp.privileges.cardTerminate=false))
on_instance ”card” then
use any_operation any_number_of_times
to_reach state_respecting (self.state!=TERMINATED)
on_instance ”card” then
use $X at_least_once to_reach
state_respecting
(self.StatusWord =
APDU_SETSTATUS_ERROR_MustHaveTerminatePriv)
on_instance ”card”
```

Using the schema we have generated13 tests.

CONCLUSION AND FUTURE WORK

- Take into account the evolution aspect i.e
  - Specification can evolve thus the model and/or
  - The security property can evolve
- Schema language extensions
- Methods and tools' evaluation on other systems and for other security properties.
- Integration of tool support

Questions?