Model-based Security Engineering

Jan Jürjens

TU Dortmund & Fraunhofer ISST

Dortmund, Germany

Secure IT-Systems

Today IT-systems pervade almost all aspects of human life. At the same time, IT-systems become more open and therefore more vulnerable.

A lot of successful academic research has been done on foundations for secure systems. Some milestones:

- Saltzer, Schroeder: Protection of Information in Computer Systems, 1975
- Gasser: Building a Secure Computer System, 1988
- Burrows, Abadi, Needham: A Logic for authentication, 1989
- Gasser, Goldstein, Kaufman, Lampson: The Digital Distributed System Security Architecture, 1989
- Ross Anderson: Security Engineering, 2001

Unfortunately, despite this successful research, today's systems still often do not satisfy the increasing expectations on their security requirements - ...

Problems

„Blind“ use of mechanisms:

- Security often compromised by circumventing (rather than breaking) them.
- Assumptions on system context, physical environment.

„Those who think that their problem can be solved by simply applying cryptography don`t understand cryptography and don`t understand their problem“ (R. Needham).

Problem: Security is Elusive

- Classical weakness in old Unix systems: “wrong password” message at first wrong letter in password. Using timing attack, reduce password space from 26^n to 26*n (n = password length)
- More recent weakness on smart-card: reconstruct secret key by timed measurement of power consumption during crypto operations

➔ Difficult to find such weaknesses using classical testing.

Special Problem: Crypto

- Cryptography plays important role in many security-critical applications
- By definition, needs to be secure against brute-force attacks

➔ Paradox: How do you get sufficient test coverage (for inputs accessible to a given attacker) of a system that needs to be secure against brute-force attacks on that input ?

This lecture: Holistic view on Security

„An expansive view of the problem is most appropriate to help ensure that no gaps appear in the strategy“ (Saltzer, Schroeder 1975). But „no complete method applicable to the construction of large general-purpose systems exists yet“.

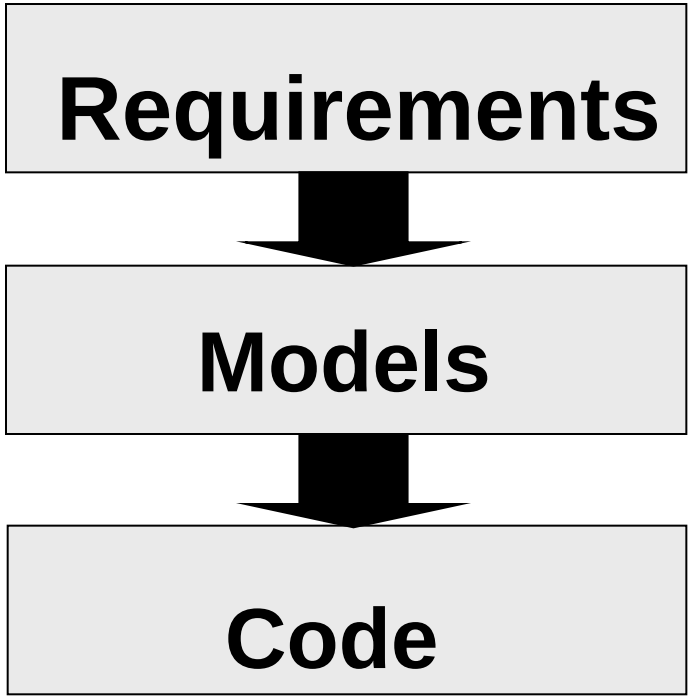
Goal: Integrate well-founded approaches for security analysis and design into practically usable (and used) software engineering.

Model-based Development

Idea: Build on model-based software development.

General goal: facilitate transition from human ideas to executed systems.

Increase quality with bounded time-to-market and cost.



Security Engineering

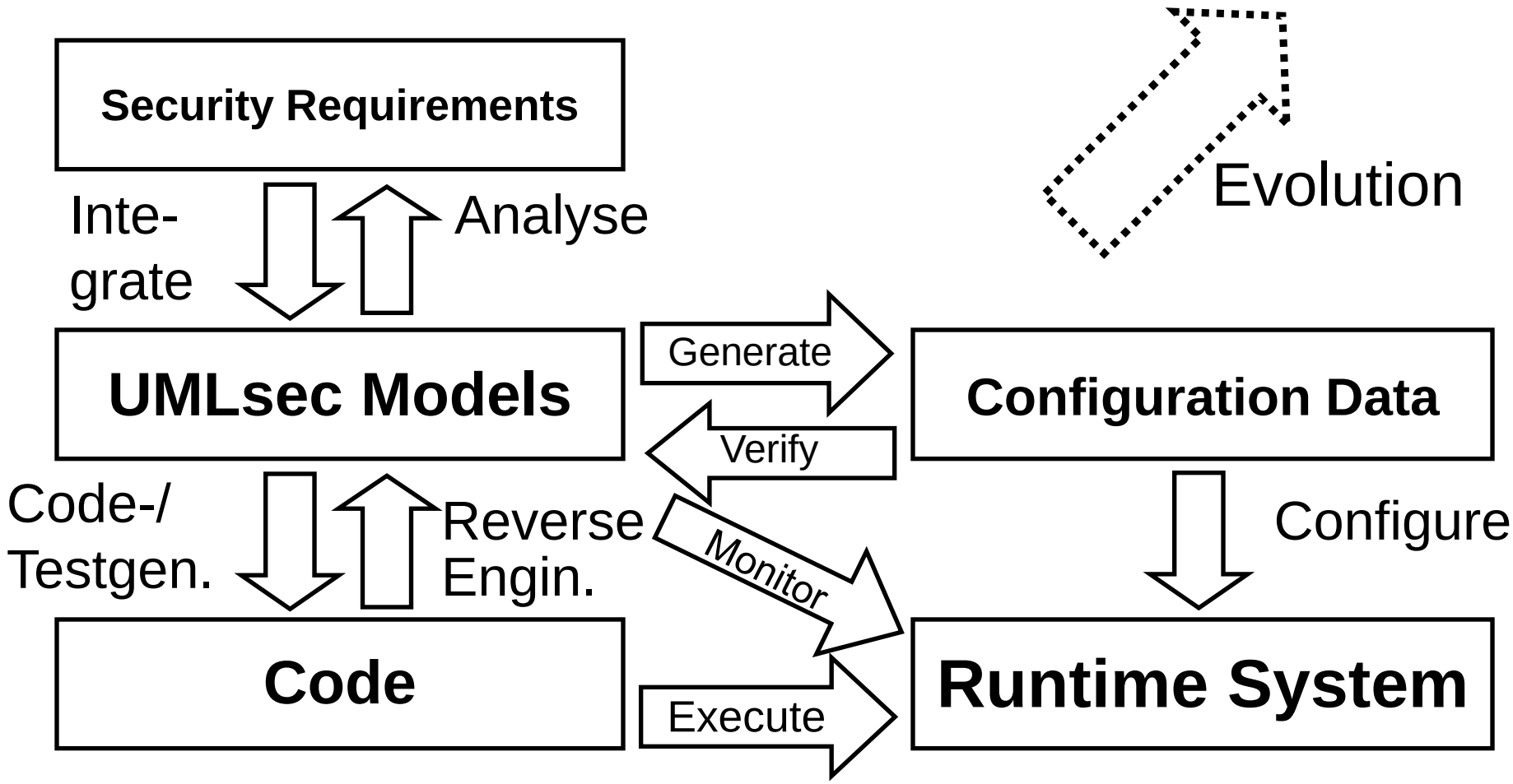
Increase security with bounded investment in time, costs (crucial for industry). Idea:

- Extract models from artefacts arising in industrial development and use of security-critical systems (UML models, source code, configuration data).

Tool-supported theoretically sound efficient automated security analysis.

➔ *Model-based Security Engineering*

Model-based Security Engineering with UMLsec



Goal: Security by design

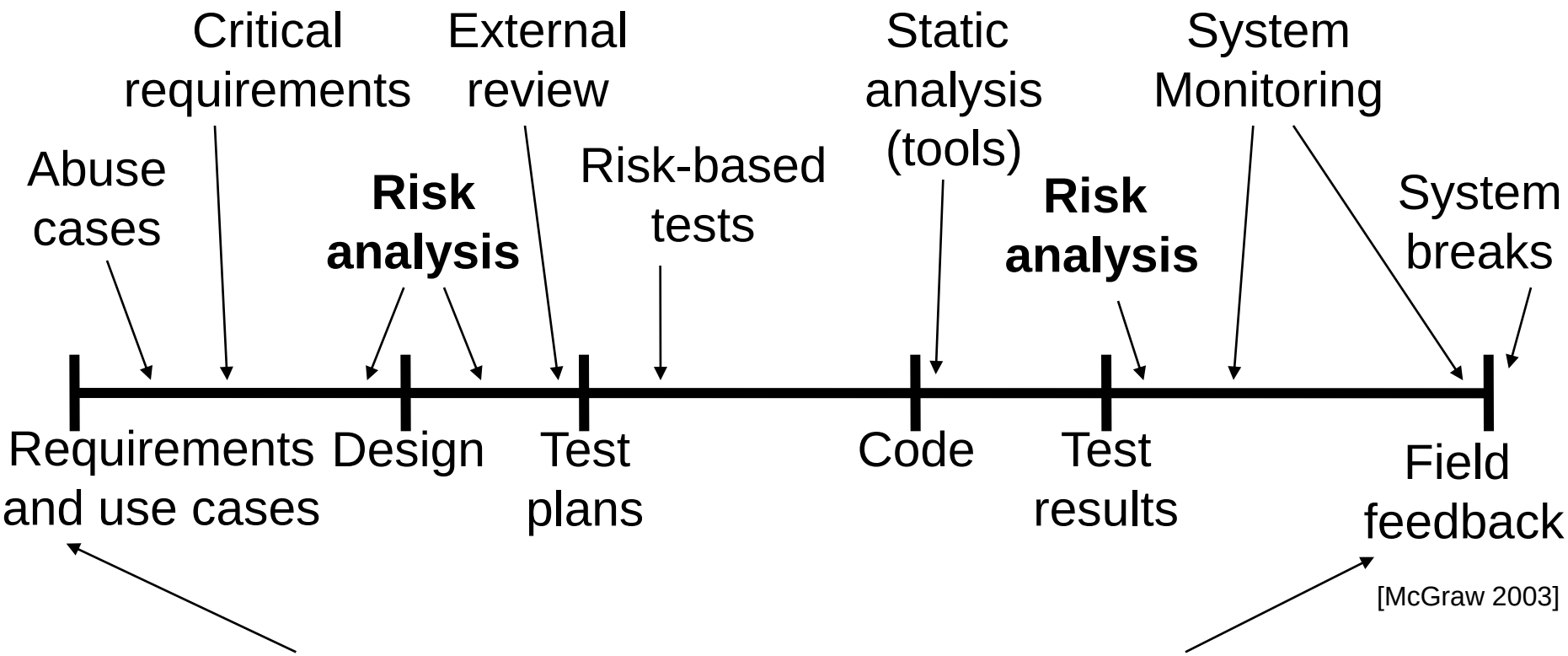
Consider security

- from early on
- within development context
- taking an expansive view
- in a seamless way.

Secure design by model analysis.

Secure implementation by test generation.

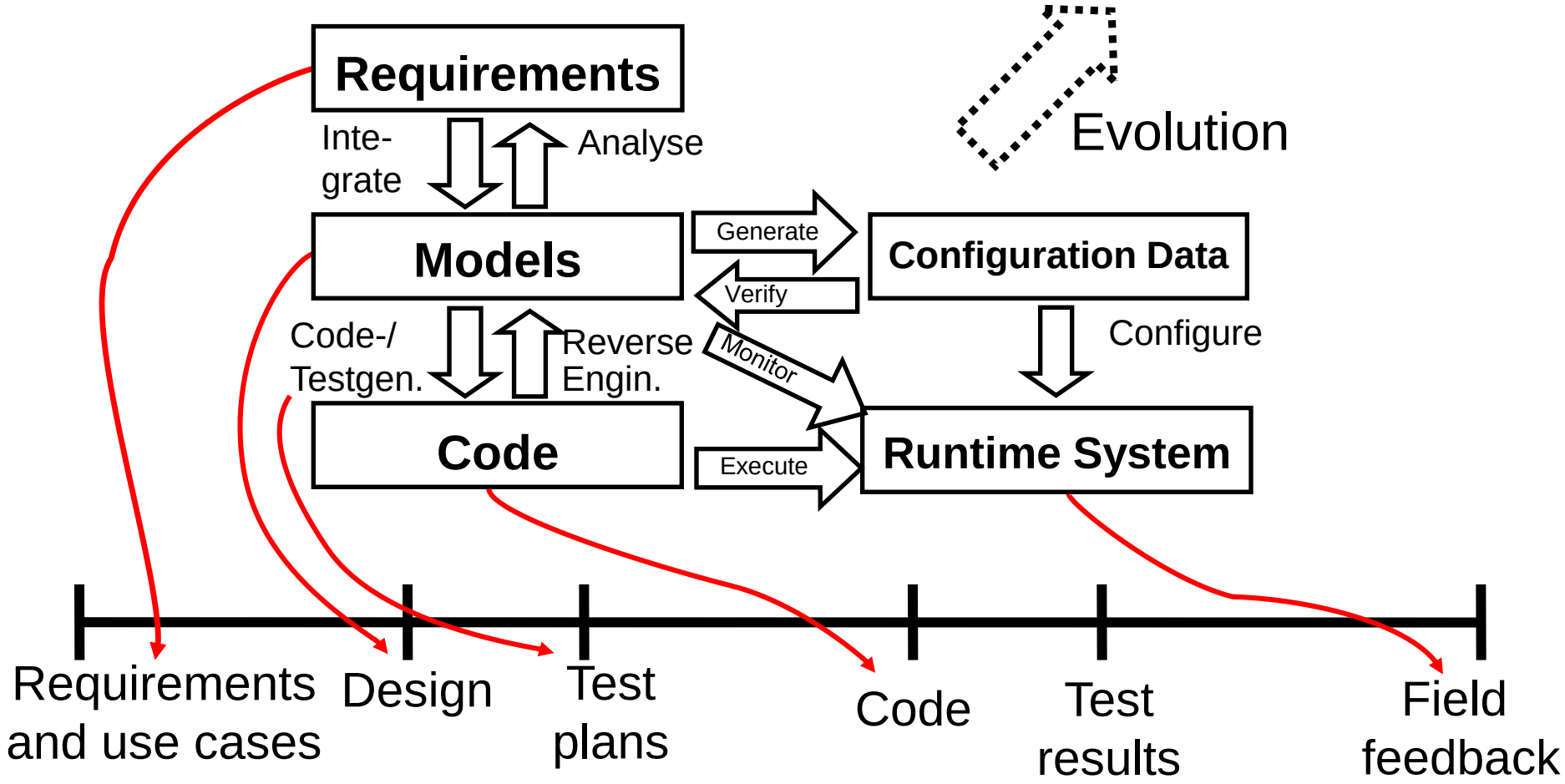
Critical System Lifecycle



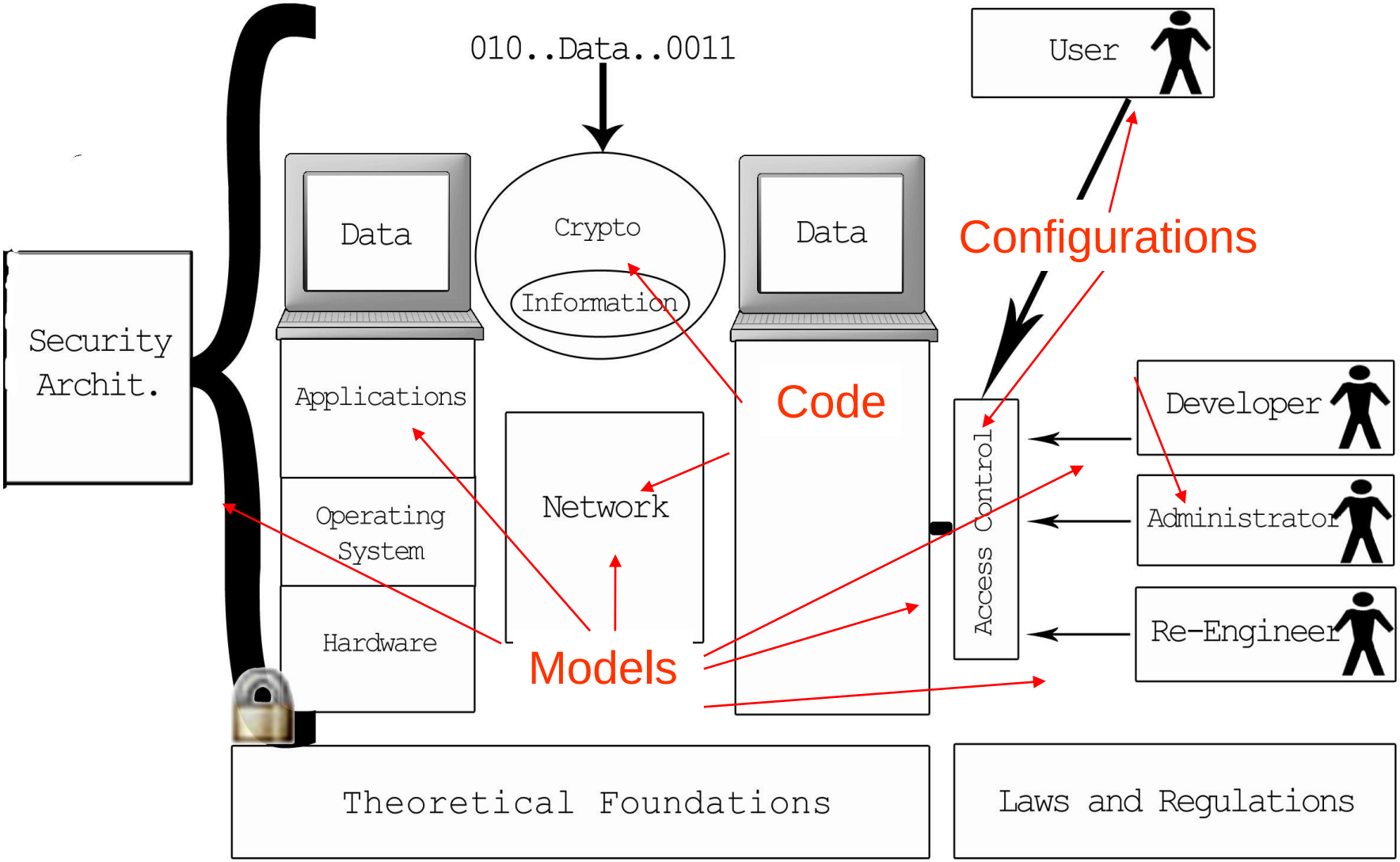
[McGraw 2003]

Model-based Security Engineering

Goal: Integrated Approach



Architectural Layers



Why UML ?

Seemingly de-facto standard in industrial modeling. Large number of developers trained in UML.

Relatively precisely defined (given the user community).

Many tools in development (also for code-generation, testing, reverse engineering, simulation, transformation).

Using UML

Goal: transport results from formal methods to security practice

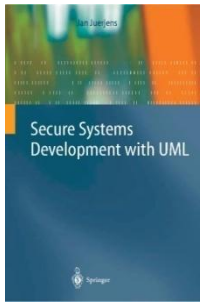
Enable developers (not trained in formal methods) to

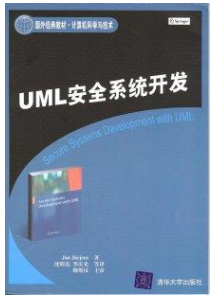
- check correctness of security designs
- deploy security mechanisms correctly in system context
- allow to analyze larger system parts beyond crypto-protocols

UMLsec: Goals

Extension for secure systems development.

- evaluate UML specifications for weaknesses in design
- encapsulate established rules of prudent secure engineering as checklist
- make available to developers not specialized in secure systems
- consider security requirements from early design phases, in system context
- make certification cost-effective





Jan Jürjens: Secure systems development with UML. Springer 2005. Chines. transl. 2009

UMLsec: How

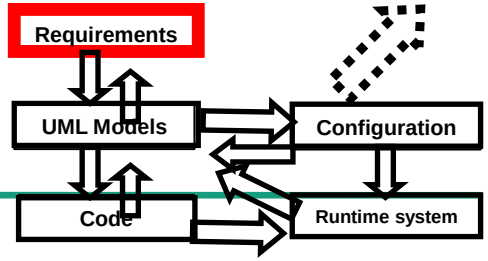
Recurring security requirements, adversary scenarios, concepts offered as stereotypes with tags on component-level.

Use associated constraints to verify specifications using automated theorem provers and indicate possible weaknesses.

Ensures that UML specification provides desired level of security requirements.

Link to code via round-trip engineering etc.

Security Requirements Engineering



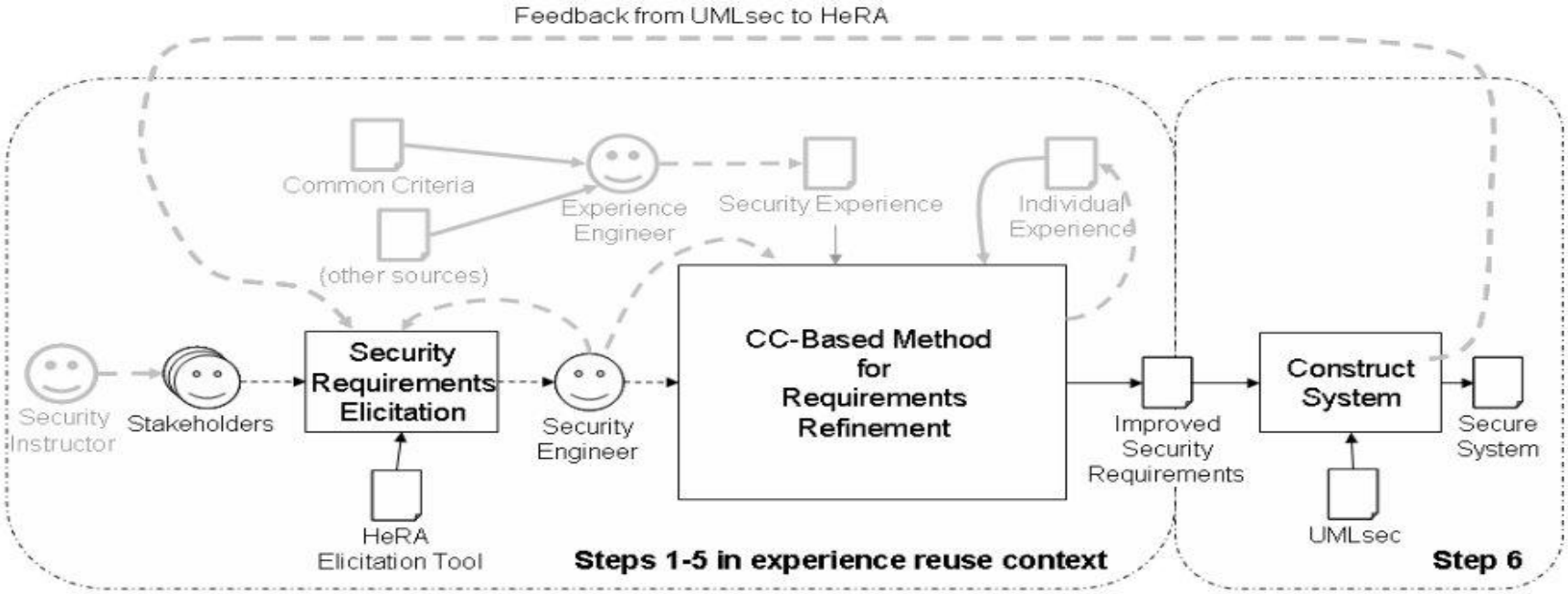
Aims:

- Identify security requirements within the requirements elicitation.

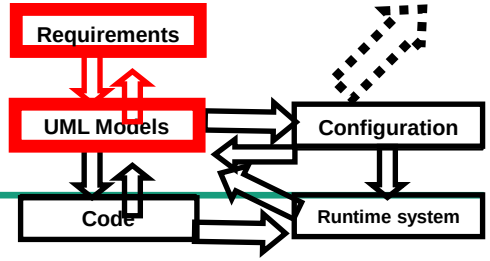
Idea: “Requirements Mining” in security standards (e.g. Common Criteria) resp. in the given specification document

Validation example: IPTV Standard of Eur. Telecom. Stand. Inst. (ETSI)

[CAISE '06, Requirem. Engin. Jour. '10, Journ. Softw. & Systems Modeling '10]

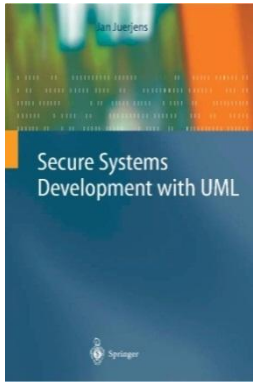


Modeling with UMLsec



Aim:

- Documentation and automated analysis of security-relevant information (e.g. security properties and requirements) as part of the system specification.

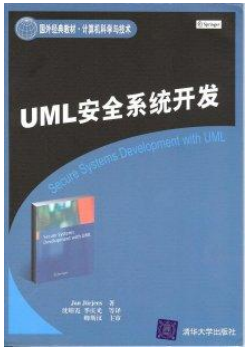


Idea:

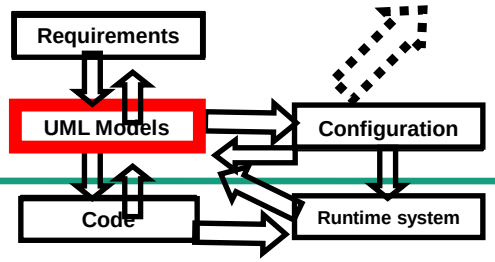
- UML for system modeling.

[FASE 01, UML 02]

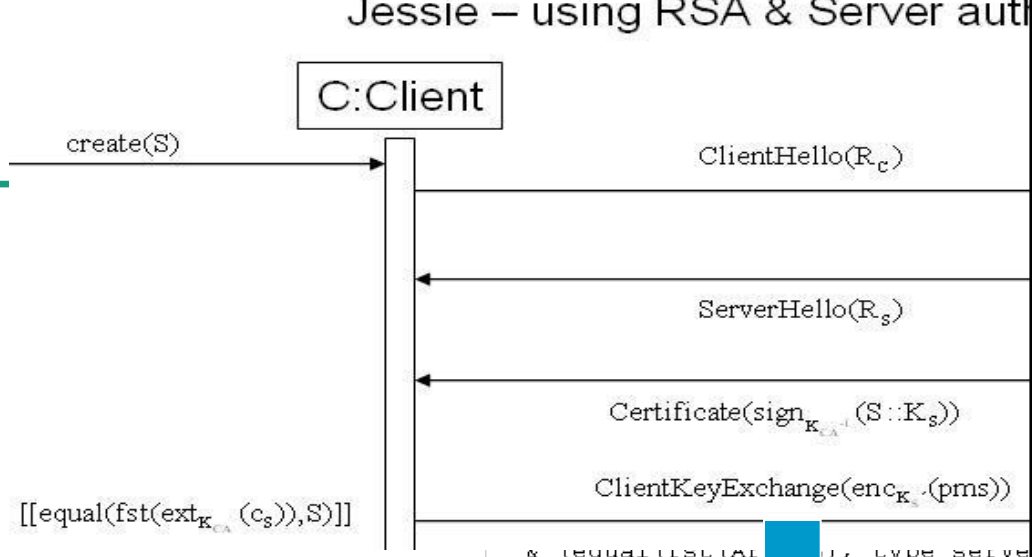
- Insert security-relevant information as stereotypes provided by UML-extension UMLsec.
- Formal semantics based on stream-processing functions as a foundation for verification.



[Jour. Logic & Algebr. Program. '08]



Model-based Security Analysis



Aim:

- Automated analysis of the system models against the sp requirements.

Idea:

- Automated generation of logical formulas in first-logic (or LTL, ...) based on formal semantics for security analysis. Transfer to the automatic theorem prover (or modelchecker/...)).

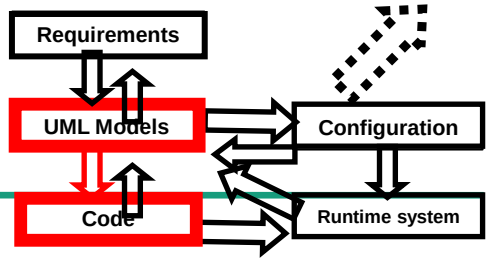
```
  & (equal(snd(ex  d(snd(ArgC_3)
  & (equal(snd(ext nd(ArgC_2), k_
)
=>(
  ((knows(ArgC_4_1)
    & equal(ArgC_4_1, type_serverh
    =>(
  ( ( true & equal(ClientKeyExchan
)
...
%------------------------

input_formula(attack,conje
   knows(mastersecret) )).
```

```
analyzing results ...
model found/total failure
time limit information: 19 t
(leaving wrapper).
task myUML_PID1491 on atbroy
(model found by strategy 300
deleting temporary files.
e-SETHEO done. exiting
```

[ICSE 05, ICSE 06]

Model-based Security Testing



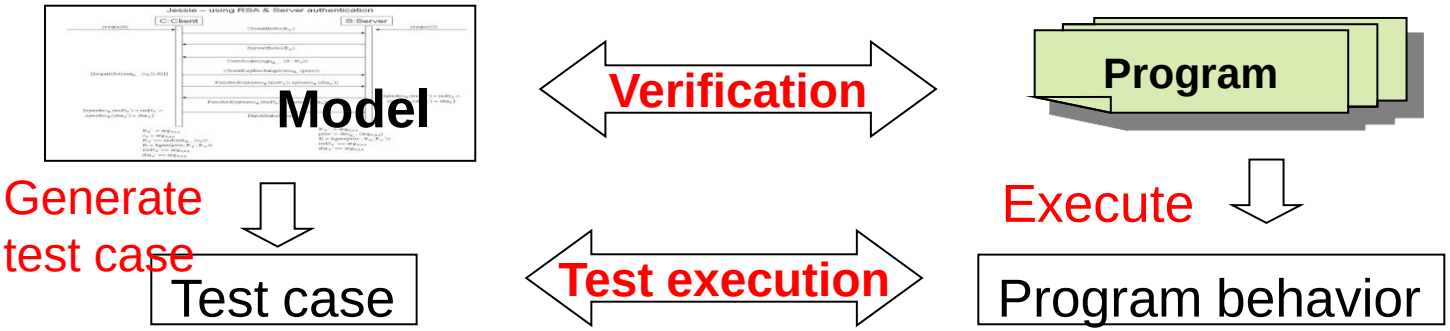
Problems with using conformance-tests for security:

- In general, complete test coverage impracticable.
- Finds only attacks which are visible on the model level.

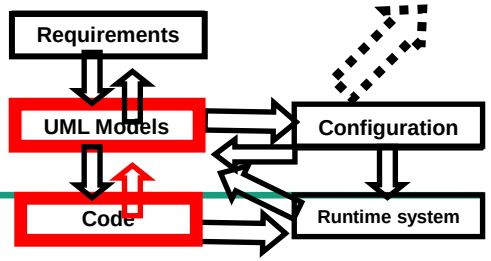
Idea: Mutation-testing.

- Focus on critical test cases
- Finds also weaknesses which are not visible on the model level.

Validation: Common Electronic Purse Specifications.
Detected several weaknesses.



Static Program Analysis



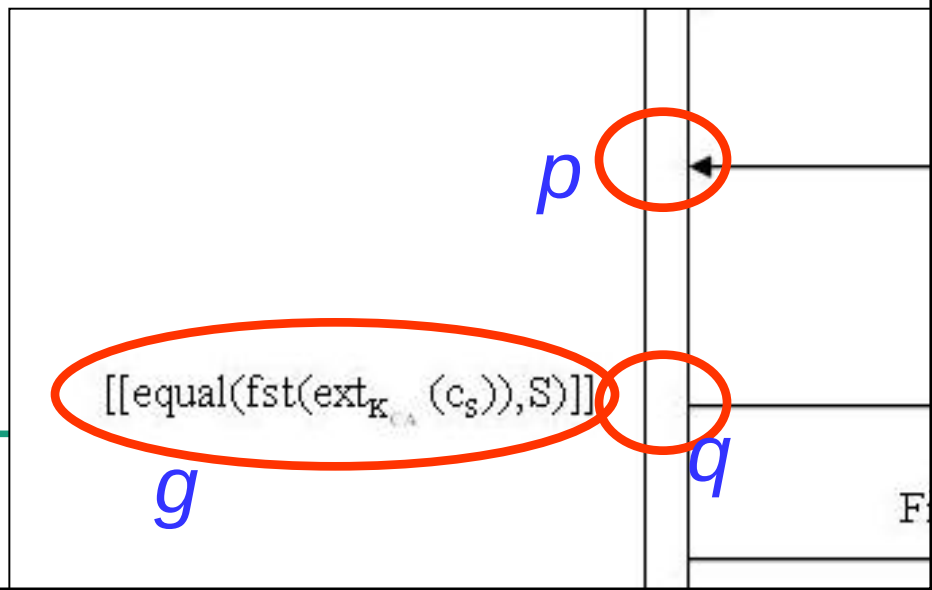
Problem: Correct use of cryptography is inherently difficult to test: sufficient test coverage amounts to brute-force attack.

Idea: Automated, formal static program analysis of correct cryptographic function calls (with ATP for FOL).

Validation: Java Secure Sockets Extension (JSSE).

Current project Csec:
C code analysis.

[ICSM 05, ASE 05, ASE 06]

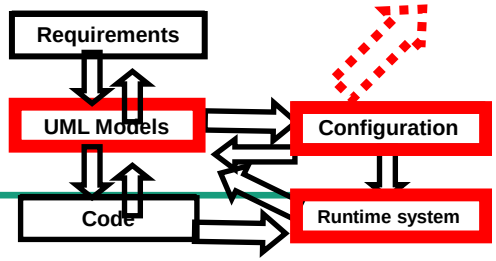


Microsoft Research

THE ROYAL SOCIETY
CELEBRATING 350 YEARS

technische universität dortmund

Security Analysis of Configuration Data



Aim: Verification if security policies are enforced by user permissions. Not feasible manually:

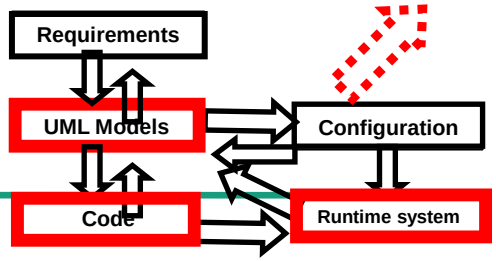
- Large amount of data (e.g. 60.000 permissions)
- Complex relations between permissions (e.g. delegation)

Idea: Automated analysis of business process models against user permissions, as well as user permissions against security policy models. [ICSE '08]

Current project (Fraunhofer Attract): Architecture for auditable business process execution (Apex).

[FASE '08]

Run-time Security Verification



General problem: Are verified implementations still secure ***in the system context*** ?

Does the static system model consider all relevant aspects ?

Are the assumptions about the system environment correct ?

Are the necessary abstractions for a static verification valid ?

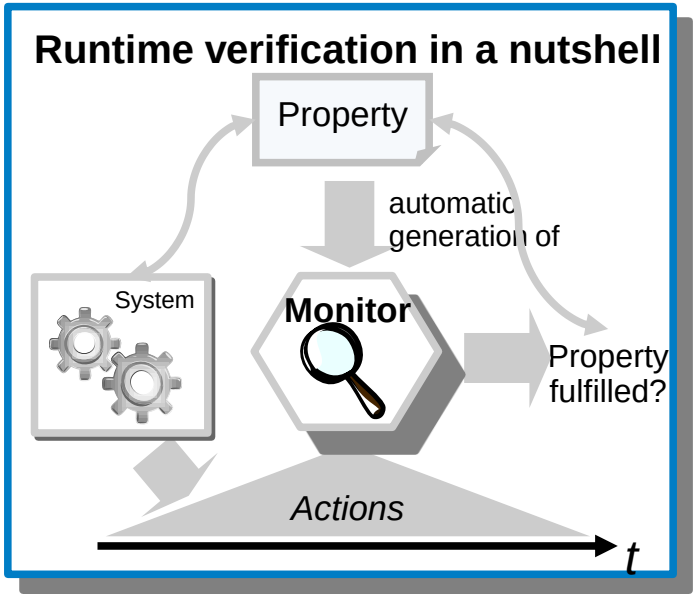
➔ Solution: Run-time verification.

Classic approach: Fred Schneider's Security Automata (only safety properties).

New approach with 3-valued semantic for LTL: also non-safety properties.

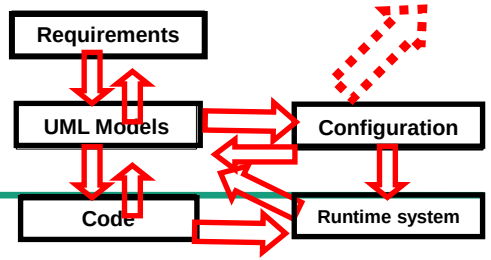
Validation with different versions Java Secure Sockets Extension.

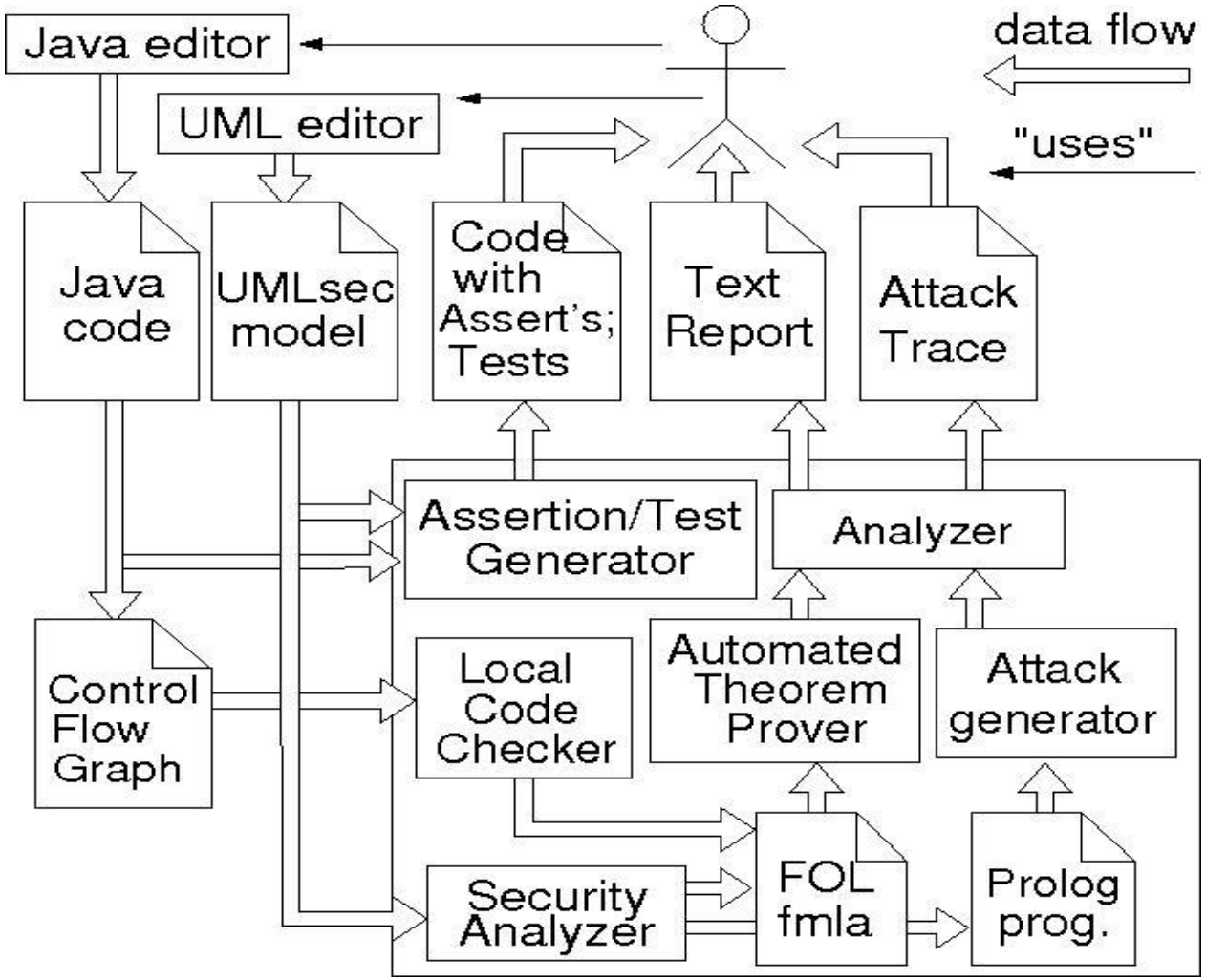
[Diss. A. Bauer]



[Jour. Computers & Security '10, Computer Journal '10]

Tool support





[UML 04, FASE 05, Jour. Softw. Tools & Techn. Transf. (STTT) 07]

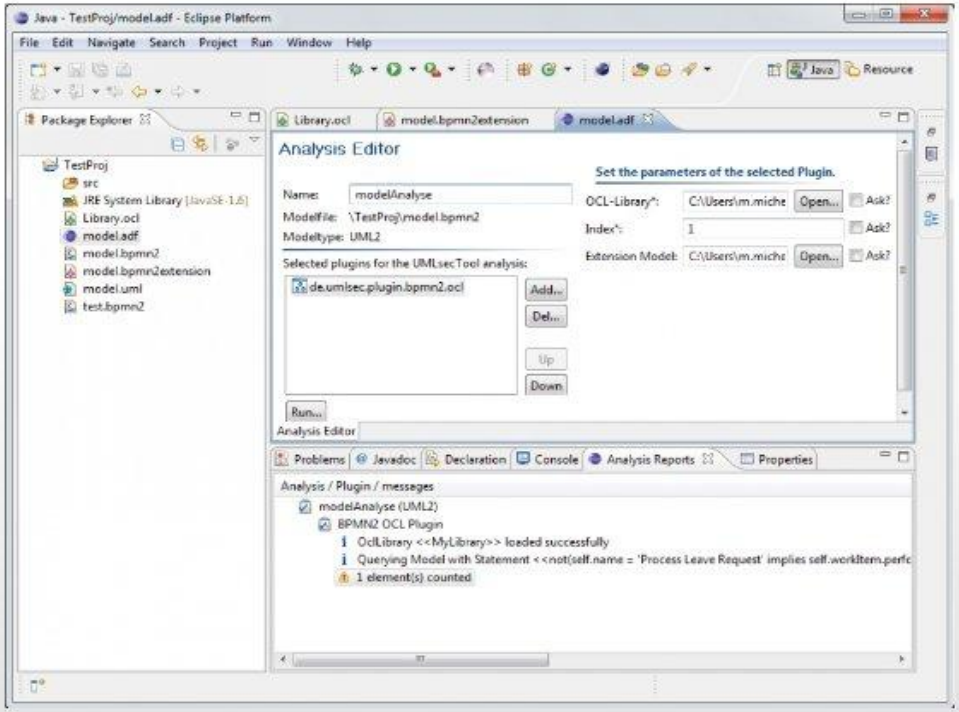
Tool Support (s. http://carisma.umlsec.de)

Welcome to CARiSMA!

Modeling offers an unprecedented opportunity for high-quality critical systems development that is feasible in an industrial context. CARiSMA enables you to perform:

- **compliance** analyses,
- **risk** analyses, and
- **security** analyses

of software models.[1)]



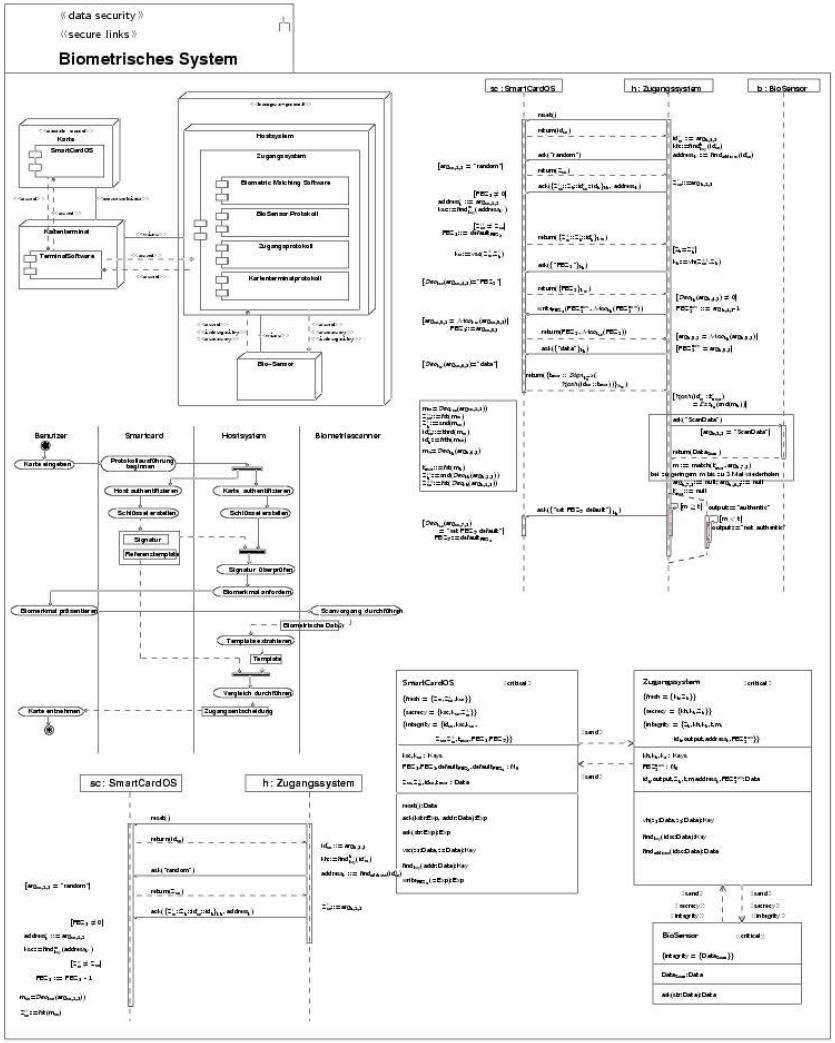
Since CARiSMA is a reimplemented variant of the former UMLsec tool it natively supports UML models.
Due to its EMF-based implementation CARiSMA can also support **domain-specific modeling languages** such as BPMN.

CARiSMA is fully **integrated into Eclipse** and can thus become part of the modeling tool of your choice including but not limited to TOPCASED, Papyrus MDT, IBM Rational Software Architect, and many others.

A flexible **plugin architecture** makes CARiSMA extensible for new languages and allows users to implement their own compliance, risk, or security checks.

Example: Biometric authentication system in industrial development.

Secure ?



Challenges

- Adapt UML to critical system application domains.
- Correct use of UML in the application domains.
- Conflict between flexibility and unambiguity in the meaning of a notation.
- Improving tool-support for critical systems development with UML.

Some Open Problems

Secure systems out of (in)secure mechanisms.

Security as pervasive property: vs. dependability, program analysis, formal methods, software engineering, programming languages, compilers, computer architectures, operating systems, reactive systems, …, …

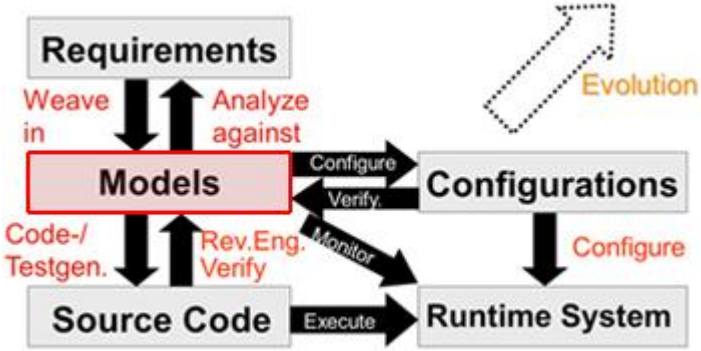
Problem: no integration / coherence.

How to put all this stuff together in a water-tight way within security engineering approach ?

Necessary for security (attacks on boundaries between views / aspects / levels …).

Roadmap

1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

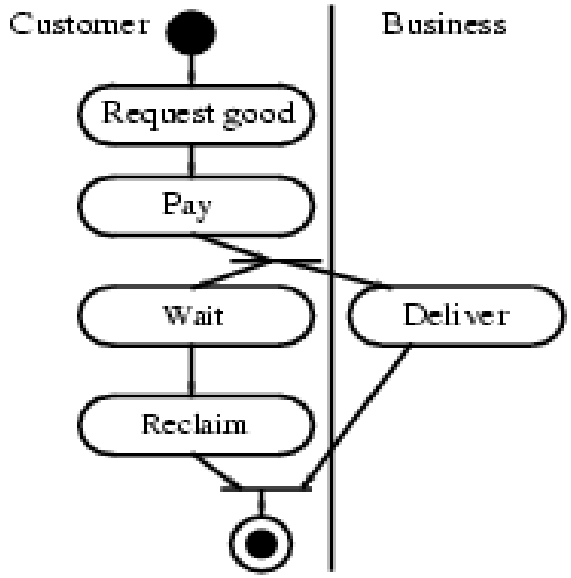


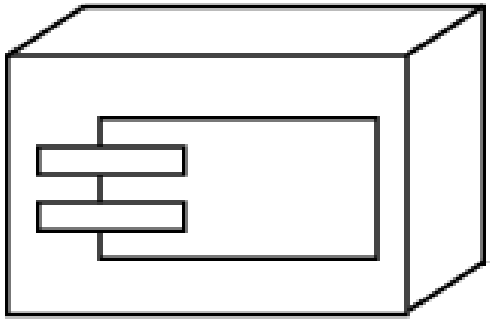
UML

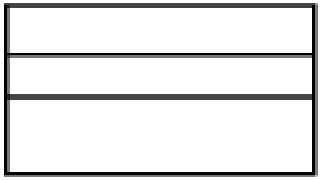
Unified Modeling Language (UML):

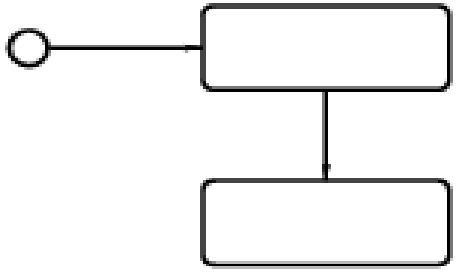
- visual modelling for OO systems
- different views on a system
- high degree of abstraction possible
- de-facto industry standard (OMG)
- standard extension mechanisms

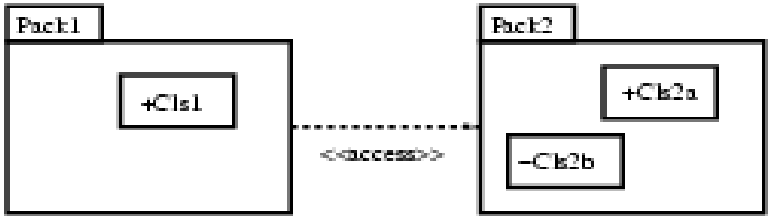
A Glimpse at UML











Used Fragment of UML

Use case diagram: discuss requirements of the system

Class diagram: data structure of the system

Statechart diagram: dynamic component behaviour

Activity diagram: flow of control between components

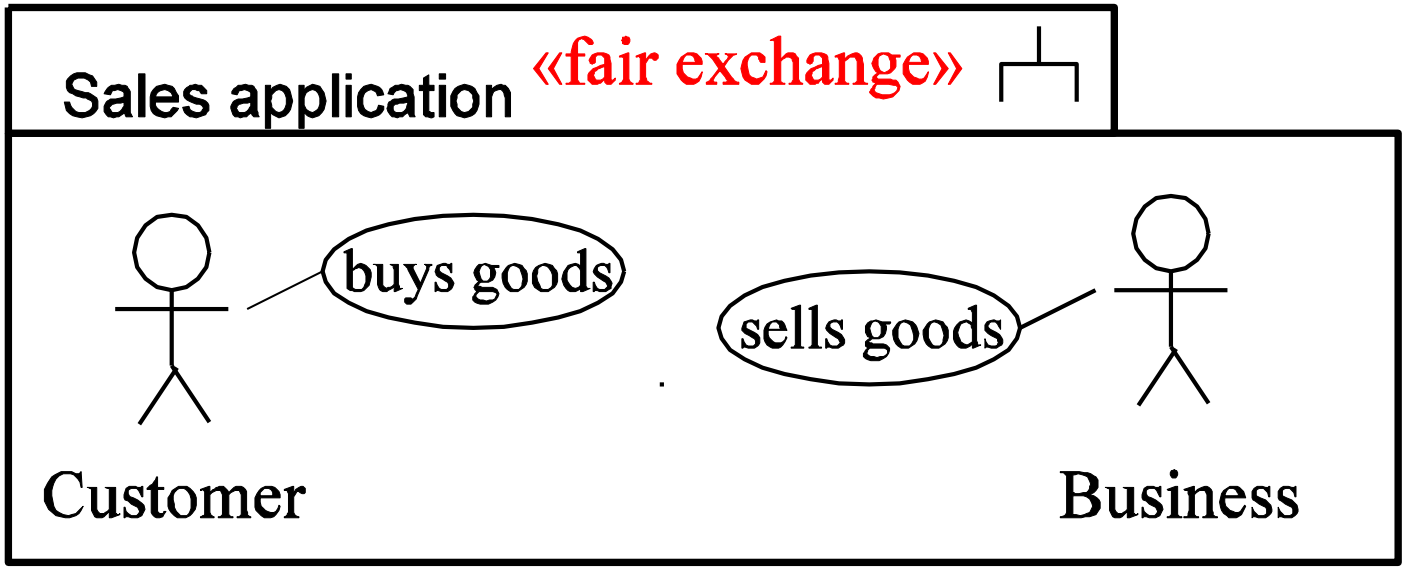
Sequence diagram: interaction by message exchange

Deployment diagram: physical environment

Package/Subsystem: collect diagrams for system part

Current: UML 2.0 (released 2004). (In this lecture use only fragment that was conservatively included from UML1.x.)

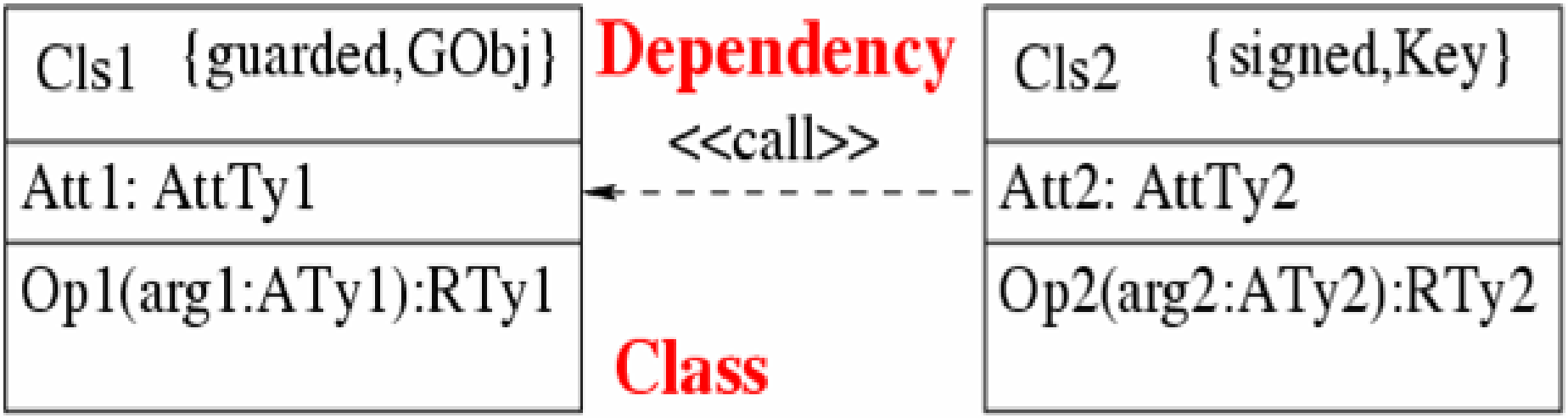
UML Run-through: Use Case Diagrams



Specify intended use cases of the system: scenarios on the functionality offered to the user or other systems.

Actors, linked to activities.

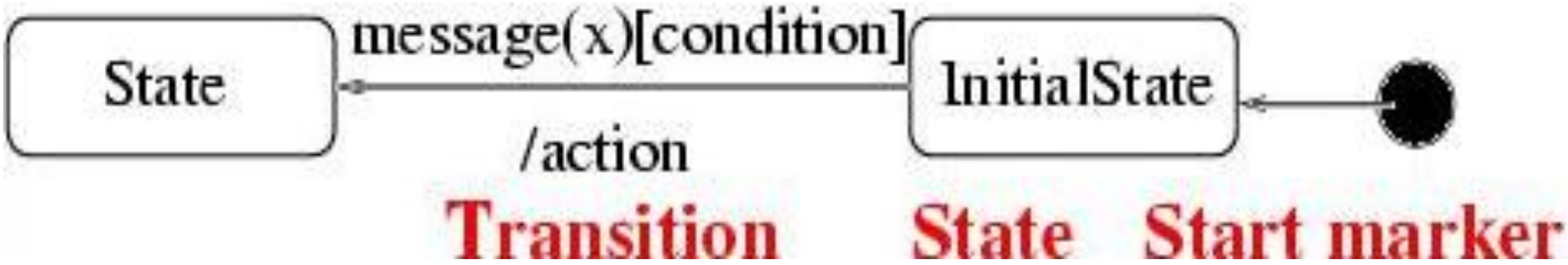
UML Run-through: Class diagram



Class structure of system.

Classes with attributes and operations/signals; relationships between classes.

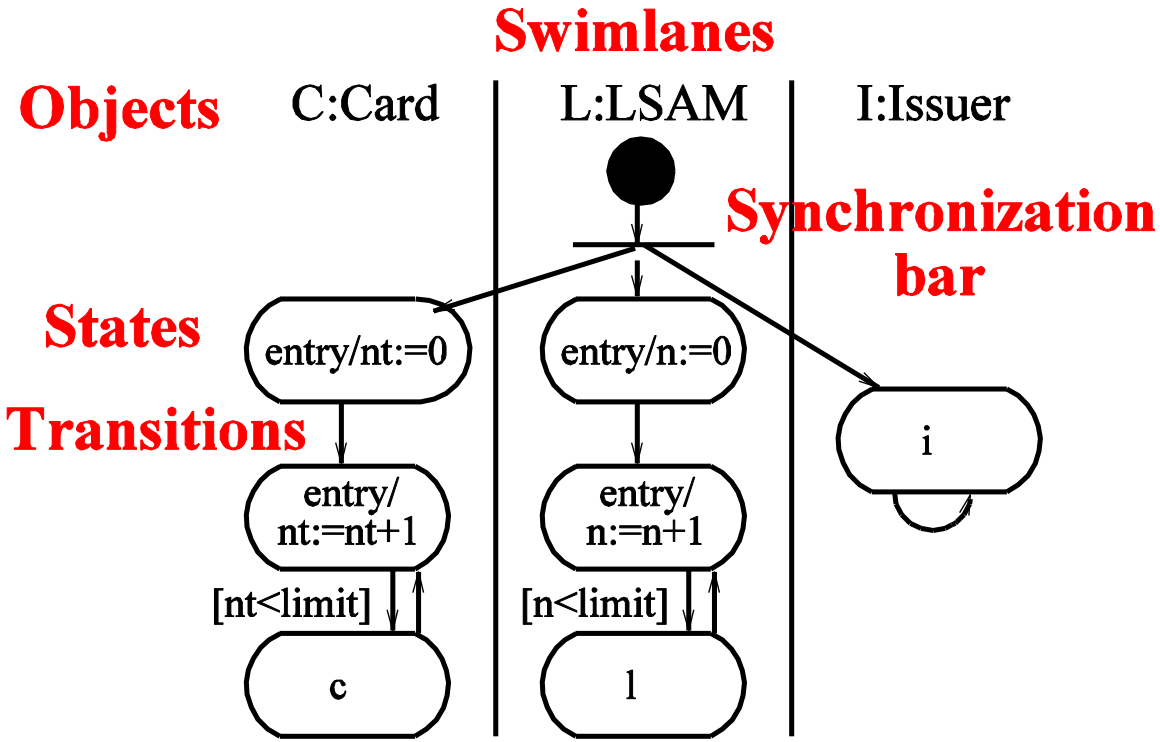
UML Run-through: Statecharts



Dynamic behaviour of individual component.

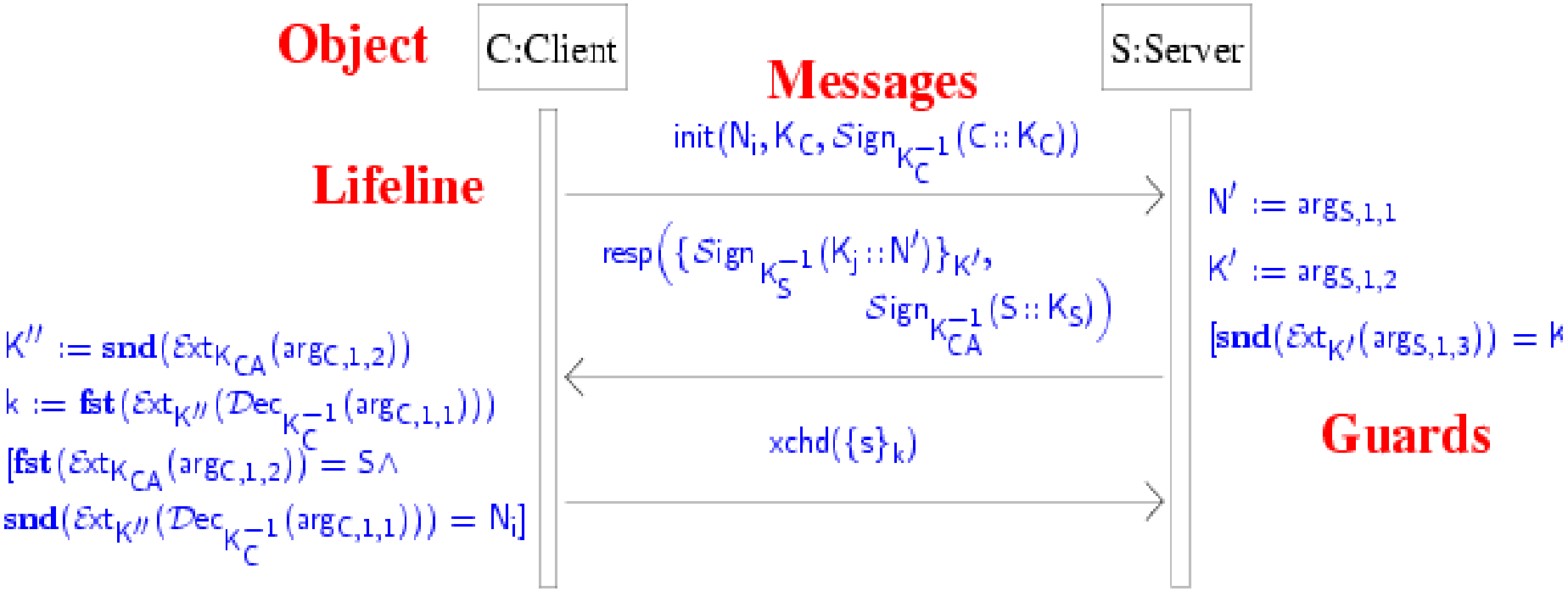
Input events cause state change and output actions.

UML Run–through: Activity Diagram



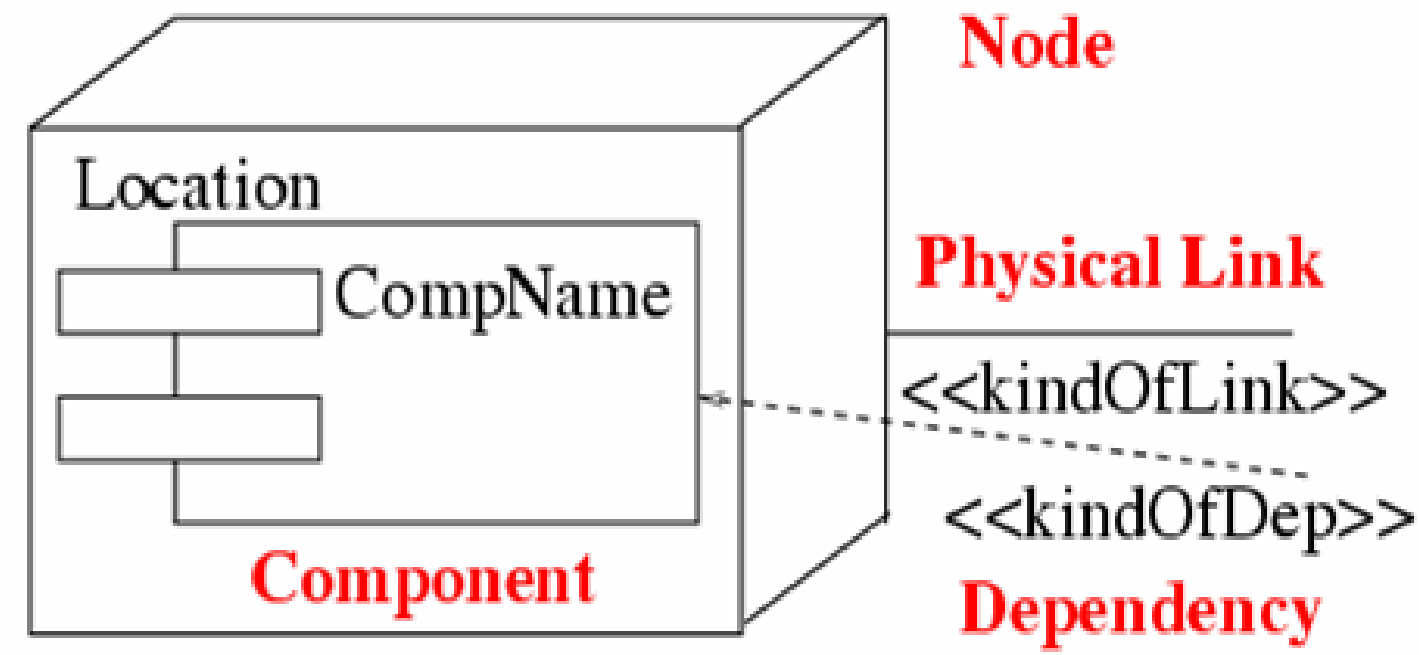
Specify the control flow between components within the system, at higher degree of abstraction than statecharts and sequence diagrams.

UML Run-through: Sequence Diagram



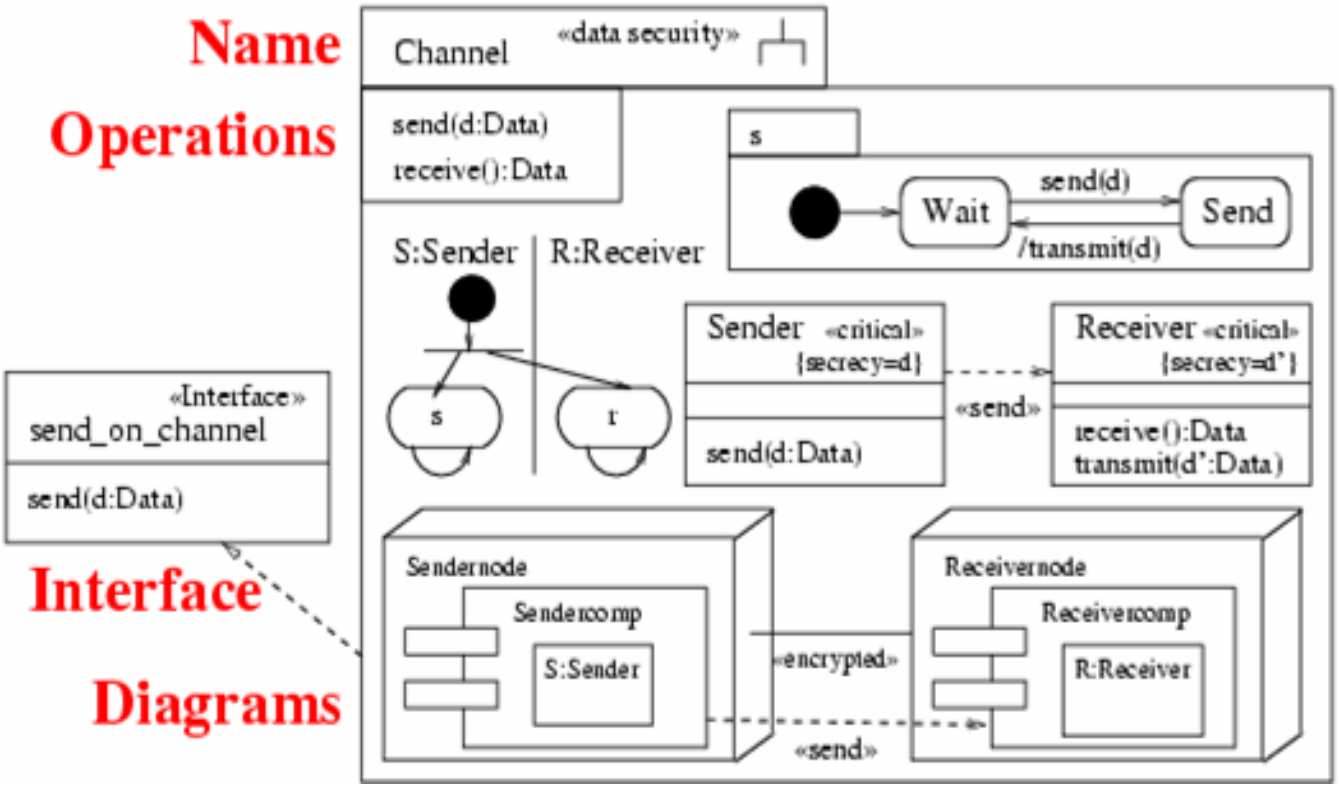
Describe interaction between objects or components via message exchange.

UML Run-through: Deployment Diagram



Describe the physical layer on which the system is to be implemented.

UML Run-through: Package



May be used to organize model elements into groups.

UML Extension Mechanisms

Stereotype: specialize model element using <<label>>.

Tagged value: attach {tag=value} pair to stereotyped element.

Constraint: refine semantics of stereotyped element.

Profile: gather above information.

Requirements on UML Extension for Security

Provide basic security requirements such as secrecy, integrity, authenticity.

Allow considering different threat scenarios depending on adversary strengths.

Allow including important security concepts (e.g. *tamper-resistant hardware*).

Allow incorporating security mechanisms (e.g. access control).

Provide security primitives (e.g. (a)symmetric encryption).

Allow considering underlying physical security.

Allow addressing security management (e.g. secure workflow).

Also: Include domain-specific security knowledge (Java, smart cards, CORBA, ...).

UMLsec: General Ideas

Activity diagram: secure control flow, coordination

Class diagram: exchange of data
preserves security levels

Sequence diagram: security-critical interaction

Statechart diagram: security preserved
within object

Deployment diagram: physical security requirements

Package: holistic view on security

UMLsec Profile (excerpt)

Stereotype	Base class	Tags	Constraints	Description
Internet	link			Internet connection
secure links	subsystem		dependency security matched by links	enforces secure communication links
secrecy	dependency			assumes secrecy
secure dependency	subsystem		call, send respect data security	structural interaction data security
no down-flow	subsystem	high	prevents down-flow	information flow
data security	subsystem		provides secrecy, integrity	basic datasec requirements
fair exchange	package	start, stop	after start eventually reach stop	enforce fair exchange
guarded access	Subsystem		guarded objects acc. through guards.	access control using guard objects

UMLsec as Integrating Formal Framework

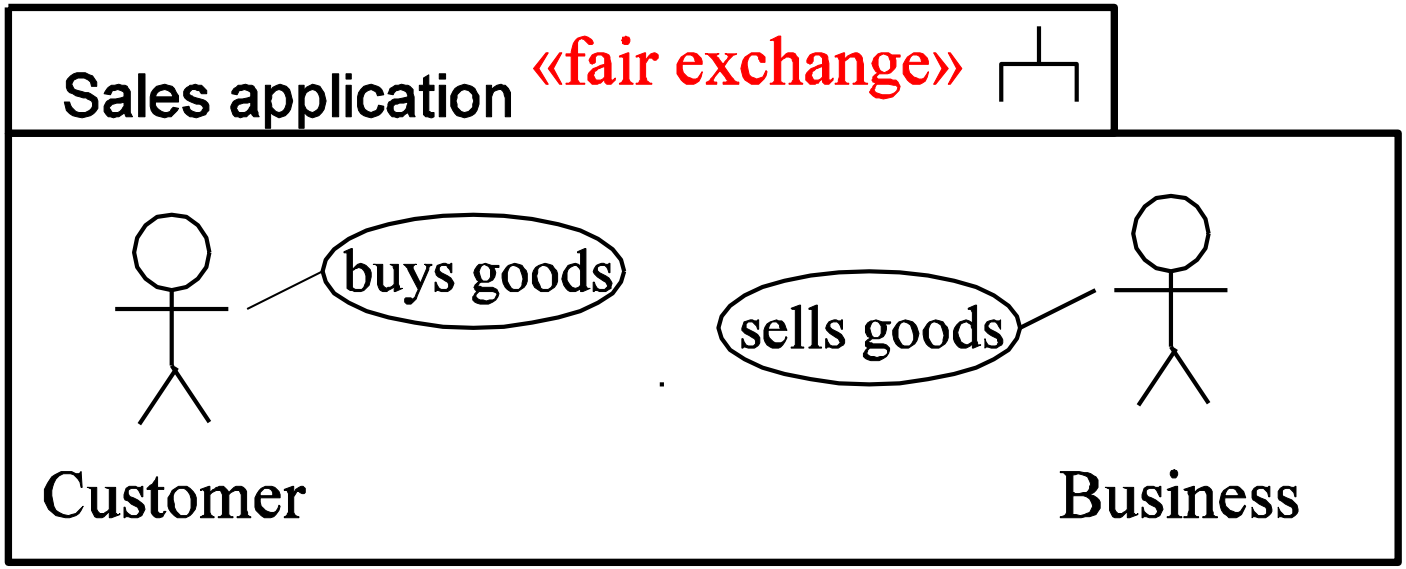
Have formalizations of major security requirements in one integrated notation.

Want to relate / combine requirements; get modularity / composability, hierarchical decomposition, refinement, … :

For example:

- If system satisfies <<secure links>> and subsystems satisfy <<data security>> then system satisfies <<data security>>.

Requirements with Use Case Diagrams



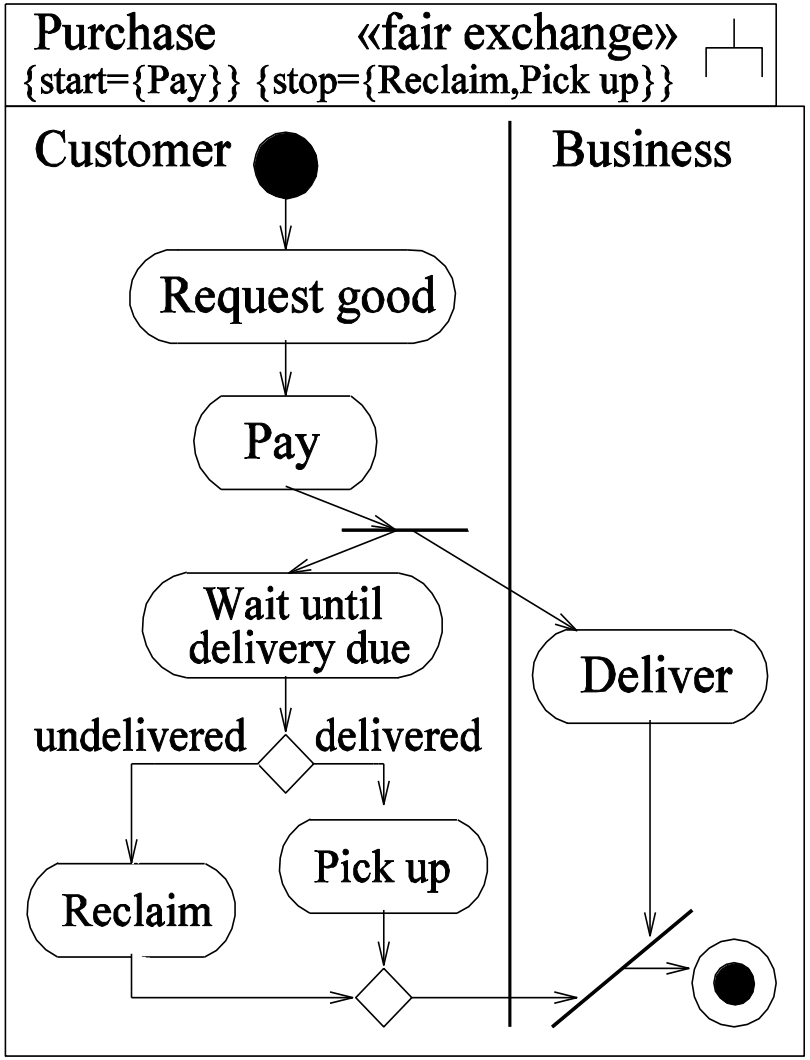
Capture security requirements
in use case diagrams.

Constraint: need to appear in corresponding activity diagram.

Fair Exchange

Customer buys good from a business.

How can enforce fair exchange:
After payment, customer is eventually either delivered good or able to reclaim payment (or vc.vs.).



<<fair exchange>>

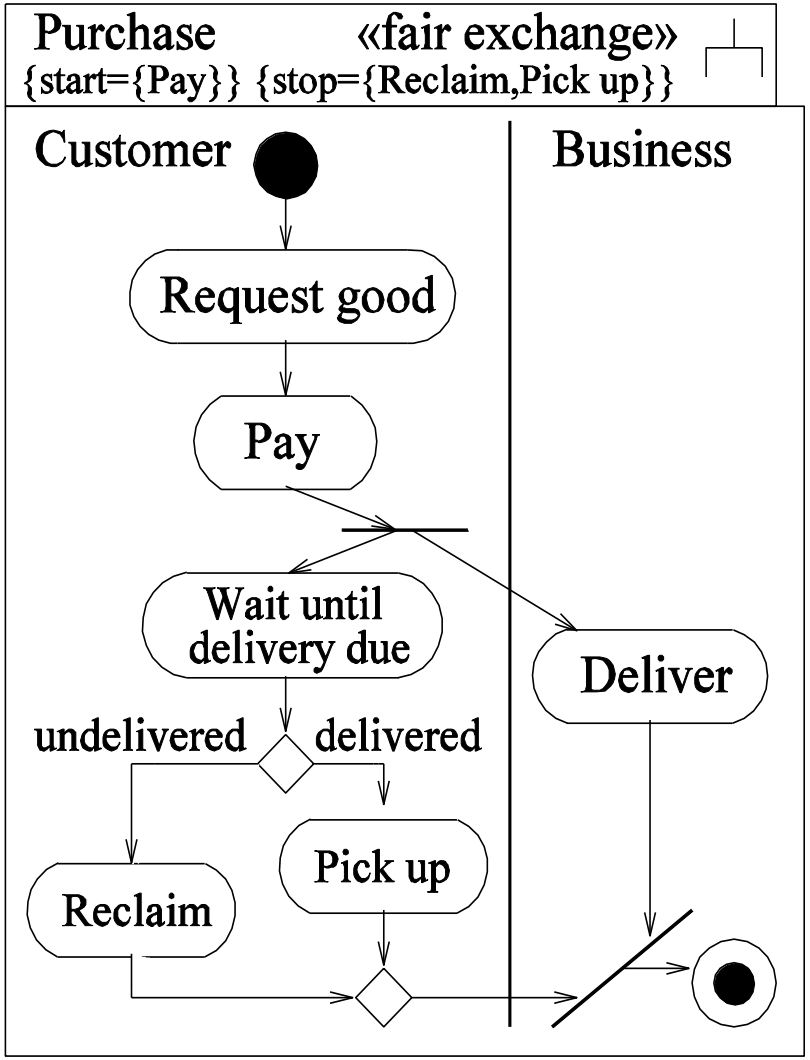
Ensures generic fair exchange condition.

Constraint: after a {start} state in activity diagram is reached, eventually reach {stop} state.

(Cannot be ensured for systems that an attacker can stop completely.)

Example

<<fair exchange>> fulfilled if adversary cannot stop system: After payment, customer is eventually either delivered good or able to reclaim payment.



<<Internet>>, <<encrypted>>, …

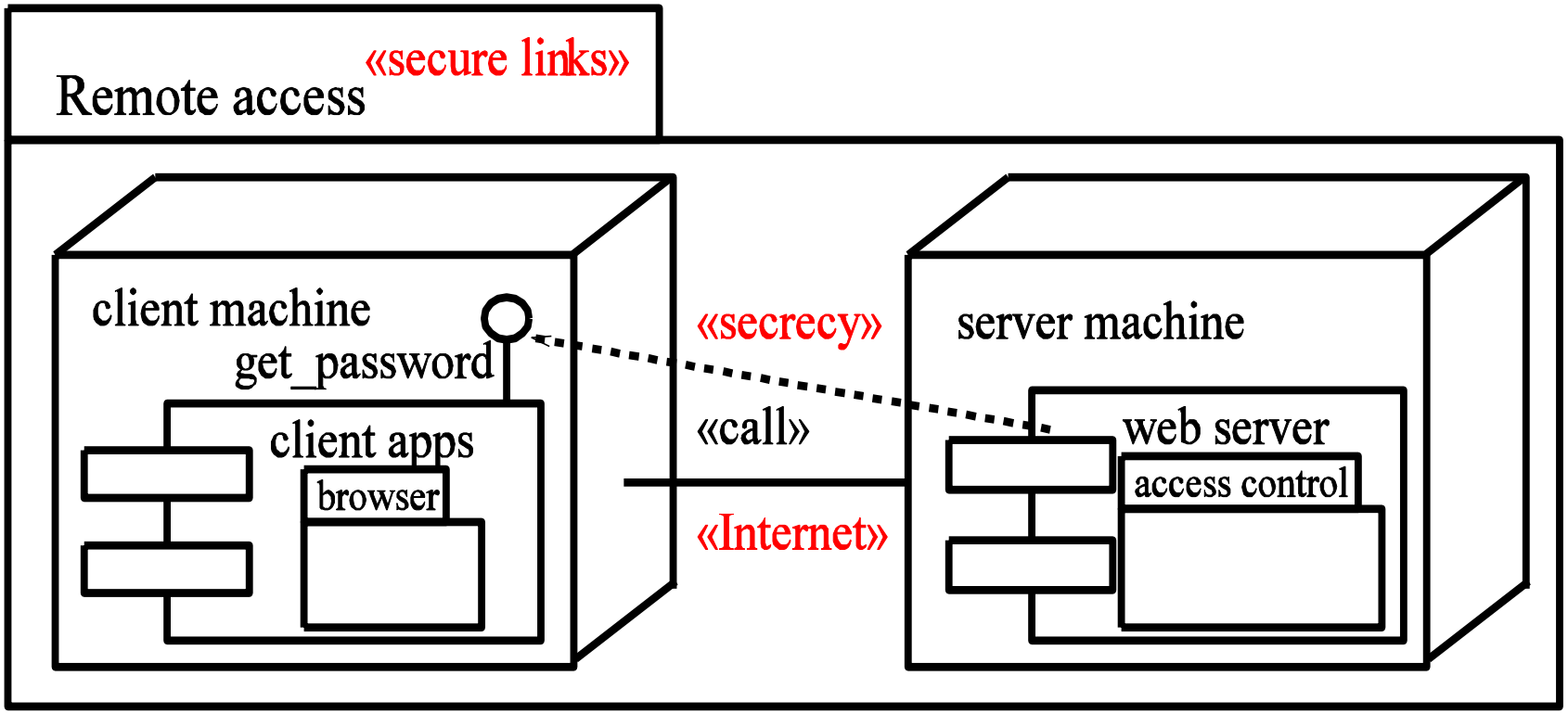
Kinds of communication links resp. system nodes.

For adversary type A, stereotype s, have set Threats (s) ∈ {delete, read, insert, access} of actions that adversaries are capable of.

Default attacker:

Stereotype	Threats ()
Internet	{delete, read, insert}
encrypted	{delete}
LAN	∅
smart card	∅

Secure Architecture



Architecture secure against default adversary ?

<<secure links>>

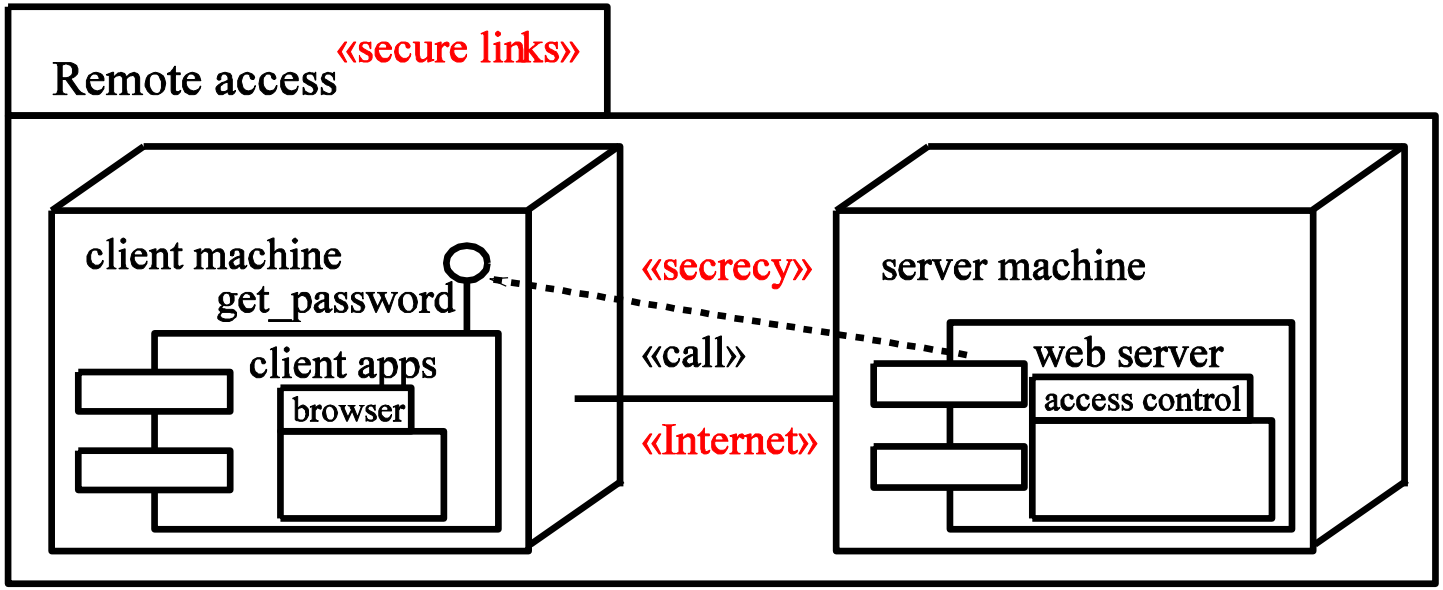
Ensures that physical layer meets security requirements on communication.

Constraint: for each dependency *d* with stereotype *s* ∈ {<<secrecy>>, <<integrity>>} between components on nodes *n*≠*m*, have a communication link *l* between *n* and *m* with stereotype *t* such that

if *s* = <<secrecy>>: have read ∉ Threats *(t)*.

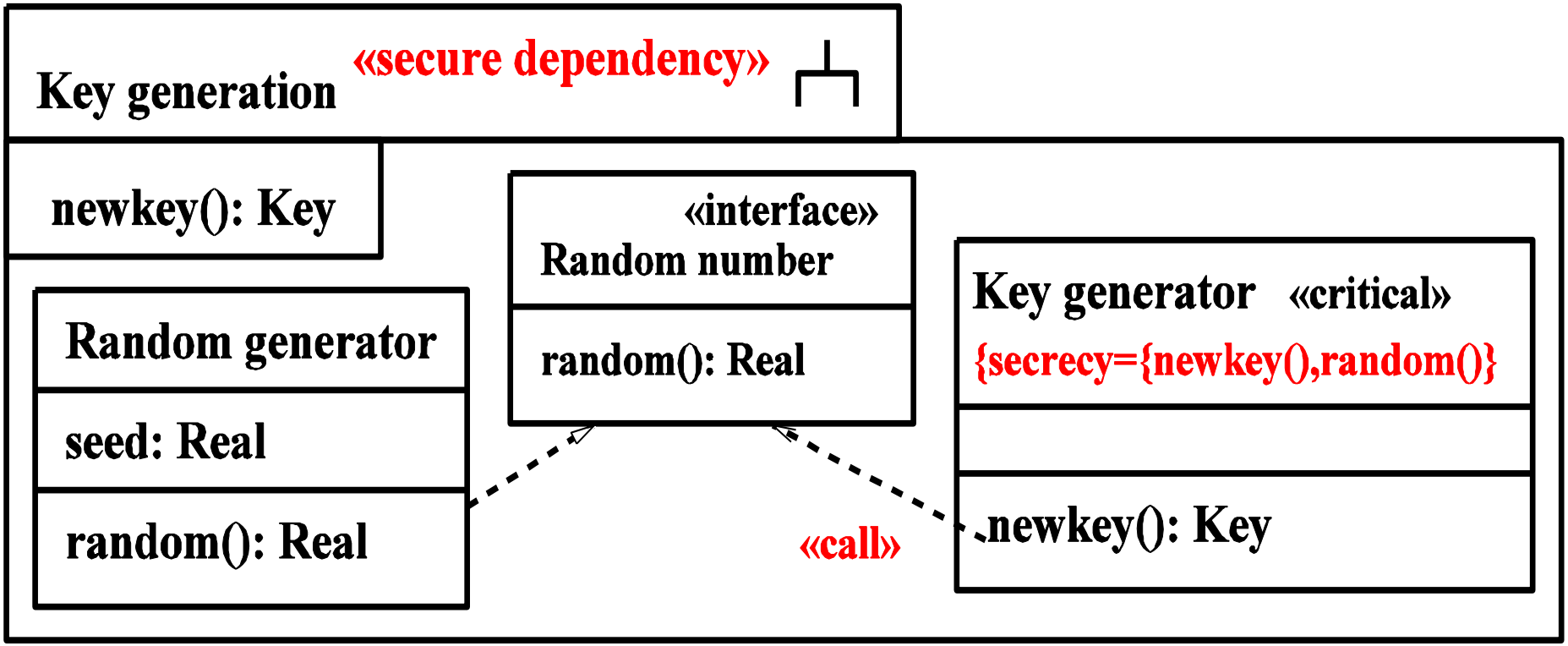
if *s* = <<integrity>>: have insert ∉ Threats *(t)*.

Example <<secure links>>



Given default adversary type, constraint for stereotype <<secure links>> violated: According to the $\text{Threats}_{\text{default}}$(Internet) scenario, <<Internet>> link does not provide secrecy against default adversary.

Secure Data Structure



Data structure secure ?

<<secure dependency>>

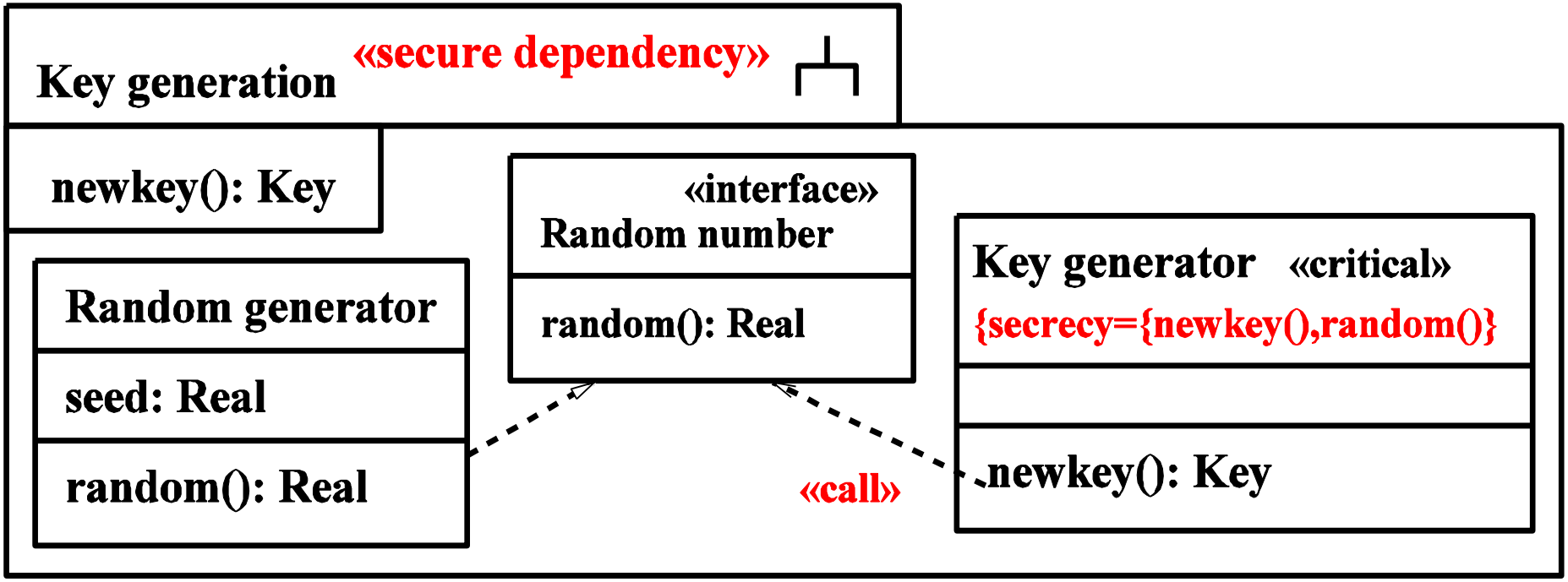
Ensure that <<call>> and <<send>> dependencies between components respect security requirements on communicated data given by tags {secrecy}, {integrity}.

Constraint: for <<call>> or <<send>> dependency from *C* to *D* (and similarly for {integrity}):

Msg in *D* is {secrecy} in *C* if and only if also in *D*.

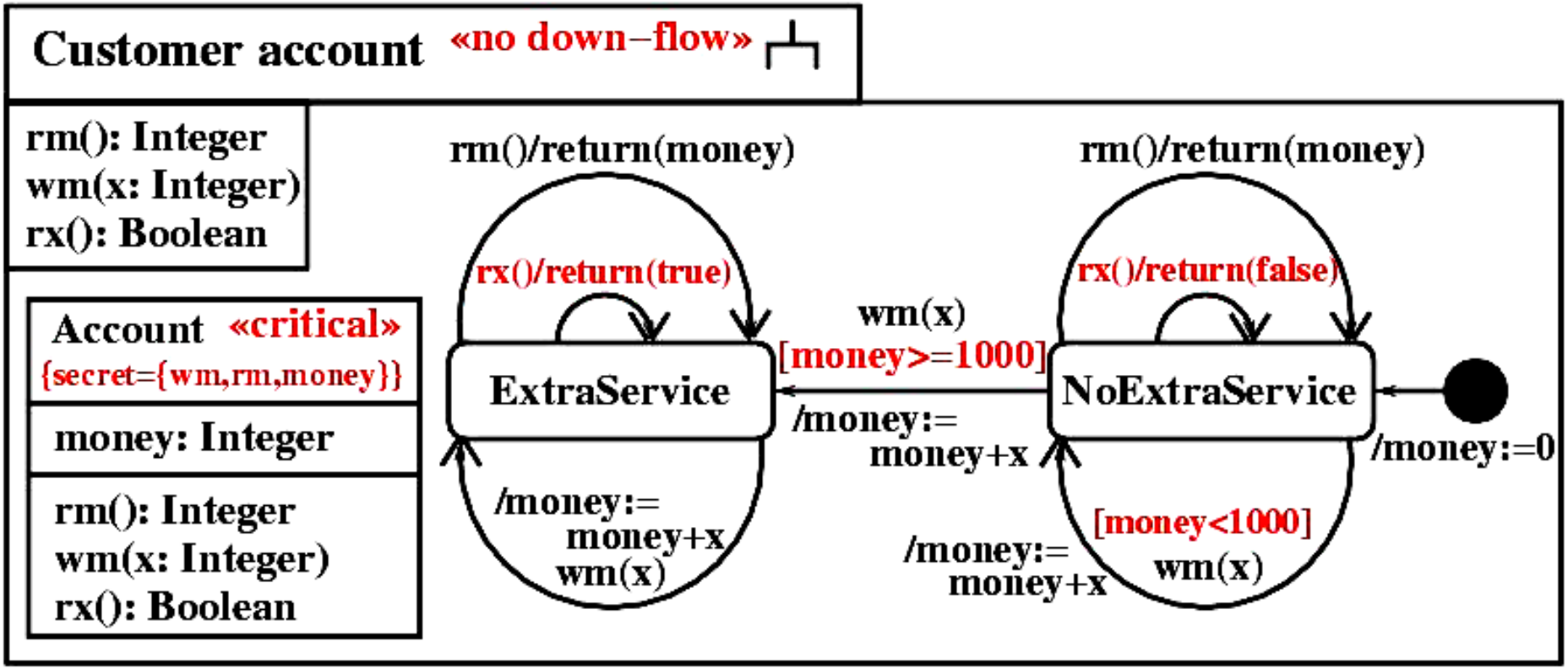
If msg in *D* is {secrecy} in *C*, dependency stereotyped <<secrecy>>.

Example <<secure dependency>>



Violates <<secure dependency>>: Random generator and <<call>> dependency do not give security level for random() to key generator.

Secure Information Flow



No partial leakage of secrets ?

<<no down–flow>>

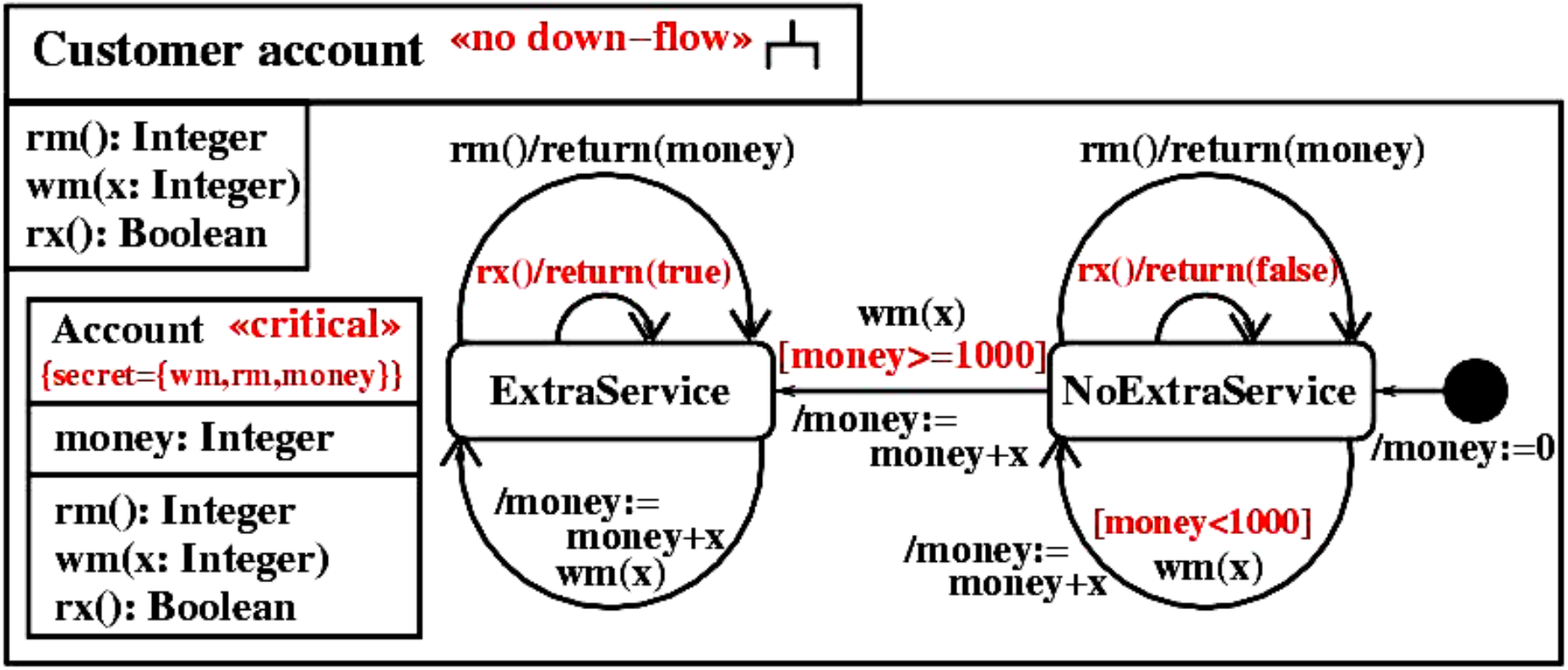
Enforce secure information flow.

Constraint:

Value of any data specified in {secrecy} may influence only the values of data also specified in {secrecy}.

Formalize by referring to formal behavioural semantics.

Example <<no down-flow>>



<<no down–flow>> violated: partial information on input of secret wm() returned by non-secret rx().

Secure Use of Cryptography

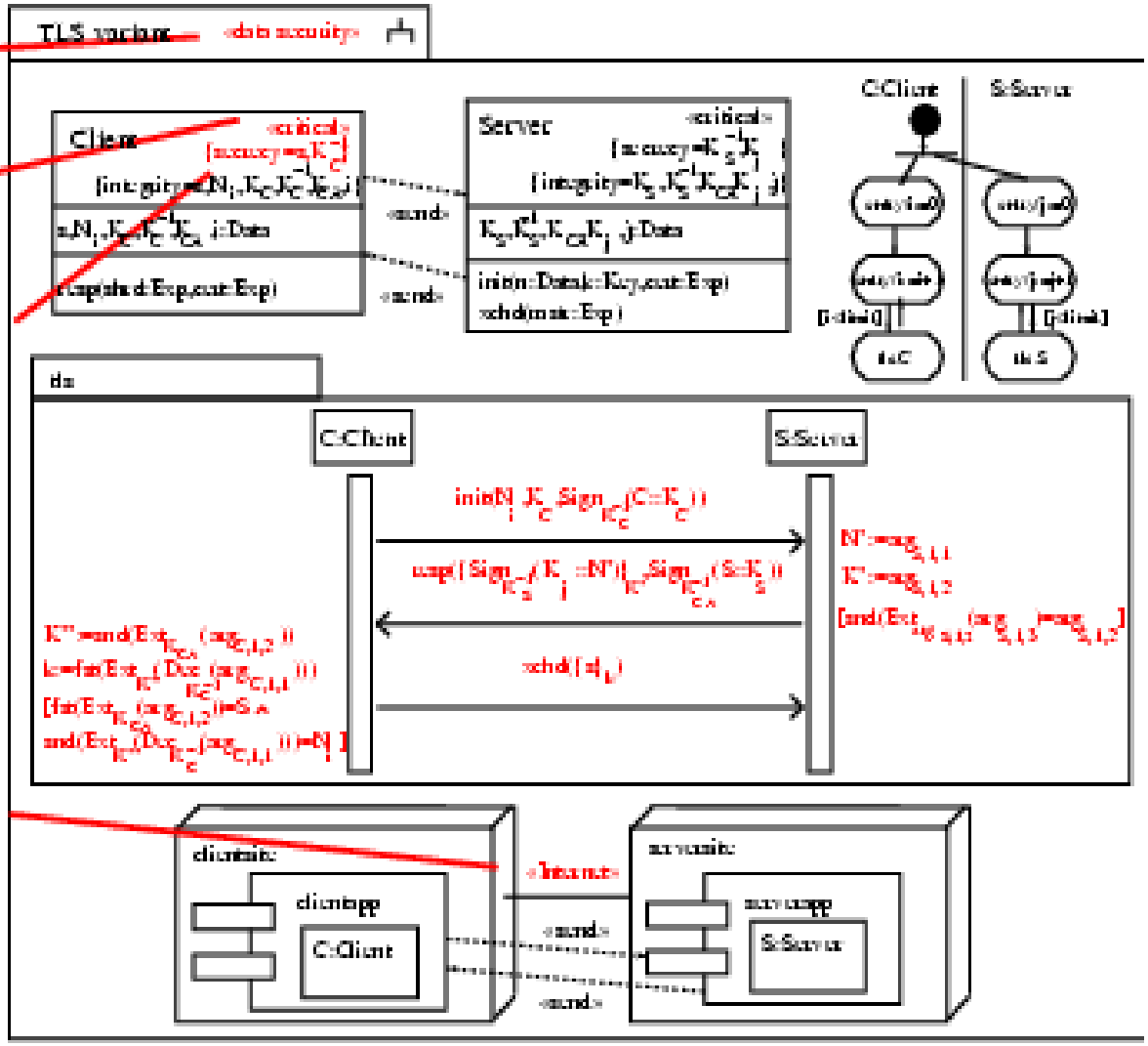
«data security»

«critical»

$\{secrecy = \{s, K_C^{-1}\}\}$

Variant of TLS (INFOCOM`99). Cryptoprotocol secure against default adversary ?

«Internet»



<<data security>>

Security requirements of data marked <<critical>> enforced against threat scenario from deployment diagram.

Constraints: Data marked {secrecy}, {integrity}, {authenticity}, {fresh} fulfills respective formalized security requirements.

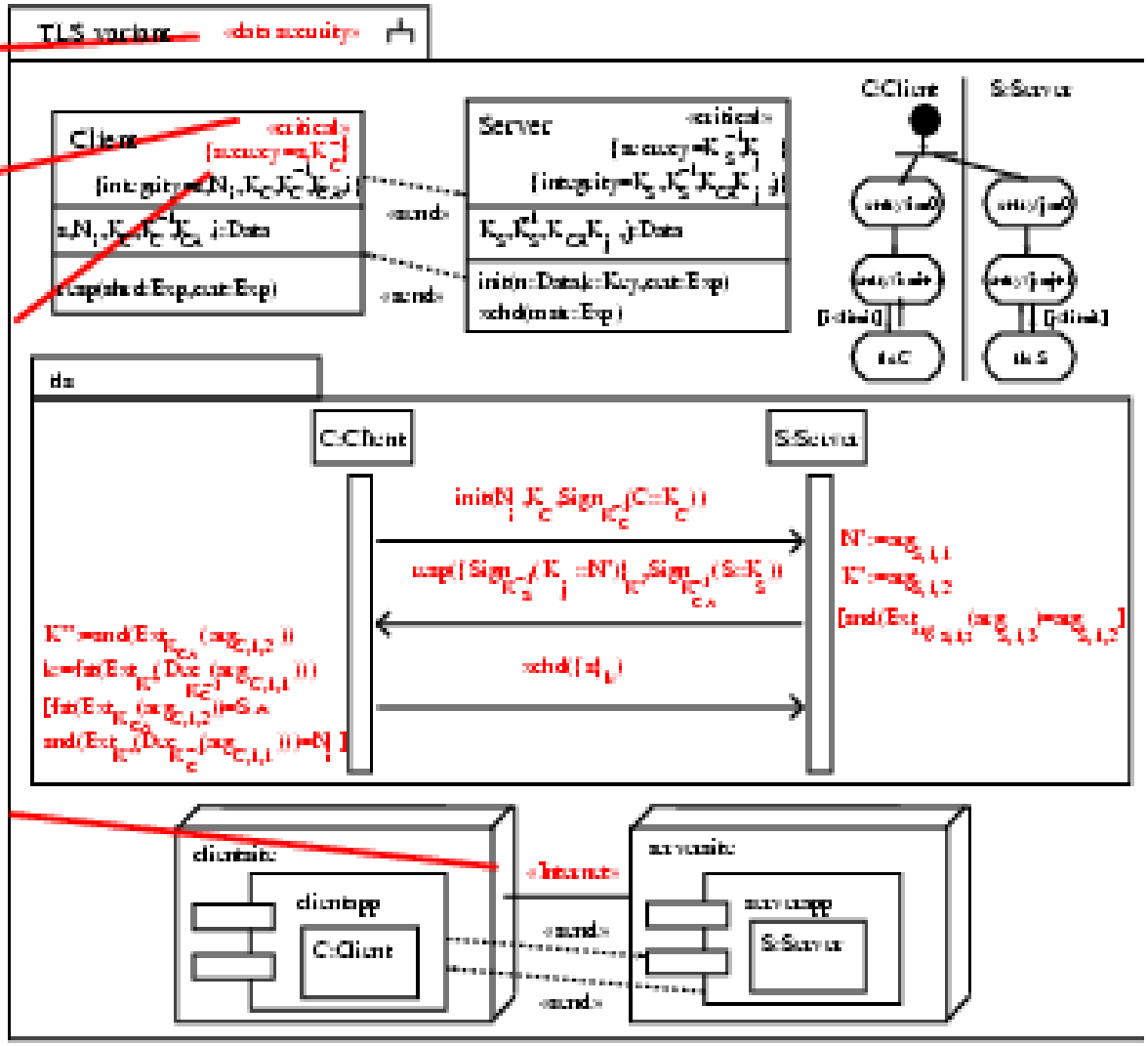
Example <<data security>>

«data security»

«critical»

$\{secrecy = \{s, K_C^{-1}\}\}$

«Internet»



Variant of TLS (INFOCOM`99). Violates {secrecy} of s against default adversary.

Does UMLsec Meet Requirements ?

Security requirements: <<secrecy>>,…

Threat scenarios: Use $\text{Threats}_{\text{adv}}$(ster).

Security concepts: For example <<smart card>>.

Security mechanisms: E.g. <<guarded access>>.

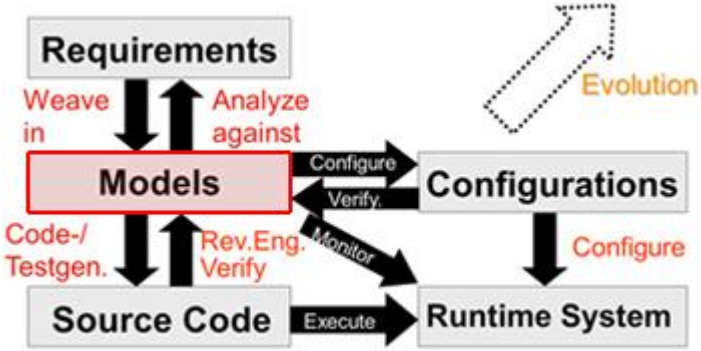
Security primitives: Encryption built in.

Physical security: Given in deployment diagrams.

Security management: Use activity diagrams.

Technology specific: Java, CORBA security.

Roadmap



1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

Java Security Architecture

Originally (JDK 1.0): sandbox.

Too simplistic and restrictive.

JDK 1.2/1.3: more fine-grained security control, signing, sealing, guarding objects, . . .)

BUT: complex, thus use is error-prone.

Java Security policies

Permission entries consist of:

- protection domains (i. e. URL's and keys)
- target resource (e.g. files on local machine)
- corresponding permissions (e.g. read, write, execute)

Signed and Sealed Objects

Need to protect integrity of objects used as authentication tokens or transported across JVMs.

A SignedObject contains an object and its signature.

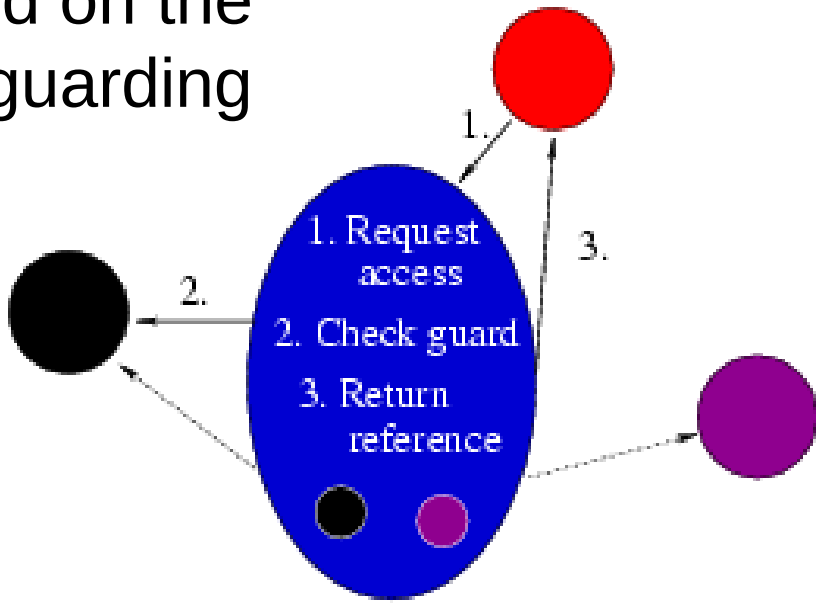
Similarly, need confidentiality.

A SealedObject is an encrypted object.

Guarded Objects

java.security.GuardedObject protects access to other objects.

- access controlled by getObject method
- invokes checkGuard method on the java.security.Guard that is guarding access
- If allowed: return reference. Otherwise: SecurityException



Problem: Complexity

- Permission depends on execution context.
- In particular: multiple threads.
- Thread may cross protection domains.
- doPrivileged() overrides execution context.
- Authentication in presence of adversaries.
- Indirect granting of access with capabilities.

→ Which run-time objects get permission ?

⇒ Use UMLsec to find out at design time.

Design Process

(1) Formulate access control requirements for sensitive objects.

(2) Give guard objects with appropriate access control checks.

(3) Check that guard objects protect objects sufficiently.

(4) Check that access control is consistent with functionality.

(5) Check mobile objects are sufficiently protected.

Reasoning

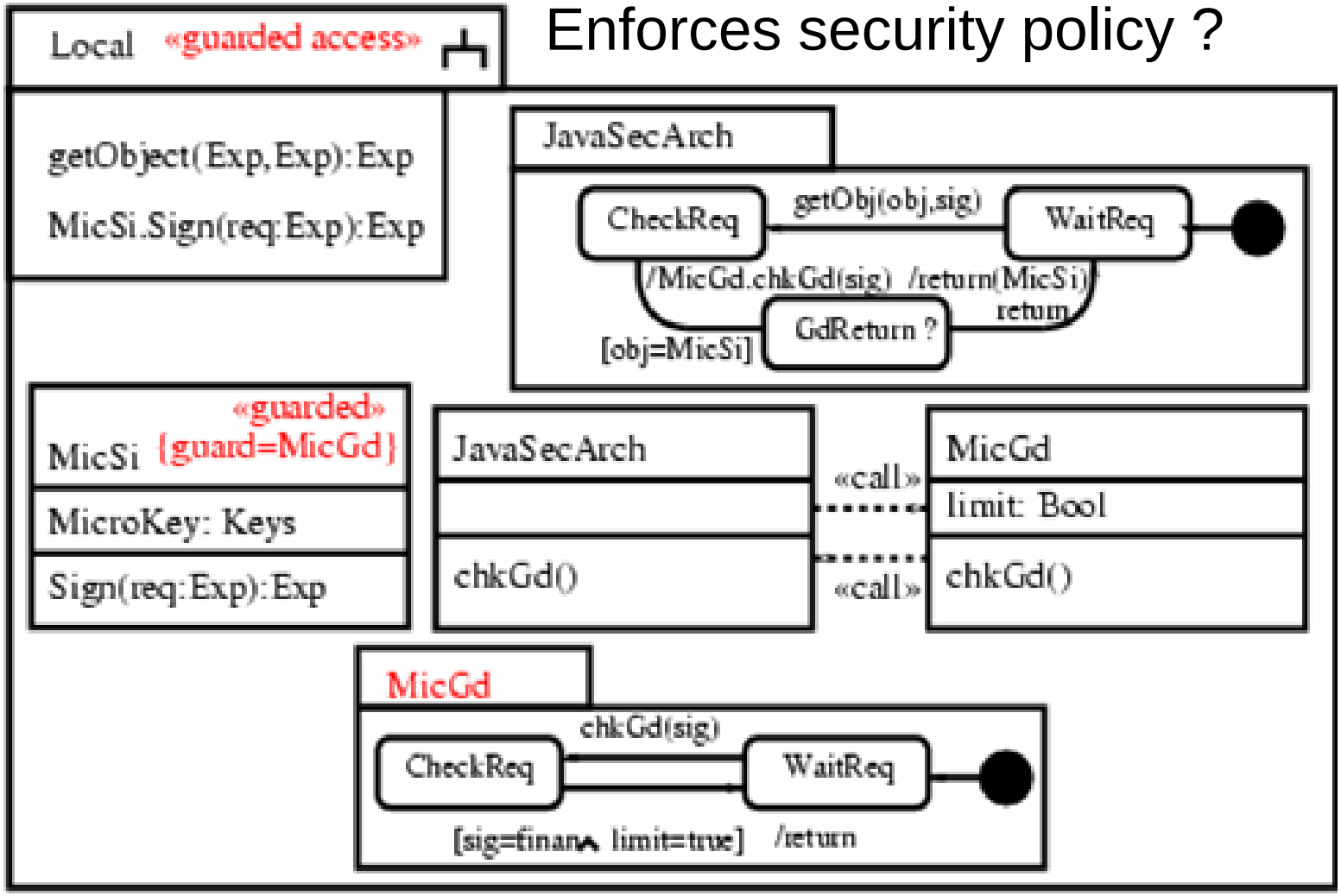
Suppose access to resource according to Guard object specifications granted only to objects signed with *K*.

Suppose all components keep secrecy of *K*.

Then only objects signed with *K* are granted access.

Secure Use of Java Security Arch.

Enforces security policy ?



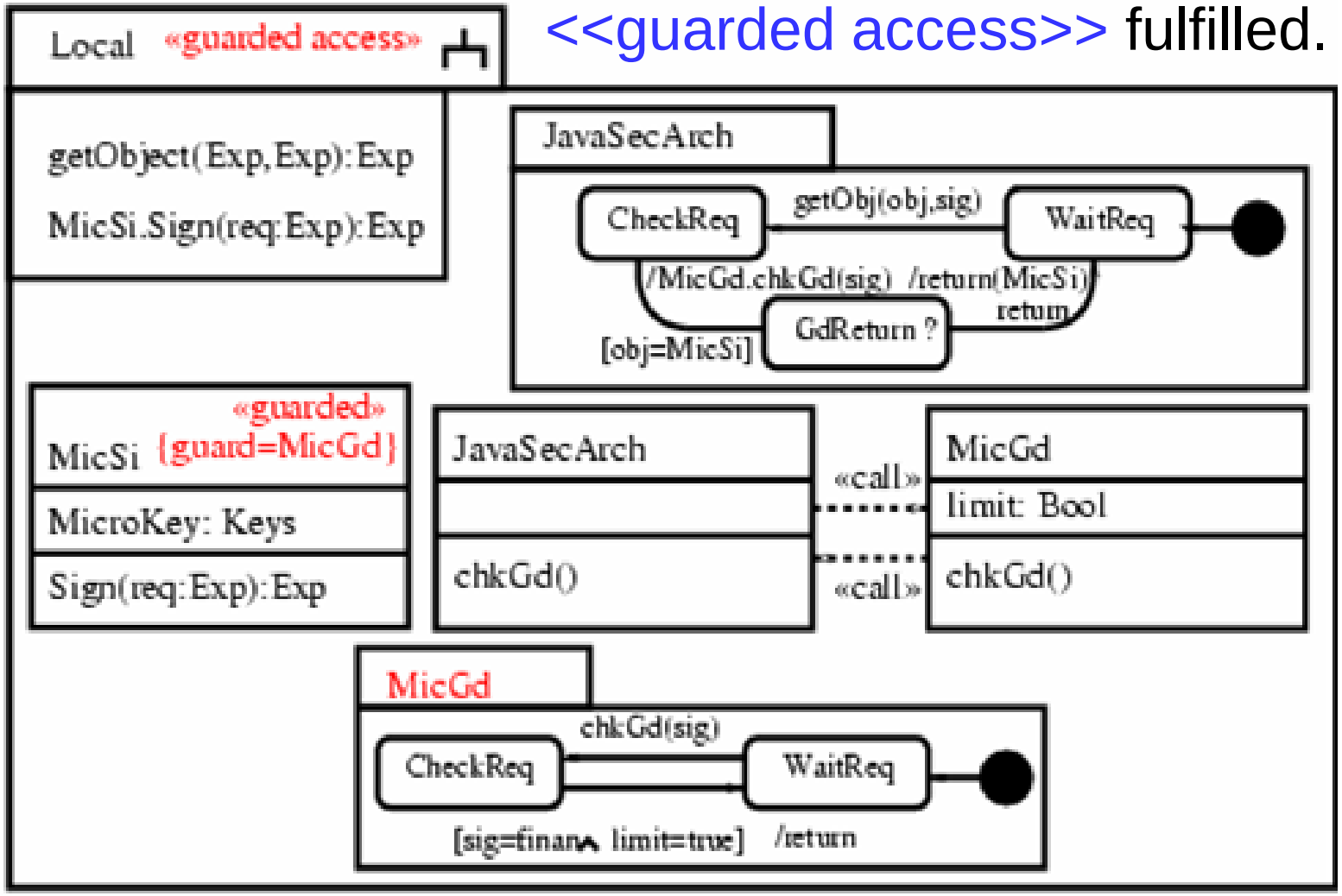
<<guarded access>>

Ensures that in Java, <<guarded>> classes only accessed through {guard} classes.

Constraints:

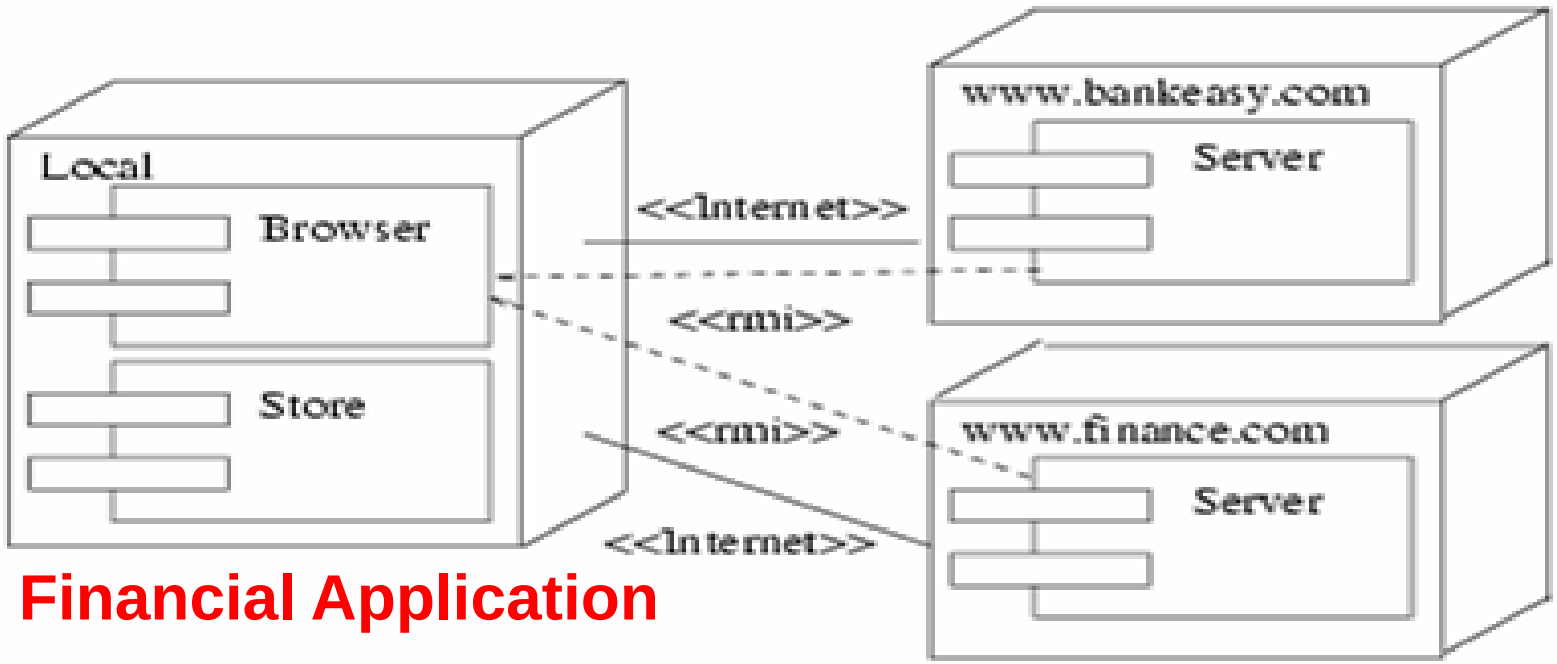
- References of <<guarded>> objects remain secret.
- Each <<guarded>> class has {guard} class enforcing security policy.

Example <<guarded access>>



<<guarded access>> fulfilled.

Example: Financial Application



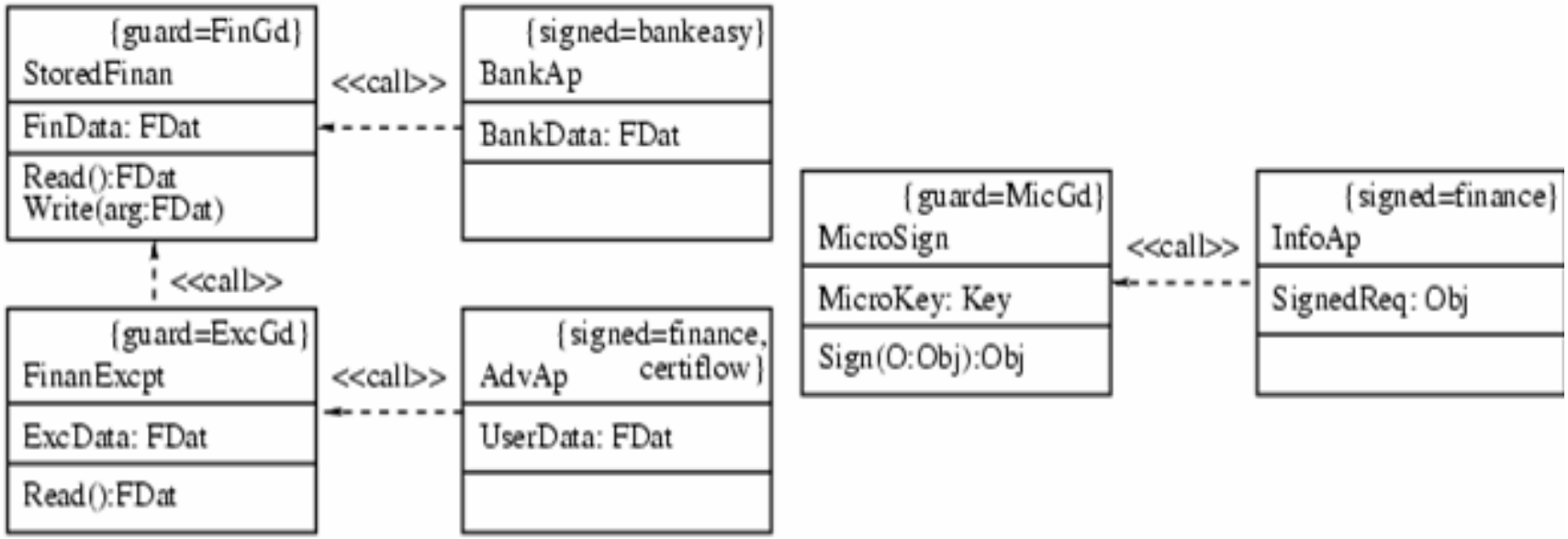
Internet bank, Bankeasy, and financial advisor, Finance, offer services to local user. Applets need certain Privileges (step1).

- Applets from and signed by bank read and write financial data between 1 pm and 2 pm.
- Applets from and signed by Finance use micropayment key five times a week.

Financial Application: Class Diagram

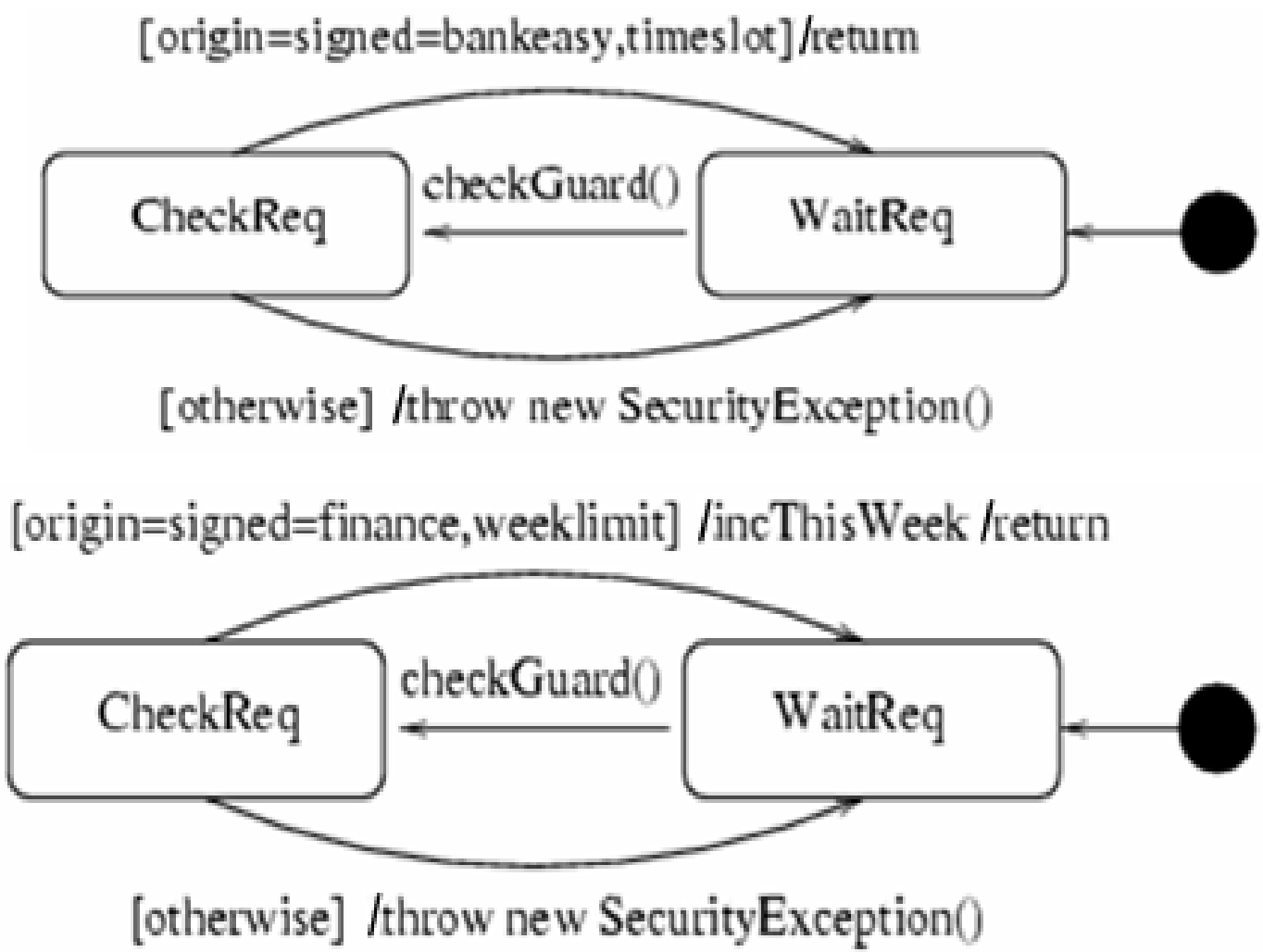
Sign and seal objects sent over Internet for integrity and confidentiality.

GuardedObjects control access.



Financial App: Guard Objects (step 2)

timeslot true between 1pm and 2pm. weeklimit true until access granted five times; incThisWeek increments counter.



Financial Application: Validation

Guard objects give sufficient protection (step 3):

UML specification for guard objects only grants permissions implied by access permission requirements.

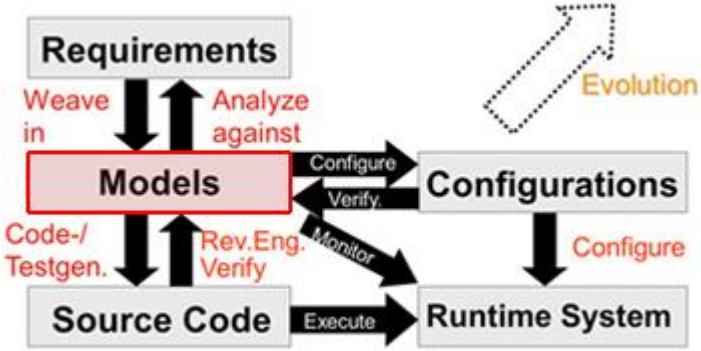
Access control vs. functionality (step 4). E.g.:

If applet should get access to micropayment key according to policy, access granted.

Mobile objects sufficiently protected (step 5) - objects sent over Internet signed and sealed.

Roadmap

1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications



Cryptographic Protocols

Need to be able to securely determine identity of communication partners. Threats:

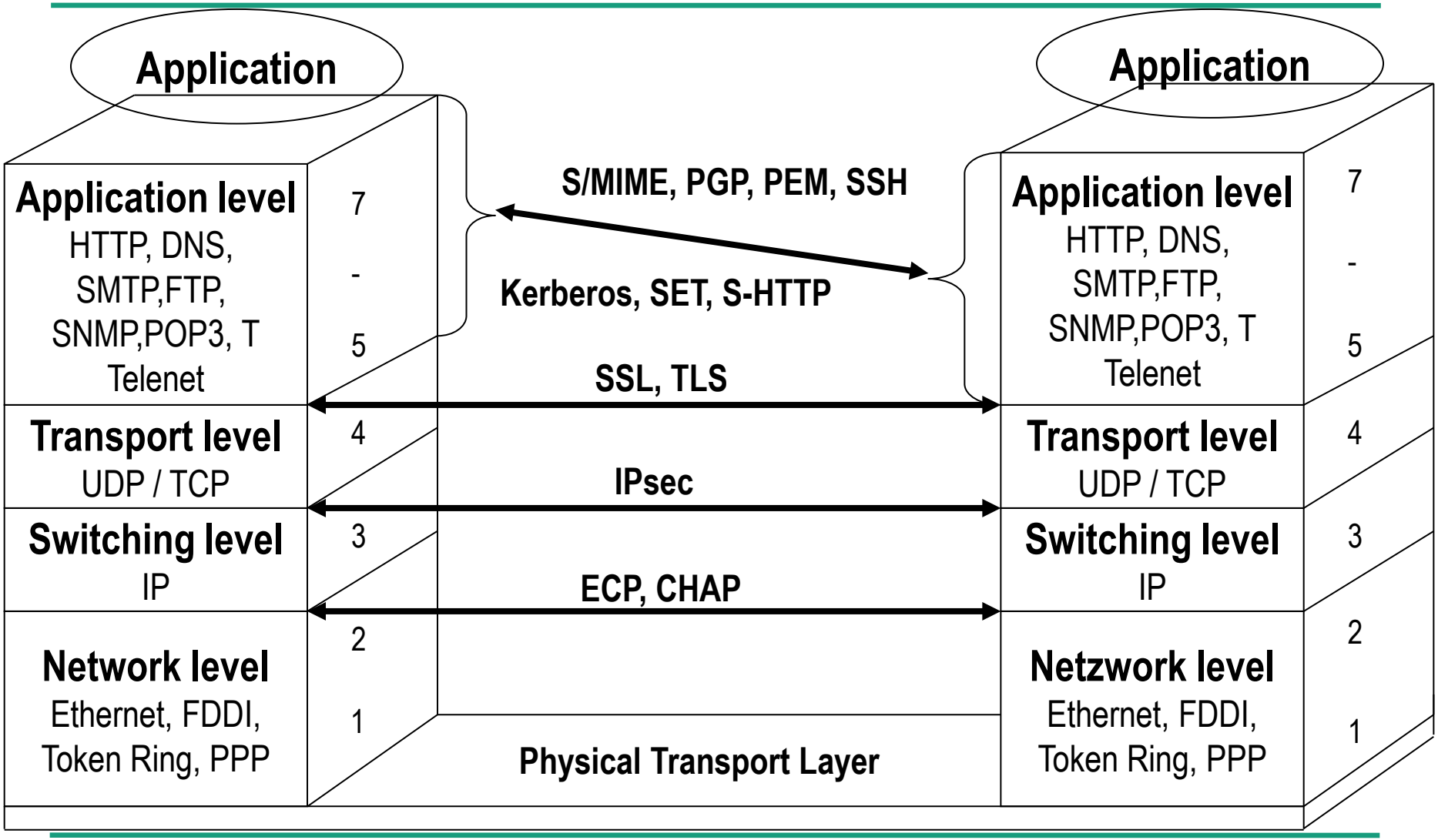
- forge identifications
- replay old identifications

Need to manage keys, perform electronic transactions, ...

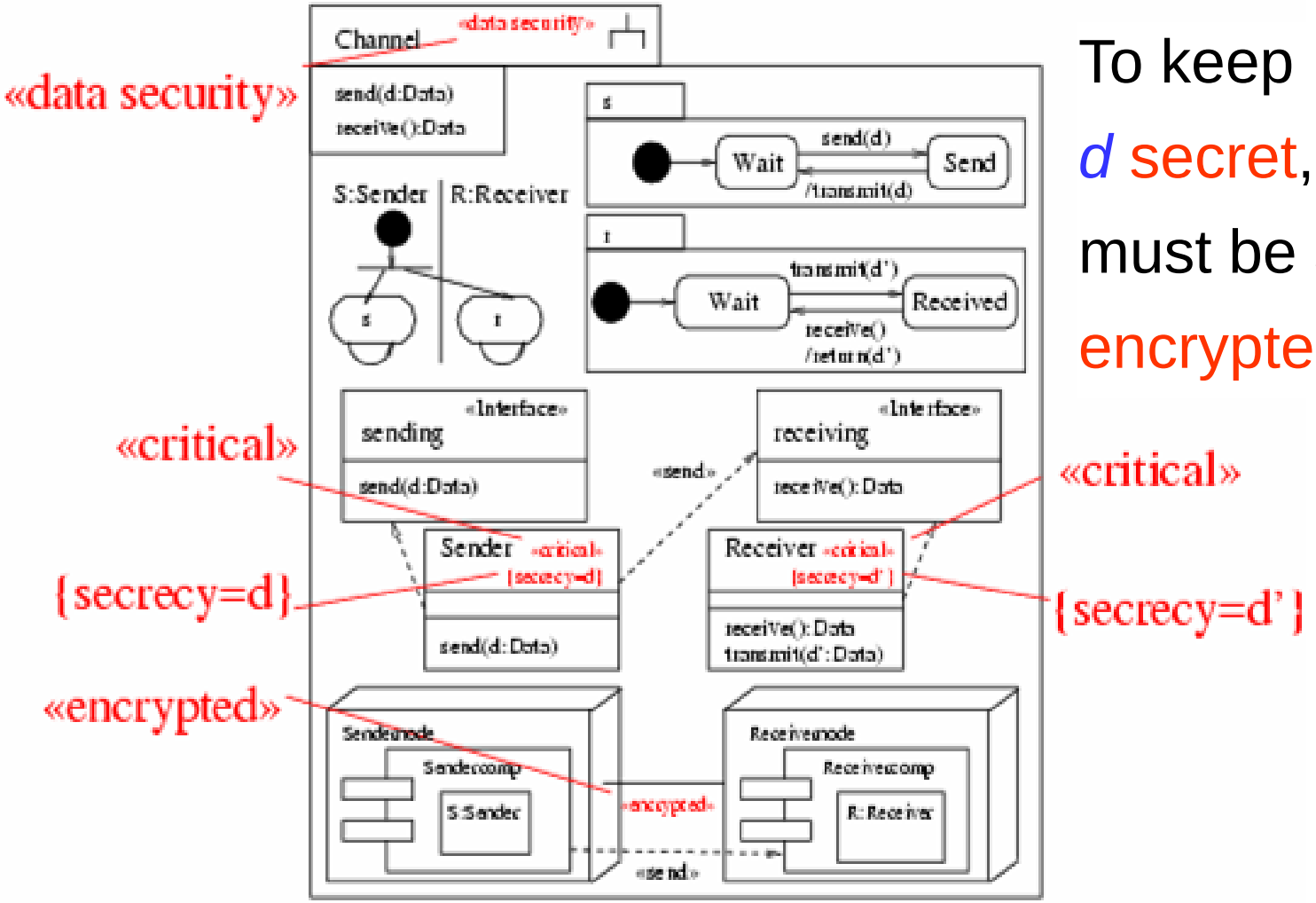
Use cryptographic protocols: Exchange of messages for distributing session keys, authenticating principals etc.

Notoriously hard to get right.

Encryption and Protocol Layers

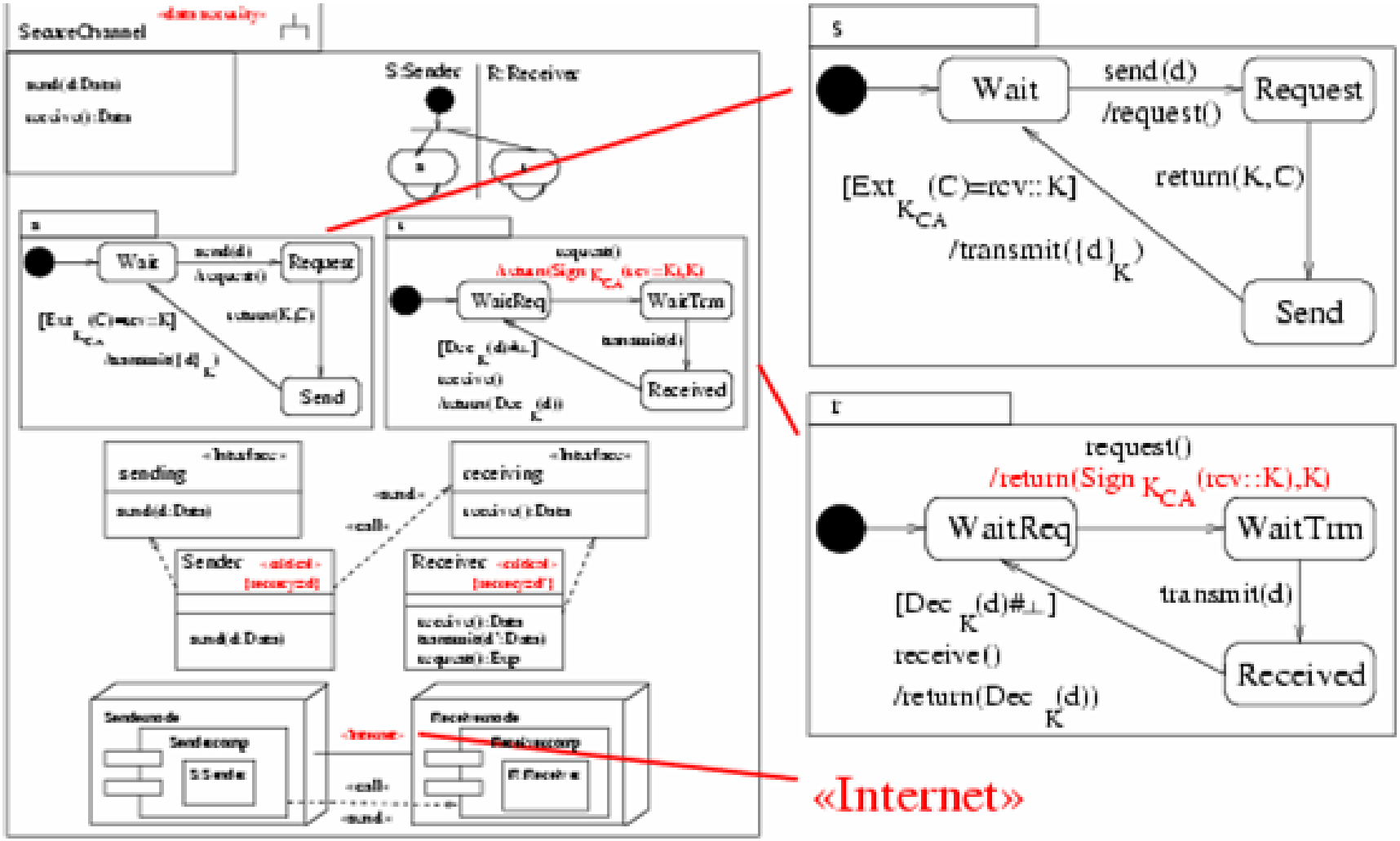


Using Protocols: Secure Channel



To keep *d* secret, must be sent encrypted.

Secure Channel Pattern: Solution



Exchange key and send encrypted data.

Fraunhofer ISST

Security Protocols

System distributed over untrusted networks.

„Adversary“ intercepts, modifies, deletes, inserts messages.

Cryptography provides security.

Cryptographic Protocol: Exchange of messages for distributing session keys, authenticating principals etc. using cryptographic algorithms

Security Protocols: Problems

Many protocols have vulnerabilities or subtleties for various reasons

- weak cryptography
- core message exchange
- interfaces, prologues, epilogues
- deployment
- implementation bugs

Security Analysis

Following Dolev, Yao (1982): To analyze system, verify against attacker model from threat scenarios in deployment diagrams who

- may participate in some protocol runs,
- knows some data in advance,
- may intercept messages on some links,
- may injects produced messages in some links
- may access certain nodes.

Adversaries

Model classes of adversaries.

May attack different parts of the system according to threat scenarios.

Example: insider attacker may intercept communication links in LAN.

To evaluate security of specification, simulate jointly with adversary model.

Cryptography

Keys are symbols, crypto-algorithms are abstract operations.

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.

Cryptographic Expressions I

Exp: quotient of term algebra generated from sets *Data*, *Keys*, *Var* of symbols using

- *_::_* (concatenation), *head(_)*, *tail(_)*,
- $(\_)^{-1}$ (inverse keys)
- *{ _ }_* (encryption)
- *Dec_( )* (decryption)
- *Sign_( )* (signing)
- *Ext_( )* (extracting from signature)

under equations …

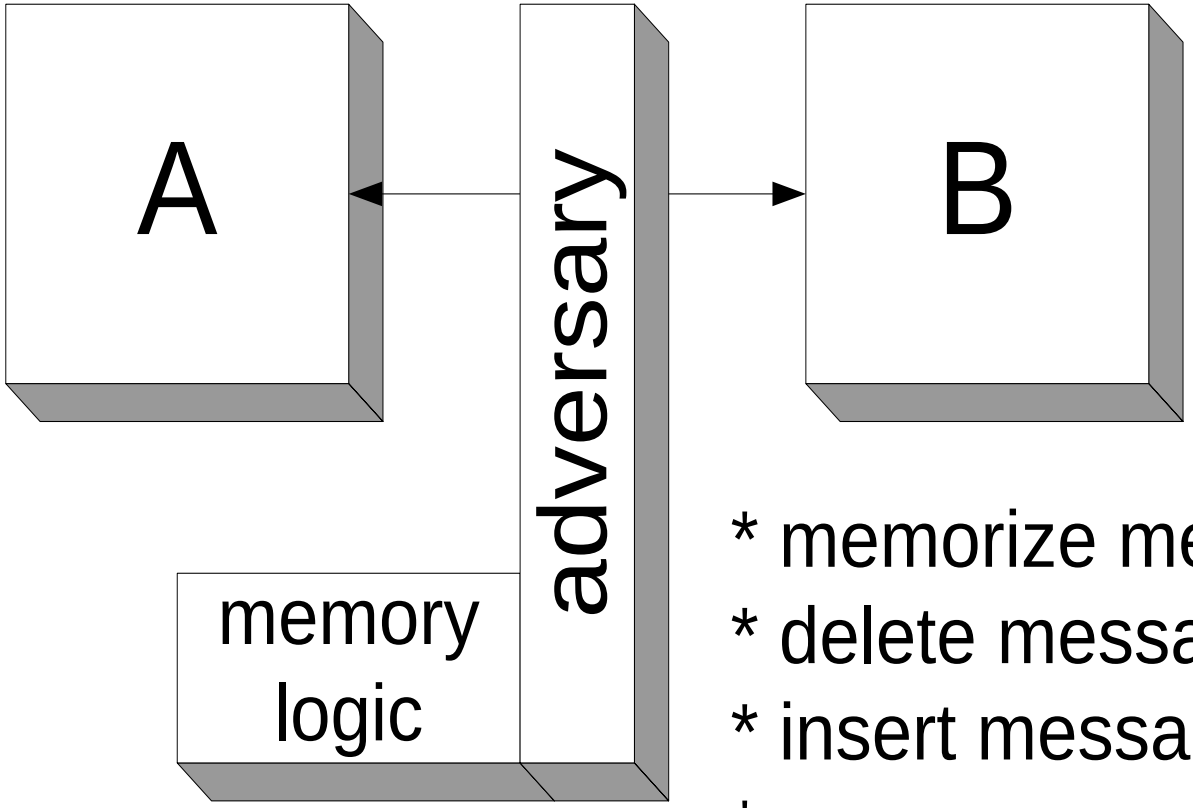
Cryptographic Expressions II

- $\forall E,K.Dec_K^{-1}(\{E\}_K)=E$
- $\forall E,K.Ext_K(Sign_K^{-1}(E))=E$
- $\forall E_1,E_2.head(E_1::E_2)=E_1$
- $\forall E_1,E_2.tail(E_1::E_2)=E_2$
- Associativity for $::$.

Write $E_1::E_2::E_3$ for $E_1::(E_2::E_3)$ and $fst(E_1::E_2)$ for $head(E_1::E_2)$ etc.

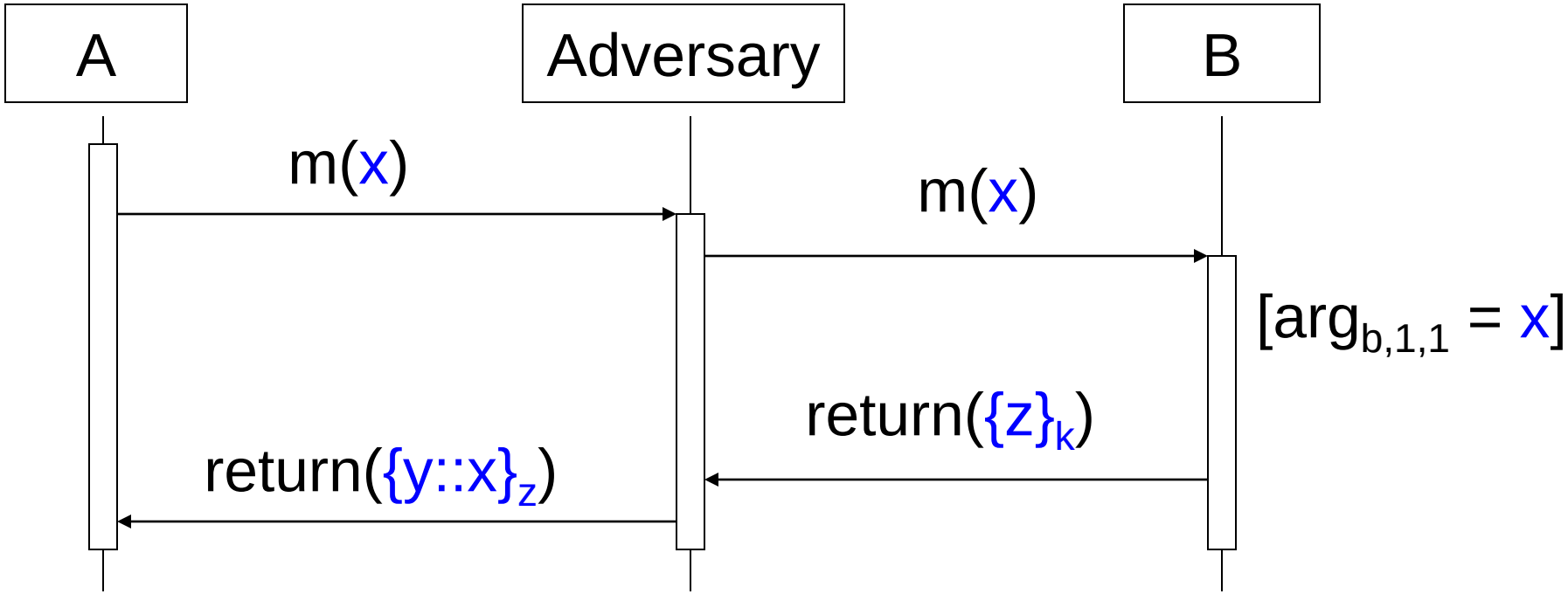
Can include further crypto-specific primitives and laws (XOR, …).

Adversary Model



* memorize message
* delete message
* insert message
* compose own message
* use cryptographic primitives

Protocol: Attack Scenario



Adversary knowledge:

k^{-1}, y, x

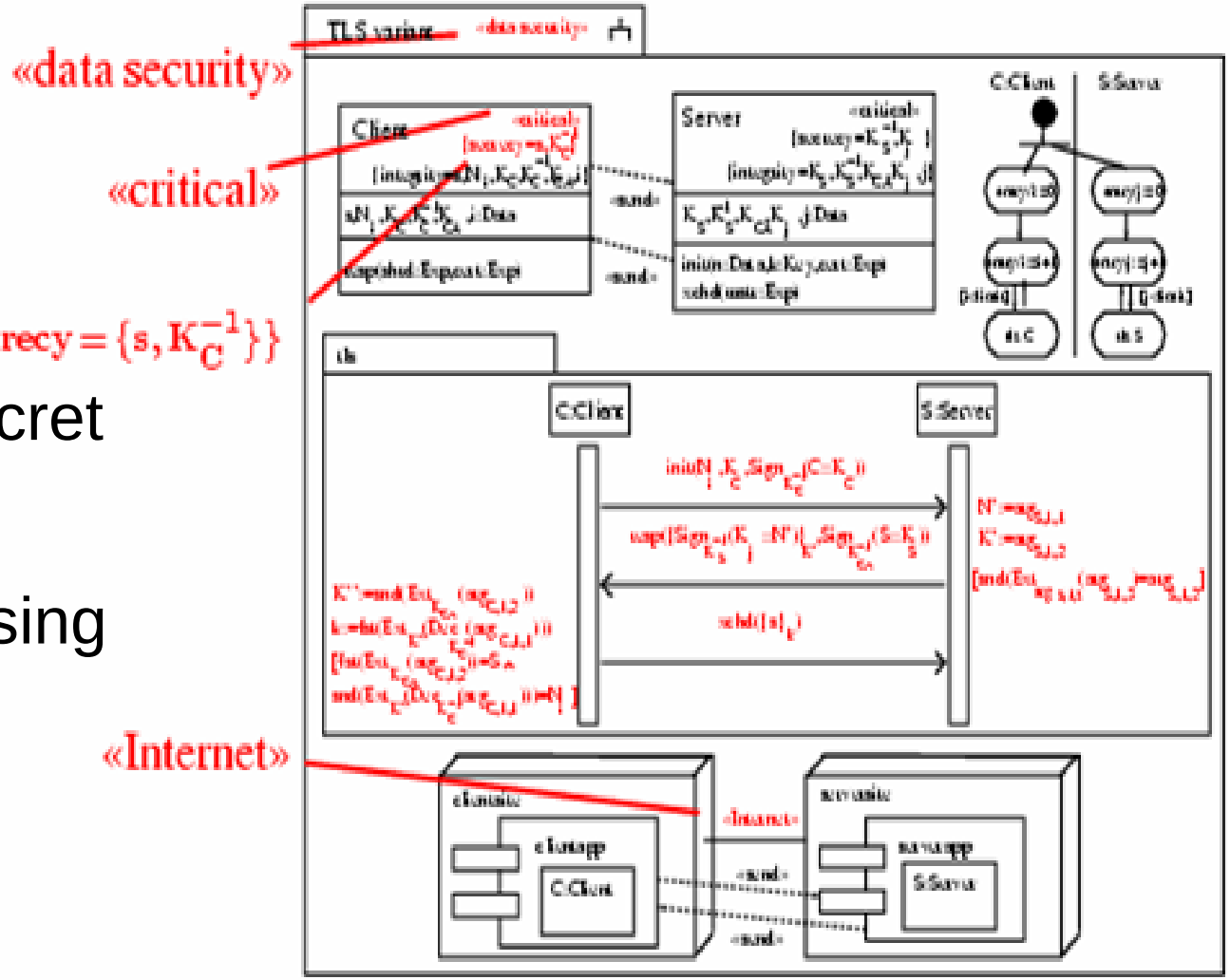
$\{z\}_k$, z

- $\forall e, k. \mathcal{D}ec_{k^{-1}}(\{e\}_k) = e$

Proposed Variant of TLS (SSL)

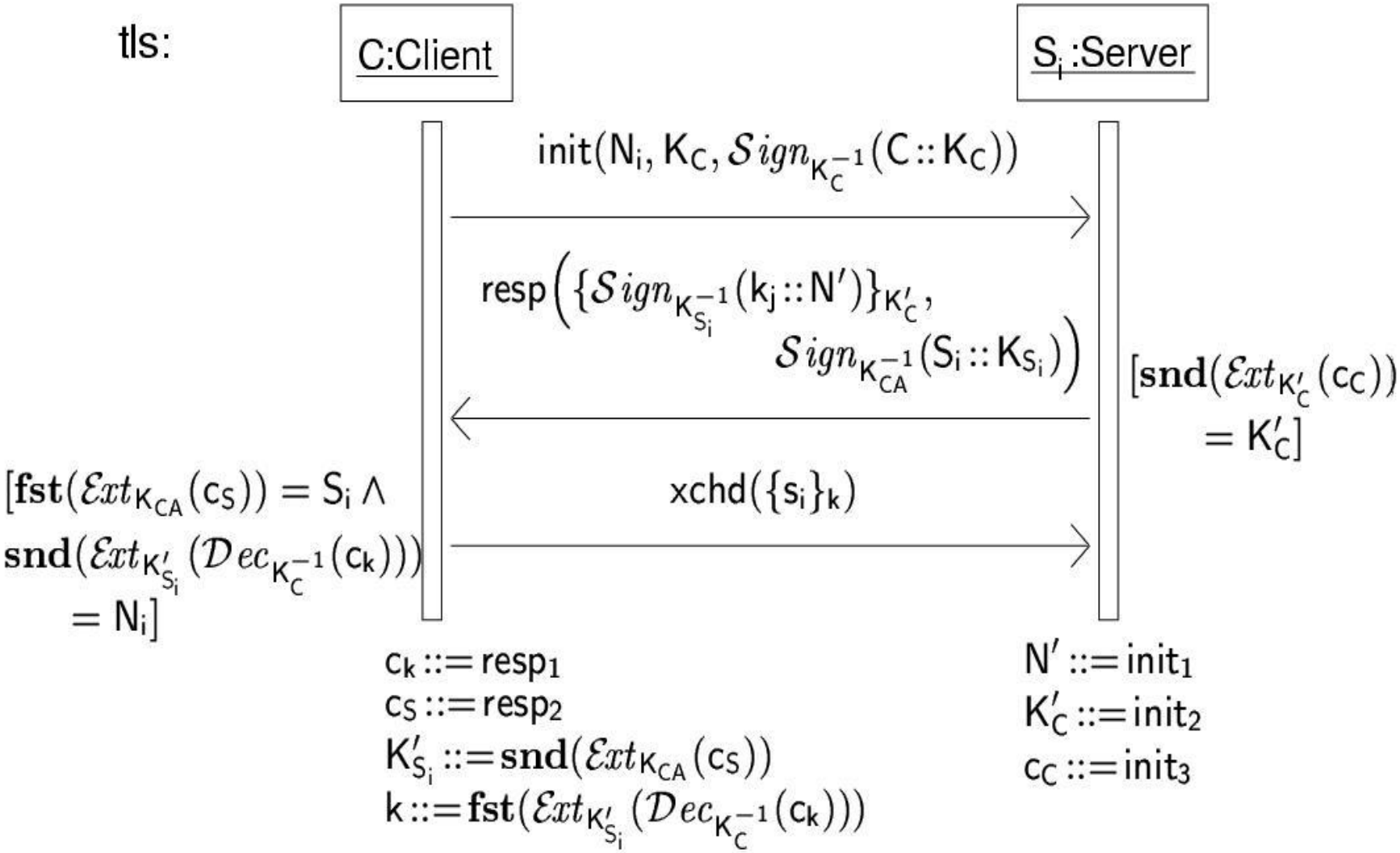
IEEE Infocom 1999.

Goal: send secret protected by session key using fewer server resources.



Protocol

tls:



Man-in-the-Middle Attack

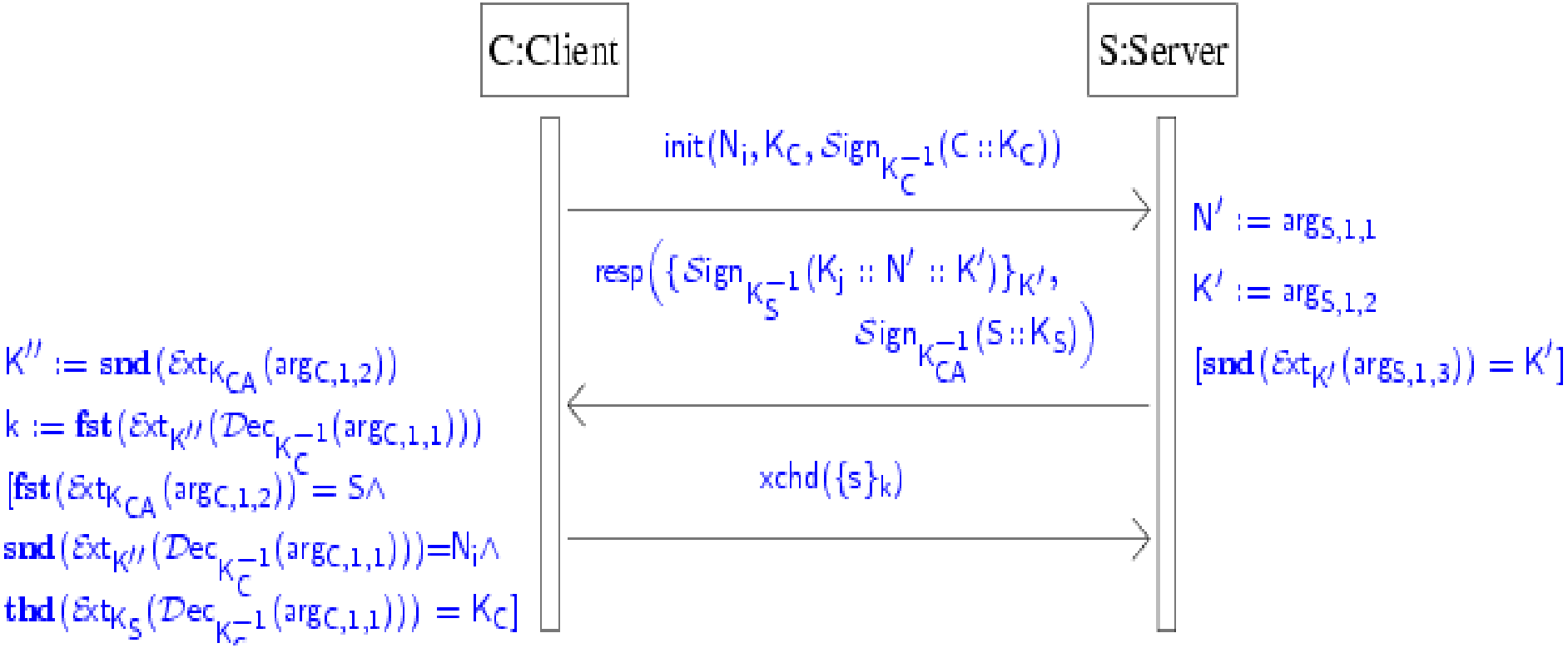
$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

$$A \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_A}::Sign_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_C}::Sign_{K_{CA}^{-1}}(S::K_S)} A$$

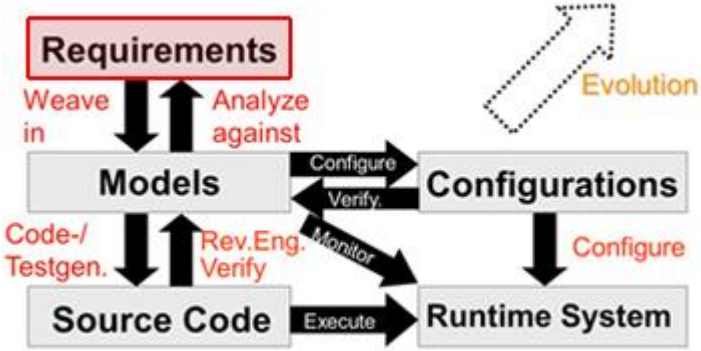
$$C \xrightarrow{\{s\}_{K_j}} A \xrightarrow{\{s\}_{K_j}} S$$

The Fix



Can prove that „secure“ (in precise sense).

Roadmap



1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

Adversary Knowledge

Specify set K_A^0 of initial knowledge of an adversary of type *A*. Let K_A^{n+1} be the Exp-subalgebra generated by K_A^n and the expressions received after *n+1*st iteration of the protocol.

Definition (Dolev, Yao 1982).

S keeps secrecy of *M* against attackers of type *A* if there is no *n* with $M \in K_A^n$.

Security Analysis in First-order Logic

Approximate adversary knowledge set from above:

Predicate *knows(E)* meaning that adversary may get to know *E* during the execution of the system.

E.g. secrecy requirement:
For any secret *s*, check whether can derive *knows(s)* from model-generated formulas using automatic theorem prover.

First-order Logic: Basic Rules

For initial adversary knowledge (K^0): Define *knows(E)* for any *E* initially known to the adversary (protocol-specific, e.g. K_A , K_A^{-1}).

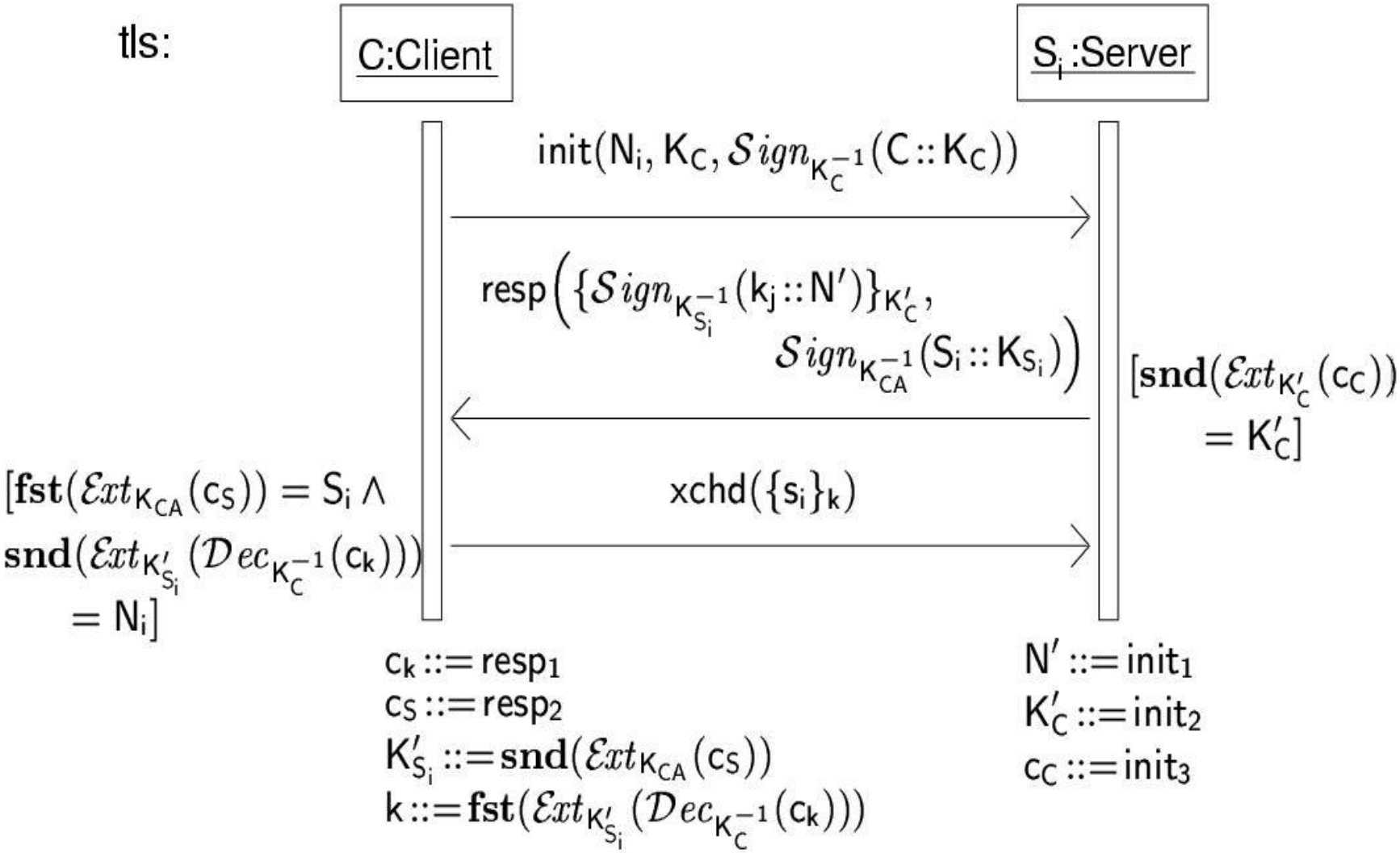
Define above equations.

For evolving knowledge (K^n) define

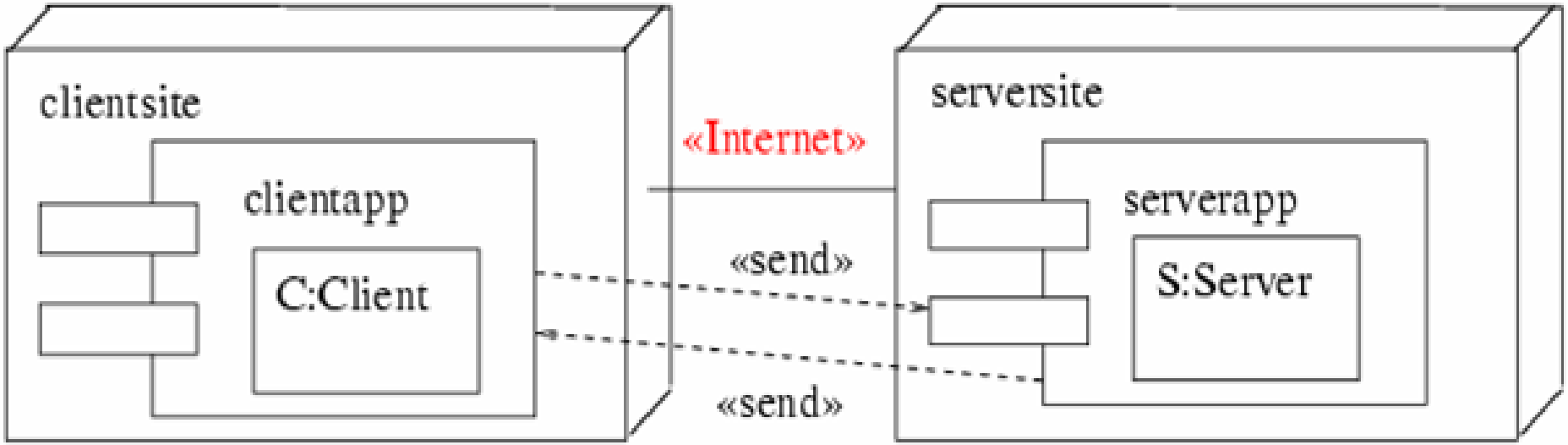
$$\forall E_1,E_2.(knows(E_1) \wedge knows(E_2) \Rightarrow knows(E_1::E_2) \wedge knows(\{E_1\}_{E2}) \wedge knows(Dec_{E2}(E_1)) \wedge knows(Sign_{E2}(E_1)) \wedge knows(Ext_{E2}(E_1)))$$

$$\forall E.(knows(E) \Rightarrow knows(head(E)) \wedge knows(tail(E)))$$

Given Sequence Diagram …



… and Physical Layer Model …



Deployment diagram.

Derived adversary model: read, delete, insert data.

… Translate to 1st Order Logic

Connection (or statechart transition)

$TR1=(in(msg\_in),cond(msg\_in),out(msg\_out))$

followed by $TR2$ gives predicate $PRED(TR1)=$

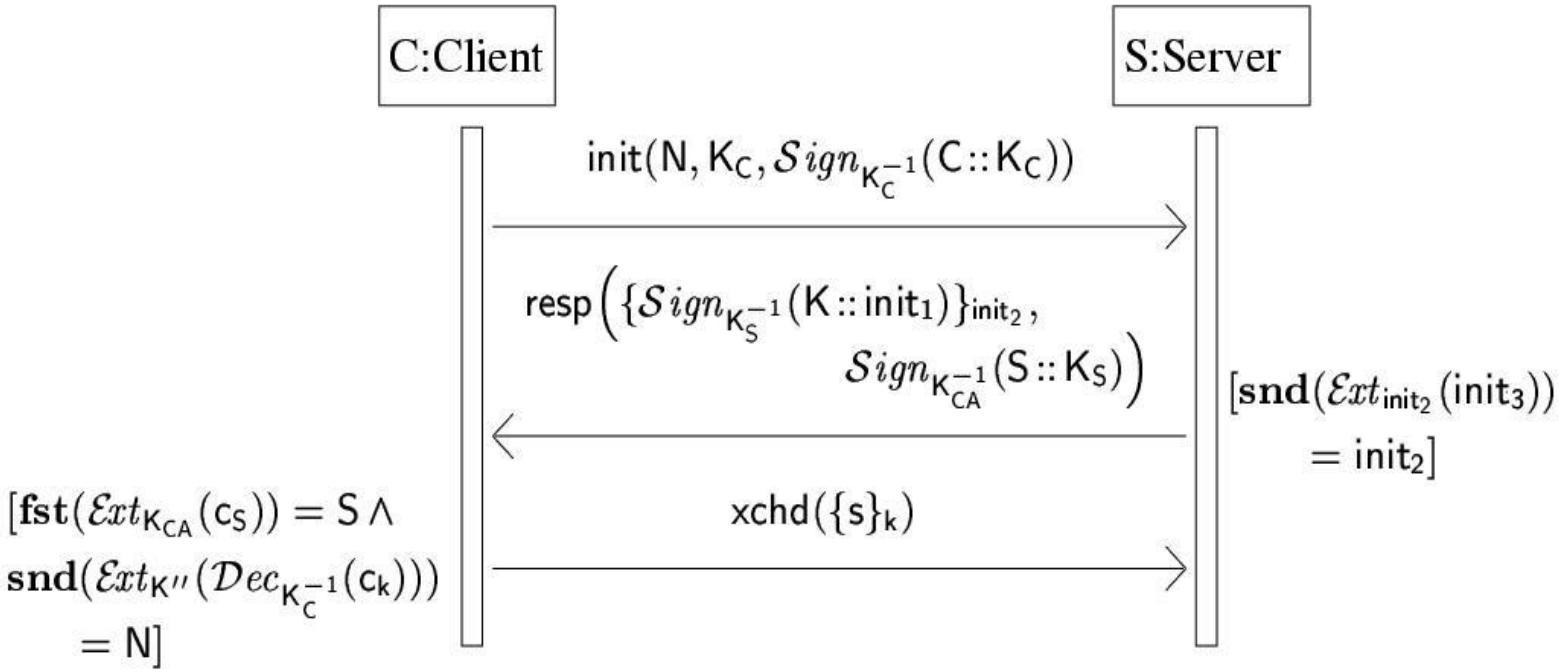
$$\forall msg\_in.\ [knows(msg\_in) \wedge cond(msg\_in) \Rightarrow knows(msg\_out) \wedge PRED(TR2)]$$

(Assume: order enforced (!).)

Can include senders, receivers in messages.

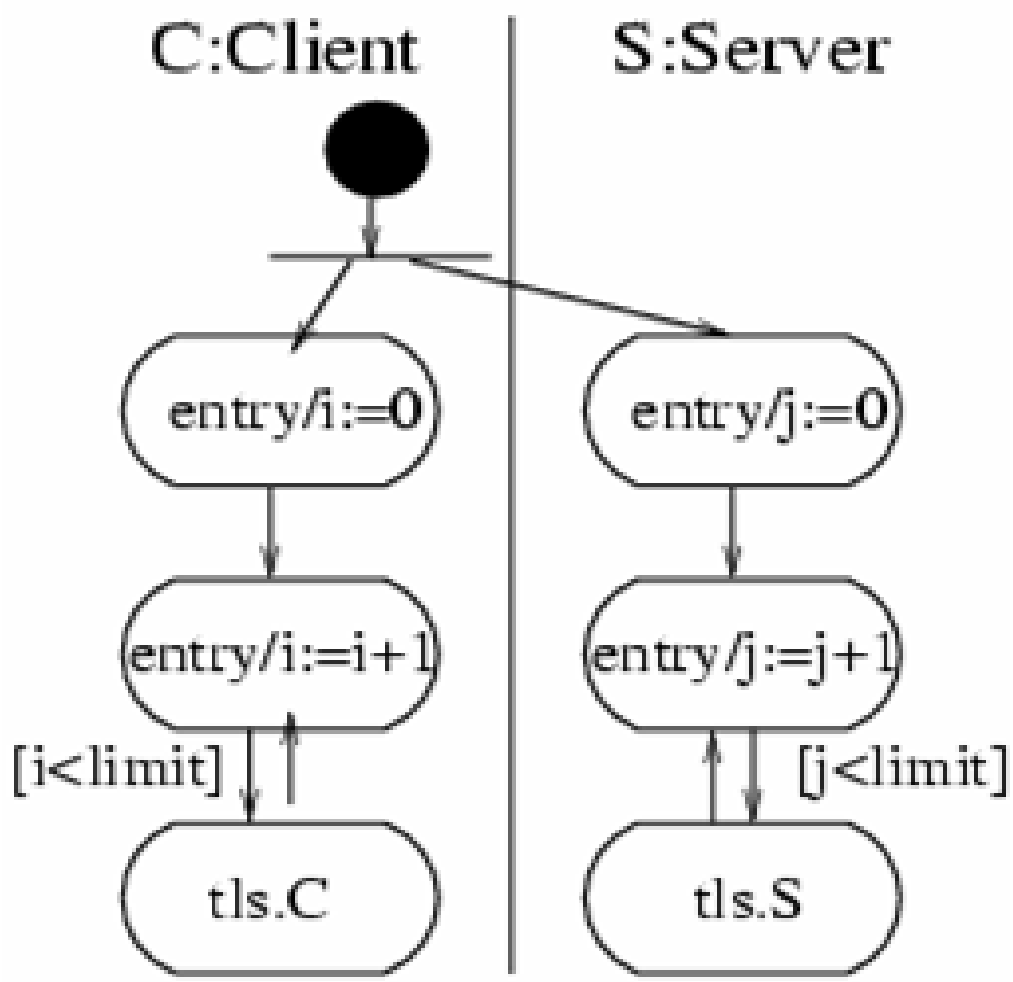
Abstraction: find all attacks, may have false positives.

Example



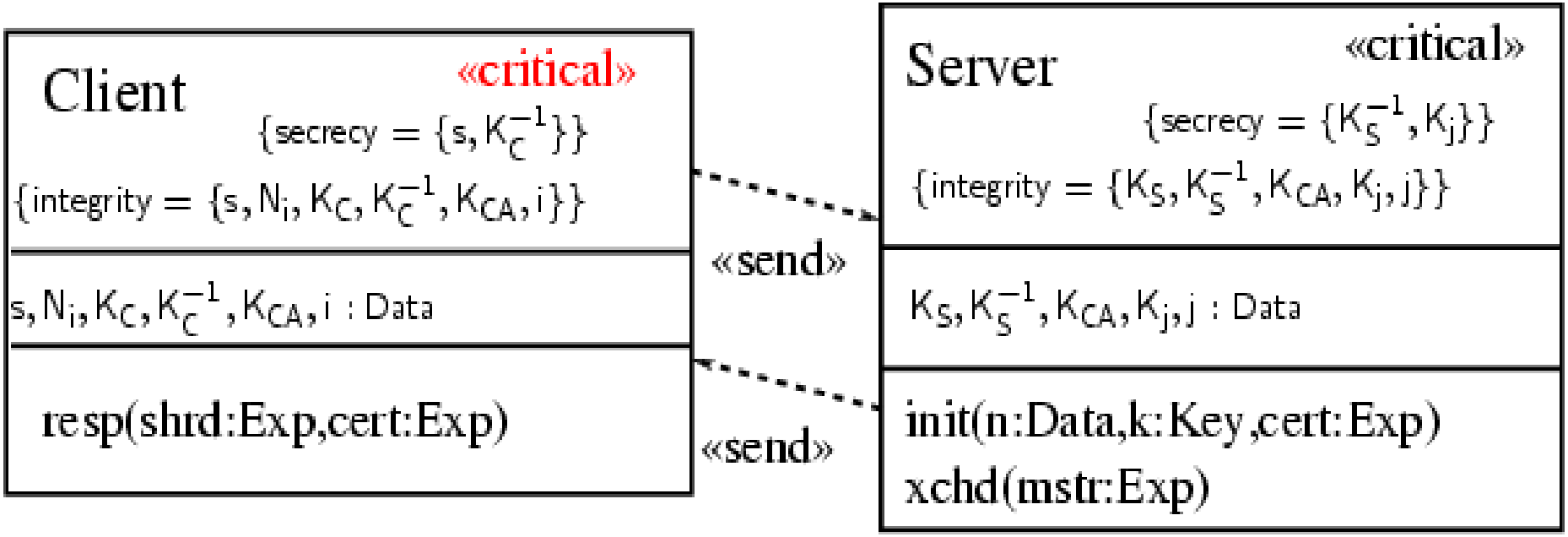
$$knows(N) \wedge knows(K_C) \wedge knows(Sign_{KC\text{-}1}(C::K_C))$$
$$\wedge\ \forall init_1, init_2, init_3.[knows(init_1) \wedge knows(init_2) \wedge$$
$$knows(init_3) \wedge snd(Ext_{init2}(init_3)) = init_2$$
$$\Rightarrow knows(\{Sign_{KS\text{-}1}(\ldots)\}_{\ldots}) \wedge [\ldots] \wedge [\ldots \Rightarrow \ldots]\ldots]$$

Execute in System Context



Activity diagram.

Formulate Data Security Requirements



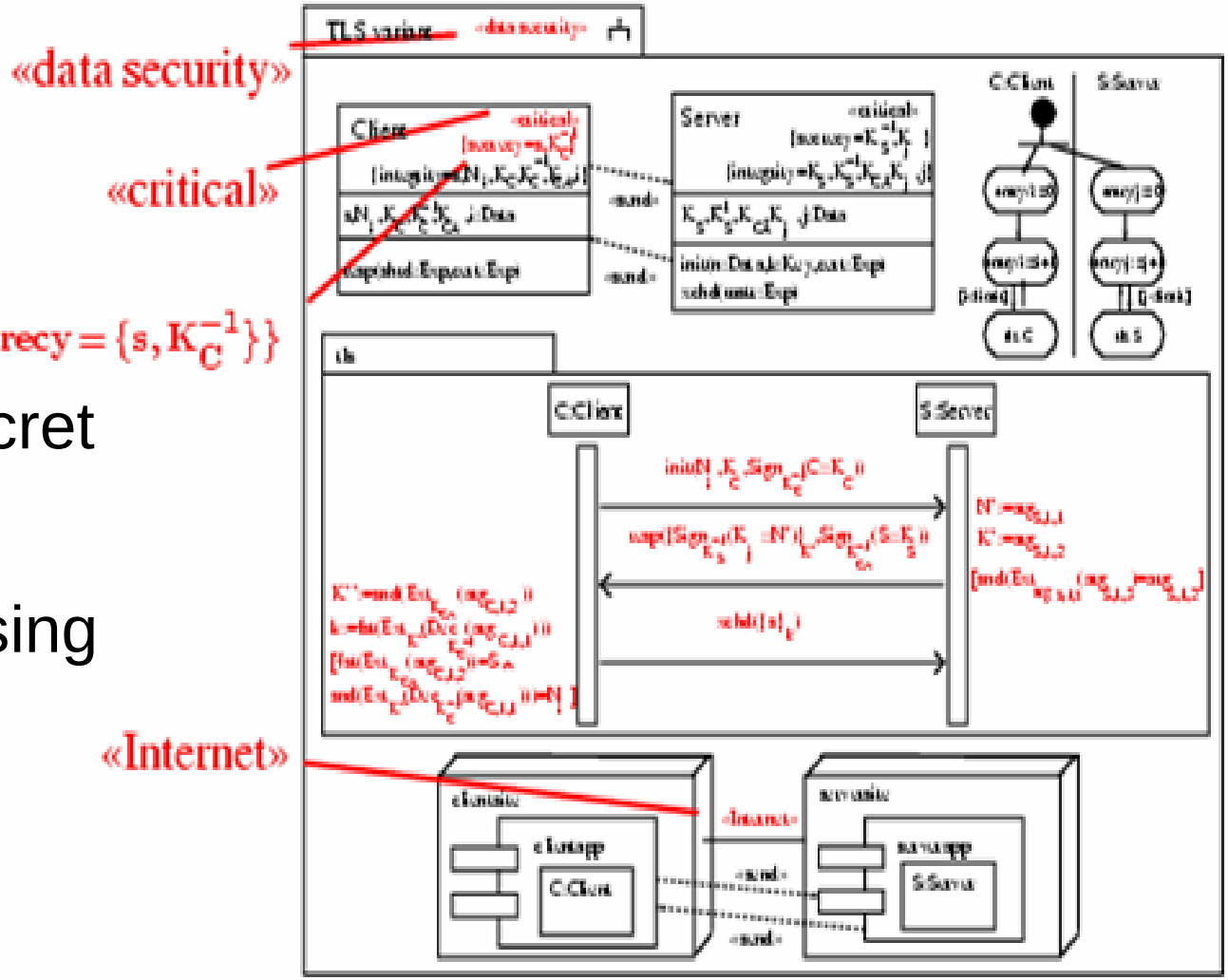
Class diagram.

Gives conjecture: *knows(s)* derivable ?

Proposed Variant of TLS (SSL)

IEEE Infocom 1999.

Goal: send secret protected by session key using fewer server resources.



TLS Variant in TPTP Notation I

```
input_formula(tls_abstract_protocol,axiom,(
 ![ArgS_11, ArgS_12, ArgS_13, ArgC_11, ArgC_12] : (
   ![DataC_KK, DataC_k, DataC_n] : (
      % Client -> Attacker (1. message)
      (      knows(n)
           & knows(k_c)
           & knows(sign(conc(c, k_c), inv(k_c) ) )  )
    & % Server -> Attacker (2. message)
      (  (   knows(ArgS_11)
           & knows(ArgS_12)
           & knows(ArgS_13)
           & ( ? [X] : equal( sign(conc(X, ArgS_12), inv(ArgS_12) ),
                              ArgS_13 ) )  )
      => (   knows(enc(sign(conc(kgen(ArgS_12), ArgS_11), inv(k_s) ),
                       ArgS_12 ) )
           & knows(sign(conc(s, k_s), inv(k_ca) ) ) )  )
```

TLS Variant in TPTP Notation II

```
  & % Client -> Attacker (3. message)
    (  (   knows(ArgC_11)
         & knows(ArgC_12)
         & equal(sign(conc(s, DataC_KK), inv(k_ca)), ArgC_12 )
         & equal(enc(sign(conc(DataC_k, DataC_n), inv(DataC_KK) ),
                     k_c), ArgC_11 )
         & ( ? [DataC_ks] : equal(sign(conc(s, DataC_ks), inv(k_ca) ),
                                  ArgC_12 ) )
         & equal(enc(sign(conc(DataC_k, n), inv(DataC_KK) ), k_c),
                 ArgC_11 )
         & equal(enc(sign(conc(DataC_k, DataC_n), inv(DataC_KK) ), k_c),
                 ArgC_11 )
       )
    => ( knows(symenc(secret, DataC_k)) )  )
) )  )).
```

```
E-SETHEO csp03 single processor running on host ...
(c) 2003 Max-Planck-Institut fuer Informatik and
    Technische Universitaet Muenchen

tlsvariant-freshkey-check.tptp
...
time limit information: 300 total (entering statistics module).
problem analysis ...
testing if first-order ...
first-order problem
...
statistics: 19 0 7 46 3 6 2 0 1 2 14 8 0 2 28 6
...
schedule selection: problem is horn with equality (class he).
schedule:605 3 300 597
...
entering next strategy 605 with resource 3 seconds.
...
analyzing results ...
proof found
time limit information: 298 total / 297 strategy (leaving wrapper).
...
e-SETHEO done. exiting
```

Surprise …

Attack

… Which Means:

Can derive *knows(s)* (!).

That is: Protocol does not preserve secrecy of *s* against adversaries.

➔ Completely insecure wrt stated goals.

But why ?

Could look at proof tree.

Or: use prolog-based attack generator.

Man-in-the-Middle Attack

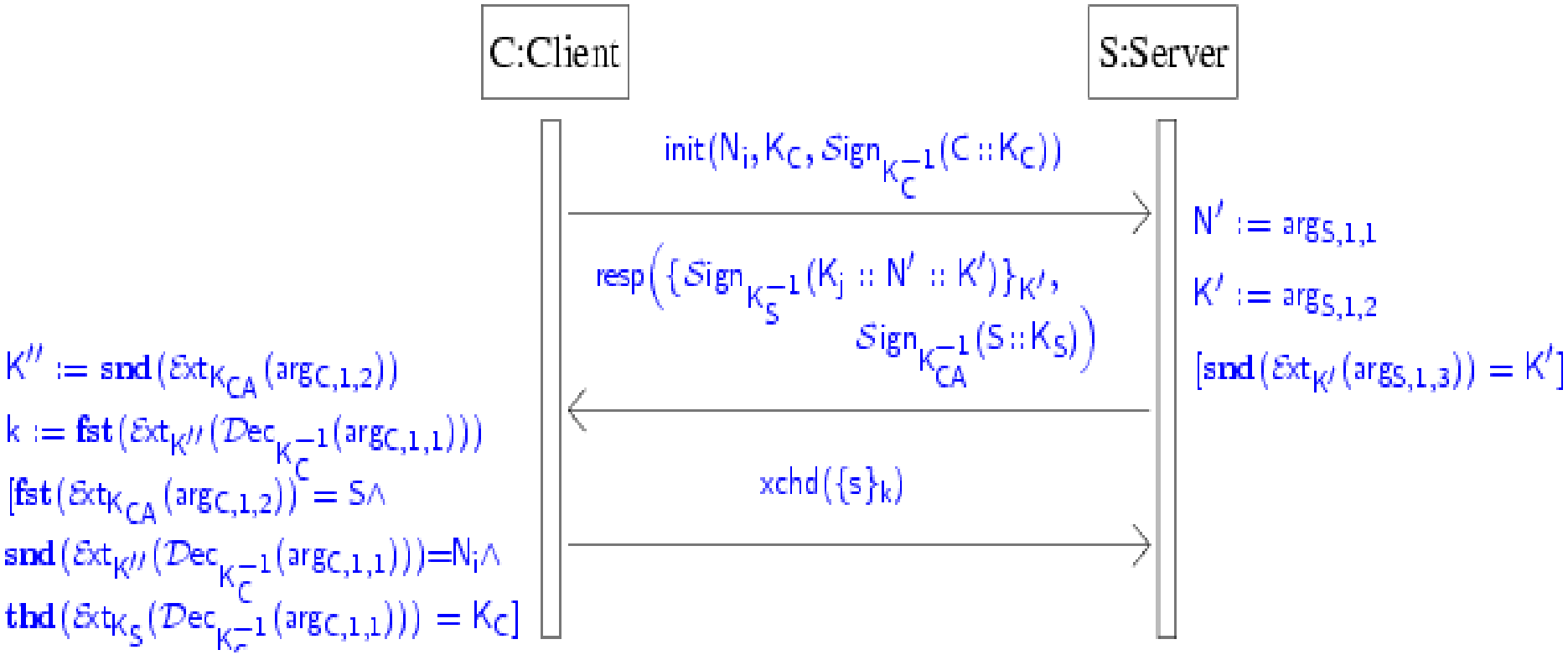
$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

$$A \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_A}::Sign_{K_{CA}^{-1}}(S::K_S)} S$$

$$C \xleftarrow{\{Sign_{K_S^{-1}}(K_j::N_i)\}_{K_C}::Sign_{K_{CA}^{-1}}(S::K_S)} A$$

$$C \xrightarrow{\{s\}_{K_j}} A \xrightarrow{\{s\}_{K_j}} S$$

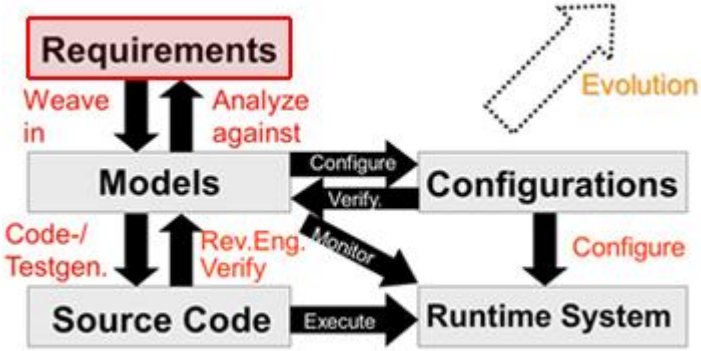
The Fix



e-Setheo: Proof that *knows(s)* not derivable.

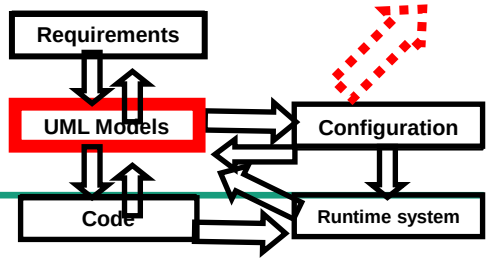
Note completeness of FOL (but also undecidability).

Roadmap



1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

General results: Security vs. Architecture



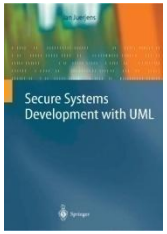
Idea: The model-based foundation allows one to investigate general questions regarding security preservation:

Security preservation vs. architectural principles:

- Horizontal layering of architectures [Safecomp'03]
- Modularization / Composition of architectural components [Concur'00]
- Service-oriented architectures [ICSOC'04]
- Aspect-oriented architectures [Models'05]

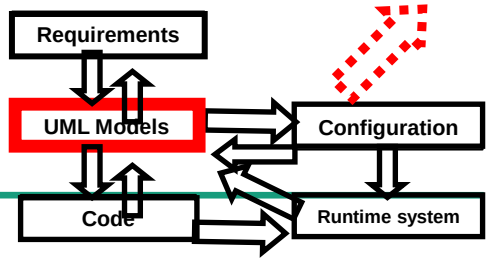
Security preservation vs. development techniques:

- Refinement of specifications [FME'01]
- Refactoring of architectures



For each: Theorem providing conditions for preservation of security.

Security vs. Refinement



Question: Under which conditions does refinement (i.e. concretization of specifications) preserve security properties ?

For behavior-preserving refinement one would expect security properties to be preserved.

„Refinement paradox“: In general not !

Observation: Problem: Mixture of nondeterminism for under-specification resp. as a security mechanism.

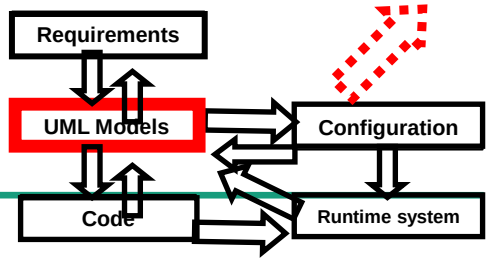
Idea: Separate the two on the modeling level.

Theorem: Then refinement preserves security requirements.

Theorem

- If P preserves secrecy of m and $P \rightsquigarrow Q$ then Q preserves secrecy of m.

Security vs. Modularization



Idea: Exploit architectural modularization for modular security verification.

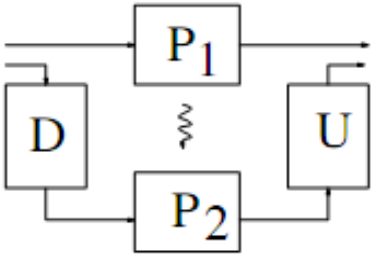
Question: Under which conditions does composition of components preserve security properties ?

Only works under suitable assumptions on other components.

Idea: Formalize as „Rely-guarantee“-condition.

$\Rightarrow$ Can verify components separately.

Validation: Java Secure Sockets Extension.



Theorem 5. *Let P_1, P_2, D and U be processes with $I_{P_1} = I_D$, $O_D = I_{P_2}$, $O_{P_2} = I_U$ and $O_U = O_{P_1}$ and such that D has a left inverse D' and U a right inverse U'. Let $m \in (\mathbf{Secret} \cup \mathbf{Keys}) \setminus \bigcup_{Q \in \{D', U'\}} (S_Q \cup K_Q)$.*

- *If P_1 preserves the secrecy of m and $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ then P_2 preserves the secrecy of m.*

Formal Foundation

For formal foundation, in particular for meta-level investigation of this security analysis approach, use formal semantics for a simplified fragment of UML, define using the formal approach of stream-processing functions. Consider the approach in the following.

[J. Jurjens: A domain-specific language for cryptographic protocols based on streams. J. Log. Algebr. Program. 78(2): 54-73 (2009)]

Cryptographic Expressions

$E ::=$	expression
d	data value ($d \in \mathcal{D}$)
N	unguessable value ($N \in \mathbf{Secret}$)
K	key ($K \in \mathbf{Keys}$)
$\mathsf{inp}(c)$	input on channel c ($c \in \mathsf{Channels}$)
x	variable ($x \in \mathbf{Var}$)
$E_1 :: E_2$	concatenation
$\{E\}_e$	encryption ($e \in \mathbf{Enc}$)
$\mathcal{D}ec_e(E)$	decryption ($e \in \mathbf{Enc}$)
$\mathcal{S}ign_e(E)$	signature creation ($e \in \mathbf{Enc}$)
$\mathcal{E}xt_e(E)$	signature extraction ($e \in \mathbf{Enc}$)

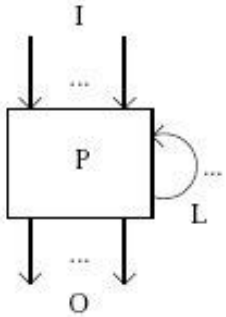
Executable Specifications

$p ::=$	programs
E	output expression ($E \in \mathbf{Exp}$)
$\mathit{either}\ p\ \mathit{or}\ p'$	nondeterministic branching
$\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'$	conditional ($E, E' \in \mathbf{Exp}$)
$\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p'$	determine if E is a key ($E \in \mathbf{Exp}$)
$\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'$	break up list into head::tail ($E \in \mathbf{Exp}$)

A process is of the form $P = (I, O, L, (p_c)_{c \in O \cup L})$ where

Motivation for system model:
as generic as possible.

- $I \subseteq \mathbf{Channels}$ (input channels)
- $O \subseteq \mathbf{Channels}$ (output channels)
- $L \subseteq \mathbf{Channels}$ (local channels)
- p_c: closed program with input channels in $I \cup L$ and output channel $c \in O \cup L$.



cf. [Broy, Stolen 2001]

Semantics: stream-processing functions

$[E](\vec{M}) = \{E(\vec{M})\}$	where $E \in \mathbf{Exp}$
$[\mathit{either}\ p\ \mathit{or}\ p'](\vec{M}) = [p](\vec{M}) \cup [p'](\vec{M})$	
$[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\vec{M}) = [p](\vec{M})$	if $[E](\vec{M}) = [E'](\vec{M})$
$[\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$	if $[E](\vec{M}) \neq [E'](\vec{M})$
$[\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p](\vec{M})$	if $[E](\vec{M}) \in \mathbf{Keys}$
$[\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$	if $[E](\vec{M}) \notin \mathbf{Keys}$
$[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p[h/x, t/y]](\vec{M})$	
if $[E](\vec{M}) = h :: t$ where $h \neq \varepsilon$ and h is not of the form $h_1 :: h_2$ for $h_1, h_2 \neq \varepsilon$	
$[\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p'](\vec{M}) = [p'](\vec{M})$	if $[E](\vec{M}) = \varepsilon$

Security Analysis using FOL

Knowledge enlargement by constructing expressions:

$$\forall E_1, E_2. (\mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2) \Rightarrow \mathsf{knows}(E_1 :: E_2) \wedge \mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(\mathcal{S}ign_{E_2}(E_1)))$$
$$\wedge(\mathsf{knows}(E_1 :: E_2) \Rightarrow \mathsf{knows}(E_1) \wedge \mathsf{knows}(E_2))$$
$$\wedge(\mathsf{knows}(\{E_1\}_{E_2}) \wedge \mathsf{knows}(E_2^{-1}) \Rightarrow \mathsf{knows}(E_1))$$
$$\wedge(\mathsf{knows}(\mathcal{S}ign_{E_2^{-1}}(E_1)) \wedge \mathsf{knows}(E_2) \Rightarrow \mathsf{knows}(E_1))$$

Security Analysis using FOL (2)

$$\phi(E) = \forall i_1, \ldots, i_n.(\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \mathsf{knows}(E(i_1, \ldots, i_n)))$$

$$\phi(\mathit{either}\ p\ \mathit{or}\ p') = \phi(p) \wedge \phi(p')$$

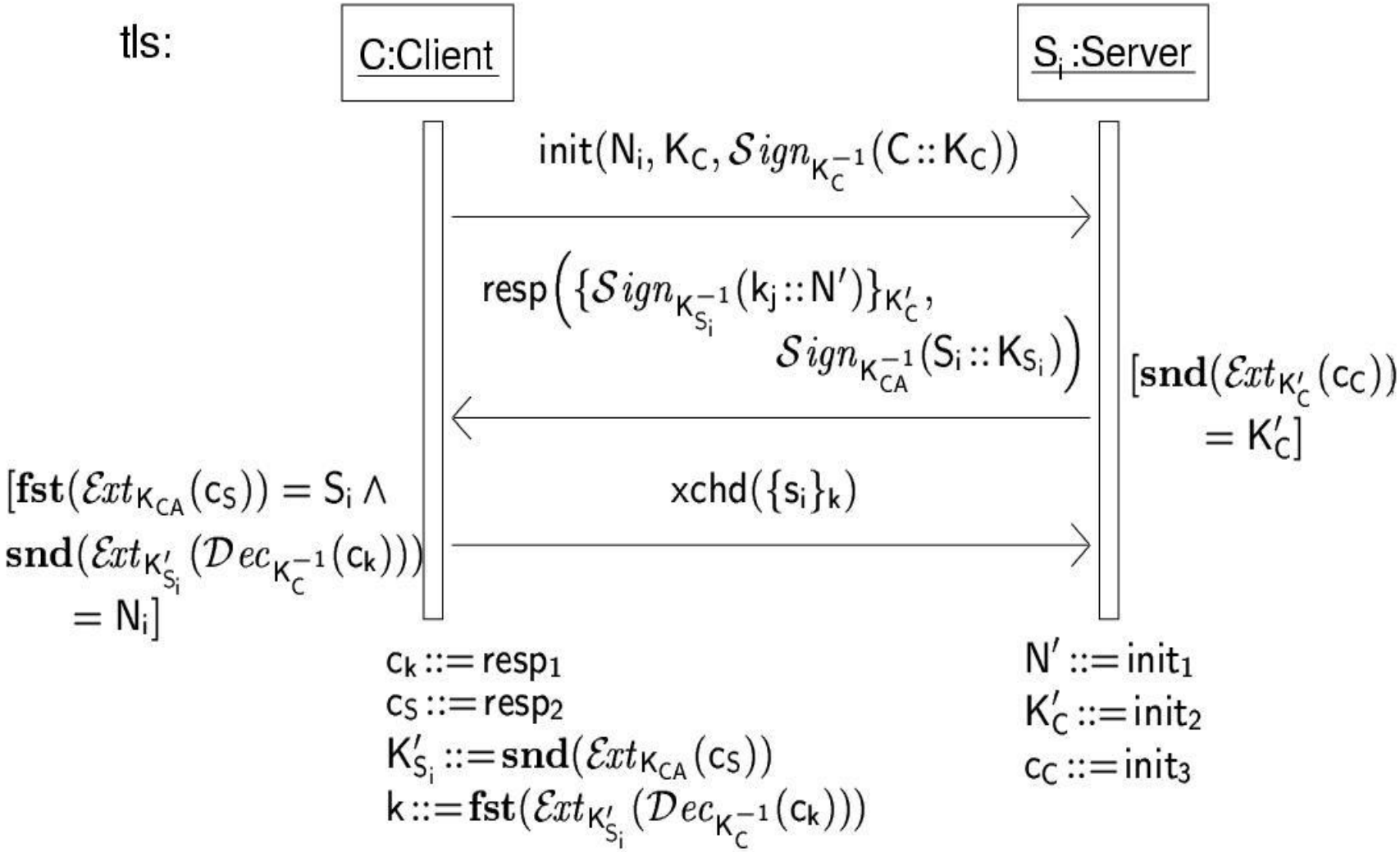
$$\begin{aligned}
&\phi(\mathit{if}\ E = E'\ \mathit{then}\ p\ \mathit{else}\ p') = \\
&\quad \forall i_1, \ldots, i_n.(\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n)) \Rightarrow \\
&\qquad (E(i_1, \ldots, i_n) = E'(i_1, \ldots, i_n) \Rightarrow \phi(p)) \\
&\qquad \wedge (E(i_1, \ldots, i_n) \neq E'(i_1, \ldots, i_n) \Rightarrow \phi(p')))
\end{aligned}$$

$$\begin{aligned}
&\phi(\mathit{case}\ E\ \mathit{of}\ \mathit{key}\ \mathit{do}\ p\ \mathit{else}\ p') = \\
&\quad \forall i_1, \ldots, i_n.(\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \\
&\qquad (\mathsf{key}(E(i_1, \ldots, i_n)) \Rightarrow \phi(p)) \wedge (\neg \mathsf{key}(E(i_1, \ldots, i_n)) \Rightarrow \phi(p')))
\end{aligned}$$

$$\begin{aligned}
&\phi(\mathit{case}\ E\ \mathit{of}\ x :: y\ \mathit{do}\ p\ \mathit{else}\ p') = \\
&\quad \forall i_1, \ldots, i_n, h, t.(\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \wedge E(i_1, \ldots, i_n) = h :: t \Rightarrow \phi(p[h/x, t/y])) \\
&\quad \wedge \forall i_1, \ldots, i_n.(\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \wedge \neg \exists h, t.E(i_1, \ldots, i_n) = h :: t \Rightarrow \phi(p'))
\end{aligned}$$

Example: TLS Variant

tls:



Executable Specification

$$
\begin{aligned}
c \stackrel{\text{def}}{=}\ & \mathit{if}\ \mathsf{inp}(l) = \varepsilon\ \mathit{then}\ N_C :: K_C :: \mathcal{S}ign_{K_C^{-1}}(C :: K_C) \\
& \quad \mathit{else}\ \mathit{case}\ \mathsf{inp}(s')\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
& \qquad \mathit{do}\ \mathit{case}\ \mathcal{E}xt_{\mathsf{inp}(a_C)}(s_3)\ \mathit{of}\ S :: x \\
& \qquad\quad \mathit{do}\ \mathit{if}\ \{\mathcal{D}ec_{K_C^{-1}}(s_2)\}_x = y :: N_C\ \mathit{then}\ \{m\}_y \\
& \qquad\qquad\qquad\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
& \qquad\quad \mathit{else}\ \text{abort} \\
& \qquad \mathit{else}\ \varepsilon
\end{aligned}
$$

$$l \stackrel{\text{def}}{=} 0$$

$$
\begin{aligned}
s \stackrel{\text{def}}{=}\ & \mathit{case}\ \mathsf{inp}(c')\ \mathit{of}\ c_1 :: c_2 :: c_3 \\
& \quad \mathit{do}\ \ \mathit{case}\ \mathcal{E}xt_{c_2}(c_3) \mathit{of}\ x :: c_2\ \mathit{do}\ N_S :: \{\mathcal{S}ign_{K_S^{-1}}(K_{CS} :: c_1)\}_{c_2} :: \mathsf{inp}(a_S) \\
& \qquad\qquad\qquad\qquad\qquad\qquad \mathit{else}\ \text{abort} \\
& \quad \mathit{else}\ \varepsilon
\end{aligned}
$$

$$r \stackrel{\text{def}}{=} \mathit{if}\ \mathcal{D}ec_{K_{CS}}(\mathsf{inp}(c')) \in \mathbf{Data} \cup \mathbf{Secret}\ \mathit{then}\ \mathcal{D}ec_{K_{CS}}(\mathsf{inp}(c'))\ \mathit{else}\ \varepsilon$$

$$a_C \stackrel{\text{def}}{=} K_{CA}$$

$$a_A \stackrel{\text{def}}{=} K_{CA} :: \mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S) :: \mathcal{S}ign_{K_{CA}^{-1}}(Z :: K_Z)$$

$$a_S \stackrel{\text{def}}{=} \mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S)$$

Translation to FOL

$$\mathsf{knows}(N_C :: K_C :: \mathcal{S}ign_{K_C^{-1}}(C :: K_C))$$

$$\wedge \forall s_1, s_2, s_3, a_1, x, y.(\mathsf{knows}(s_1) \wedge \mathsf{knows}(s_2) \wedge \mathsf{knows}(s_3) \wedge \mathsf{knows}(a_1) \wedge \{s_3\}_{a_1} = S :: x \wedge \{\mathcal{D}ec_{K_C^{-1}}(s_2)\}_x = y :: N_C \Rightarrow \mathsf{knows}(\{m\}_y))$$

$$\wedge \mathsf{sent}(0)$$

$$\wedge \forall c_1, c_2, c_3, a_1, x.(\mathsf{knows}(c_1) \wedge \mathsf{knows}(c_2) \wedge \mathsf{knows}(c_3) \wedge \mathsf{knows}(a_1) \wedge \{c_3\}_{c_2} = x :: c_2 \Rightarrow \mathsf{knows}(N_S :: \{\mathcal{S}ign_{K_S^{-1}}(K_{CS} :: c_1)\}_{c_2} :: a_1))$$

$$\wedge \mathsf{knows}(K_{CA})$$

$$\wedge \mathsf{knows}(K_{CA} :: \mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S) :: \mathcal{S}ign_{K_{CA}^{-1}}(Z :: K_Z))$$

$$\wedge \mathsf{knows}(\mathcal{S}ign_{K_{CA}^{-1}}(S :: K_S))$$

$$\wedge \mathsf{knows}(\mathsf{abort})$$

$$\wedge \mathsf{knows}(\varepsilon)$$

Can derive *knows(s)*
➔ does not preserve secrecy of *s*.

Refinement

Definition Q refines P ($P \rightsquigarrow Q$) if for each $\vec{s} \in \mathbf{Stream}_{I_P}$ have $[\![P]\!](\vec{s}) \supseteq [\![Q]\!](\vec{s})$.

Example $(\mathit{either}\ p\ \mathit{or}\ q) \rightsquigarrow p$

Theorem

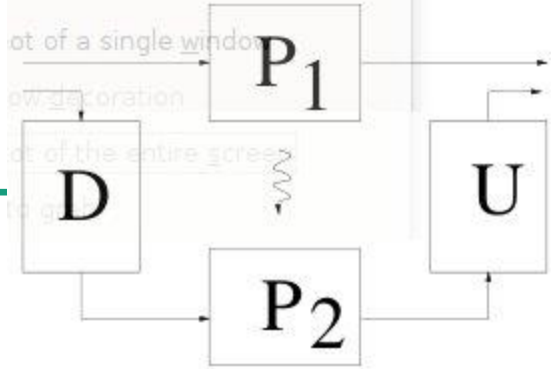
- If P preserves secrecy of m and $P \rightsquigarrow Q$ then Q preserves secrecy of m.
- If P preserves secrecy of m assuming C and $P \rightsquigarrow Q$ then Q preserves secrecy of m assuming C.

Restriction of Interface

Theorem 4

- *If P preserves the secrecy of m then P preserves the secrecy of m assuming C (for any condition C on the input/output behavior of the process P).*

Refactoring



Theorem 5 *Let P_1, P_2, D and U be processes with $I_{P_1} = I_D$, $O_D = I_{P_2}$, $O_{P_2} = I_U$ and $O_U = O_{P_1}$ and such that D has a left inverse D' and U a right inverse U'. Let $m \in (\mathbf{Secret} \cup \mathbf{Keys}) \setminus \bigcup_{Q \in \{D', U'\}} (S_Q \cup K_Q)$.*

- *If P_1 preserves the secrecy of m and $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ then P_2 preserves the secrecy of m.*
- *If P_1 preserves the secrecy of m assuming $C \subseteq \mathbf{Stream}_{O_{P_1}} \times \mathbf{Stream}_{I_{P_1}}$ and $P_1 \overset{(D,U)}{\rightsquigarrow} P_2$ then P_2 preserves the secrecy of m assuming $[\![U']\!] \circ C \circ [\![D']\!]$.*

Change up to Set of Behaviours

We consider the situation where a process has changed arbitrarily, except that for a set of input/output behaviors C it has remained the same.

Definition 3 Let P_1 and P_2 be processes with $I_{P_1} = I_{P_2}$ and $O_{P_1} = O_{P_2}$. We define $P_1 \rightsquigarrow_C P_2$ for a total relation $C \subseteq \mathbf{Stream}_{O_{P_1}} \times \mathbf{Stream}_{I_{P_1}}$ to hold if for each $\vec{s} \in \mathbf{Stream}_{I_{P_1}}$ and each $\vec{t} \in [\![P_2]\!]$, $(\vec{t}, \vec{s}) \in C$ implies $\vec{t} \in [\![P_1]\!]$.

Theorem 6
Given total relations $C, D \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$ with $C \subseteq D$, if P preserves the secrecy of m assuming C and $P \rightsquigarrow_D P'$ then P' preserves the secrecy of m assuming C.

Rely-Guarantee

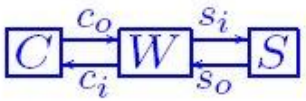
A obeys $C \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$ if
for all $\vec{s} \in \mathbf{Stream}_{I_A}$ and $\vec{t} \in [\![A]\!](\vec{s})$, have $(\vec{s}\lfloor_O, \vec{t}\lfloor_I) \in C$.

Definition P preserves the secrecy of m assuming $C \subseteq \mathbf{Stream}_{O_P} \times \mathbf{Stream}_{I_P}$ if exists no A obeying C such that $[\![P]\!] \otimes [\![A]\!]_r$ may eventually output m (and $m \notin S_A \cup K_A$).

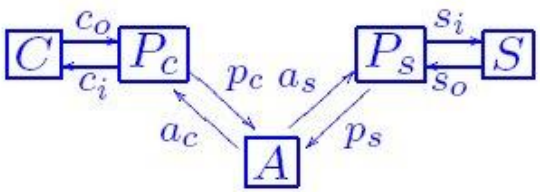
Example $p \stackrel{\text{def}}{=} \mathit{if}\ c = \mathbf{password}\ \mathit{then}\ \mathbf{secret}\ \mathit{else}\ \varepsilon$ preserves secrecy of secret assuming $C = \{(\vec{t}, \vec{s}) : \forall n. \vec{s}_n \neq \mathbf{password}\}$.

Example: Introducing Secure Channel

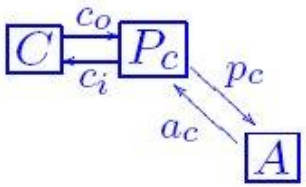
Want confidential channel W from C to S:



Transport layer vulnerable against active attacks:



Implement using handshake protocol (here client side).



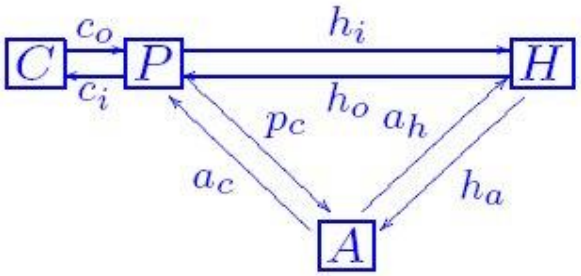
Want P_c so that for each suitable C: $C \otimes P_c$ sends out m from C encrypted under negotiated key $K \in \mathbf{Keys}$ to network while preserving secrecy.

Secure Channel (overview)

First: P_c nondeterministic sum of outputs needed for handshake and sending encrypted secret.

For any C: $C \otimes P_c$ preserves secrecy of m.

Next, split P_c into H (handshake protocol) and P (encrypts data from C under key from H and sends out to network):



Have conditional interface refinement from P_c to $P \otimes H$.

Use this to show that for each suitable C:
$C \otimes P \otimes H$ preserves secrecy of m.

Secure Channel: abstraction

First step: P_c nondeterministic sum of possible outputs.

$$p_c \stackrel{\text{def}}{=} \textit{either if } c_o = \varepsilon \textit{ then } \varepsilon \textit{ else } \{c_o\}_K \textit{ or } c_K$$

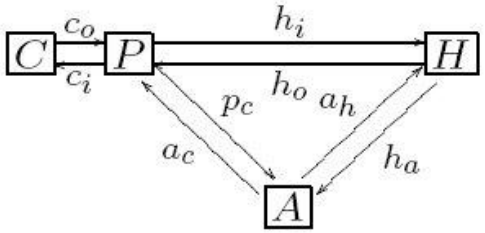
$$c_i \stackrel{\text{def}}{=} \textit{either } \varepsilon \textit{ or } \mathsf{ok}$$

$$\begin{aligned}
c_K \stackrel{\text{def}}{=} \; & \textit{either } N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C) \\
& \textit{or case } a_c \textit{ of } s_1 :: s_2 :: s_3 \\
& \quad \textit{do case } \mathcal{D}ec_{K_{AC}}(s_3) \textit{ of } S :: x \\
& \quad\quad \textit{do if } \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C \textit{ then } \{K\}_y \textit{ else } \mathsf{abort} \\
& \quad\quad \textit{else } \mathsf{abort} \\
& \quad \textit{else } \mathsf{abort}
\end{aligned}$$

Secure Channel: concretization (1)

For any $C(n)$, $C(n) \otimes P_c$ preserves secrecy of n.

Next, split P_c into H and P:



$$
\begin{aligned}
h_a \stackrel{\text{def}}{=} \; & \mathit{if}\ h_i = \varepsilon\ \mathit{then}\ N_C :: K_C :: \mathcal{D}ec_{K_C^{-1}}(C :: K_C) \\
& \quad \mathit{else}\ \mathit{case}\ a_h\ \mathit{of}\ s_1 :: s_2 :: s_3 \\
& \qquad \mathit{do}\ \mathit{case}\ \mathcal{D}ec_{K_{AC}}(s_3)\ \mathit{of}\ S :: x \\
& \qquad\quad \mathit{do}\ \mathit{if}\ \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C\ \mathit{then}\ \{m\}_y\ \mathit{else}\ \text{abort} \\
& \qquad\quad \mathit{else}\ \text{abort} \\
& \qquad \mathit{else}\ \text{abort}
\end{aligned}
$$

Secure Channel: concretization (2)

$$
\begin{aligned}
h_o &\stackrel{\text{def}}{=} \textit{if } h_i = \varepsilon \textit{ then } \varepsilon \\
&\qquad \textit{else case } a_h \textit{ of } s_1 :: s_2 :: s_3 \\
&\qquad\quad \textit{do case } \mathcal{D}ec_{K_{AC}}(s_3) \textit{ of } S :: x \\
&\qquad\qquad \textit{do if } \{\mathcal{D}ec_{K_C}(s_2)\}_x = y :: N_C :: K_C \textit{ then } \mathsf{finished} \textit{ else } \varepsilon \\
&\qquad\qquad \textit{else } \varepsilon \\
&\qquad\quad \textit{else } \varepsilon \\
h_i &\stackrel{\text{def}}{=} 0 \\
p_c &\stackrel{\text{def}}{=} \textit{if } c_o = \varepsilon \textit{ then } \varepsilon \textit{ else } \{c_o\}_K \\
c_i &\stackrel{\text{def}}{=} \textit{if } h_o = \mathsf{finished} \textit{ then } \mathsf{ok} \textit{ else } \varepsilon
\end{aligned}
$$

Secure Channel: refinement

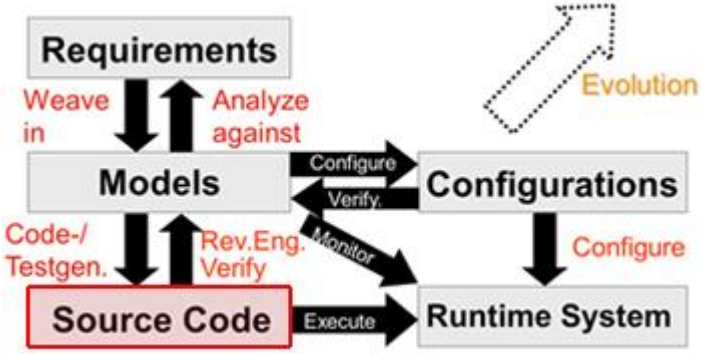
Have conditional interface refinement $P_c \overset{(D,U)}{\rightsquigarrow}_T P \otimes H$ where

$T = \{(\vec{s}, \vec{t}) : \forall n.(\forall i \leq n(\vec{s}(\tilde{c}_i)) \lfloor_i \neq finished \Rightarrow \forall i \leq n+1.(\vec{s}(\tilde{c}_o)) \lfloor_i = \varepsilon)\}$

$((\vec{s}, \vec{t}) \in \mathbf{Stream}_{O_{P_c}} \times \mathbf{Stream}_{I_{P_c}})$ and
U, D given by $I_D = \{\tilde{c}_o, \tilde{a}_c\}$, $O_D = \{c_o, a_c, a_h\}$, $I_U = \{c_i, p_c, h_a\}$ and $O_U = \{\tilde{c}_i, \tilde{p}_c\}$ and $c_o \overset{\text{def}}{=} \tilde{c}_o, a_c \overset{\text{def}}{=} \tilde{a}_c, a_h \overset{\text{def}}{=} \tilde{a}_c, \tilde{c}_i \overset{\text{def}}{=} c_i, \tilde{p}_c \overset{\text{def}}{=} h_a$ (after renaming channels of P_c to $\tilde{c}_o, \tilde{c}_i, \tilde{p}_c, \tilde{a}_c$).

Thus for $C(n)$ with $[\![C(n)]\!] \subseteq T$, have interface refinement $C(n) \otimes P_c \overset{(D,U)}{\rightsquigarrow} C(n) \otimes P \otimes H$. So for any $C(n)$ with $[\![C(n)]\!] \subseteq T$, $C(n) \otimes P \otimes H$ preserves secrecy of n.

Roadmap



1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

Security Analysis: Model or Code ?

How do I know a crypto-protocol implementation (as opposed to specification) is secure ?

Model:

+ earlier (less expensive to fix flaws)
+ more abstract ➔ more efficient
- more abstract ➔ may miss attacks
- programmers may introduce security flaws
- even code generators, if not formally verified

Code:

+ „the real thing“ (which is executed)

➔ Do both where feasible !

Solution

Possible solutions in the context of model-based security engineering :

1. Verify specification, write code generator, verify code generator. Problems:
   - very challenging to verify code generator
   - generated code satisfactory for given requirements (maintainability, performance, size, …) ?
   - not applicable to existing implementations
2. Generate models from code and verify these.
   - Advantages:
     -- Seems more automatic.
     -- Users in practice can work on familiar artifact (code),
     don t need to otherwise change development process (!).
   - Challenges: Currently possible for restricted code or using significant annotations. Need to verify model generator.
3. Create models and code manually and verify code against models. Advantages:
   - Split heavy verification burden (Model-level analysis more efficient).
   - Get some verification result already in design phase (for non-legacy implementations) => cheaper to fix.

Just an Exercise in Code Verification ?

State of the art in code verification in practice: execution exploration by *testing*. Limitations:

- For highly interactive systems usually only partial test coverage due to test-space explosion.
- Cryptography inherently un-testable since resilient to brute-force attack.

Interactive formal software verification (Isabelle et al): assumes specialist users.

Automated … (Bandera, Soot et al.): scalability wrt. code size / complexity; sophistication of properties (security).

NB: There are specialized approaches in development to formally verify crypto-protocol implementations independently from models (e.g. Aizatulin, Dupressoir, Gordon, Jürjens @ CSF‘11, CCS‘11, CCS‘12) => not in scope of this lecture.

Model-Code Traceability Mapping

„meaning“

„meaning“

compare meaning!

Backtrace assignments

Defined during model creation

Sent and received data

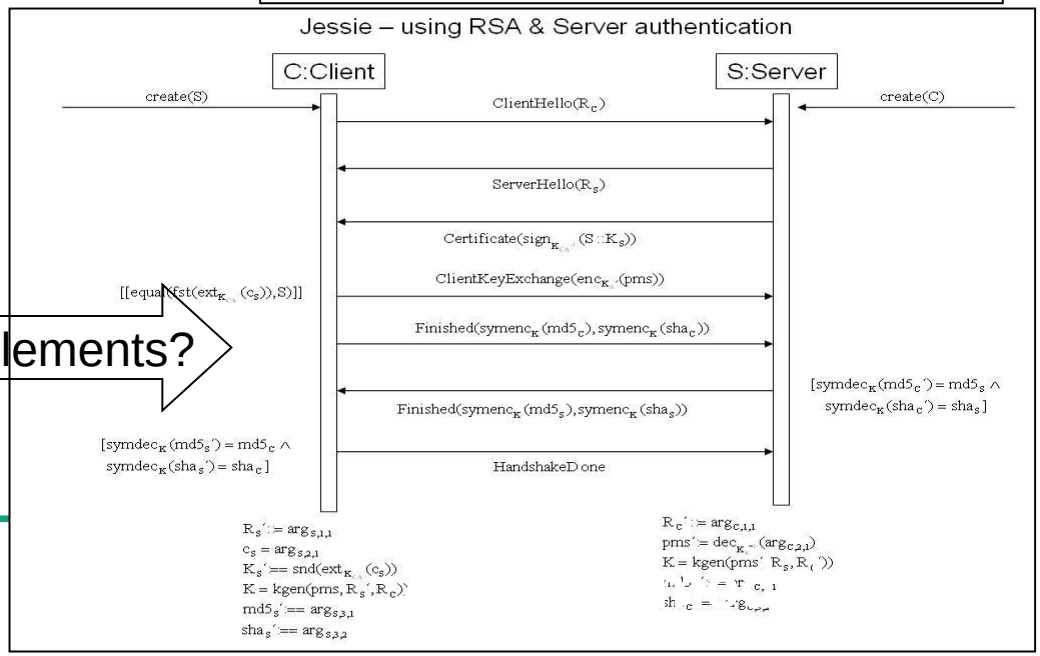
Elements of connections

Find

Implement -ation

.java

Implements?



technische universität dortmund

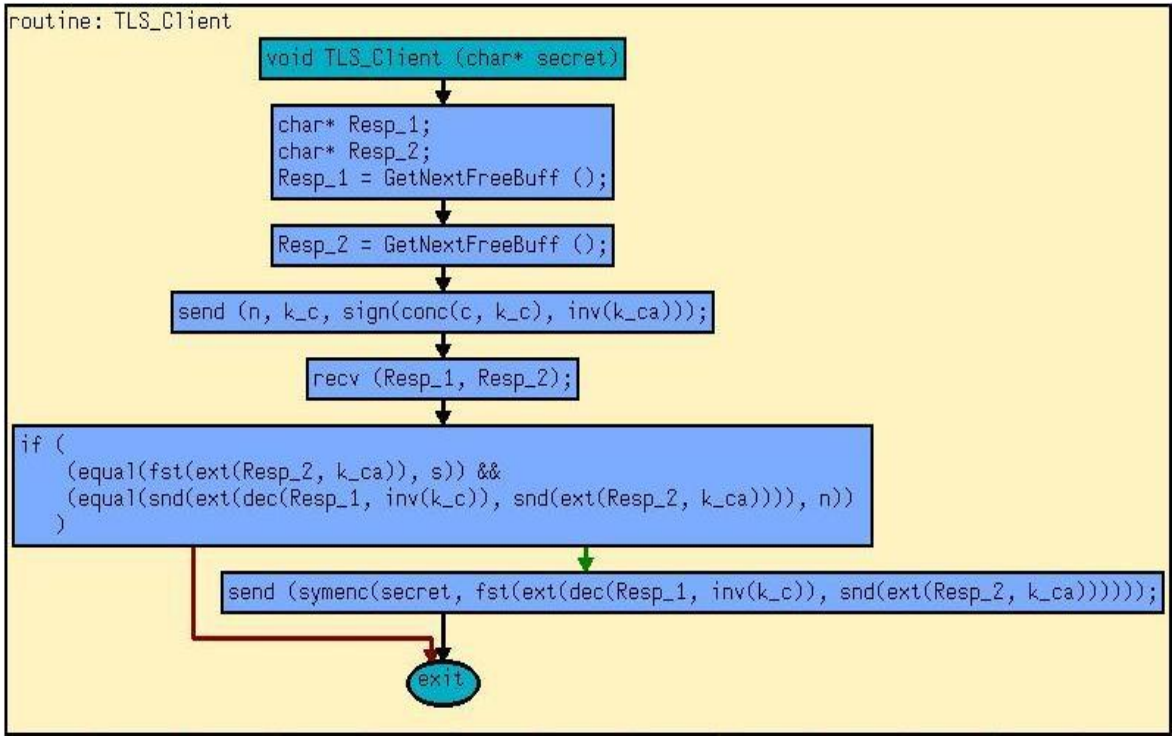
Models from Code

Generate control flow graph (e.g. aicall (Absint)).

Transform to state machine: [ASE05,ASE06]

trans(state,inpattern,
condition,action,
nextstate)

where action can be
outpattern or
localvar:=value.



Experiences

Can generate behavioral models from code (e.g. CFGs). Problem: too concrete

➔ understanding + automated verification hard (even with annotations).

Constructing abstract specifications from practical software is manually intensive.

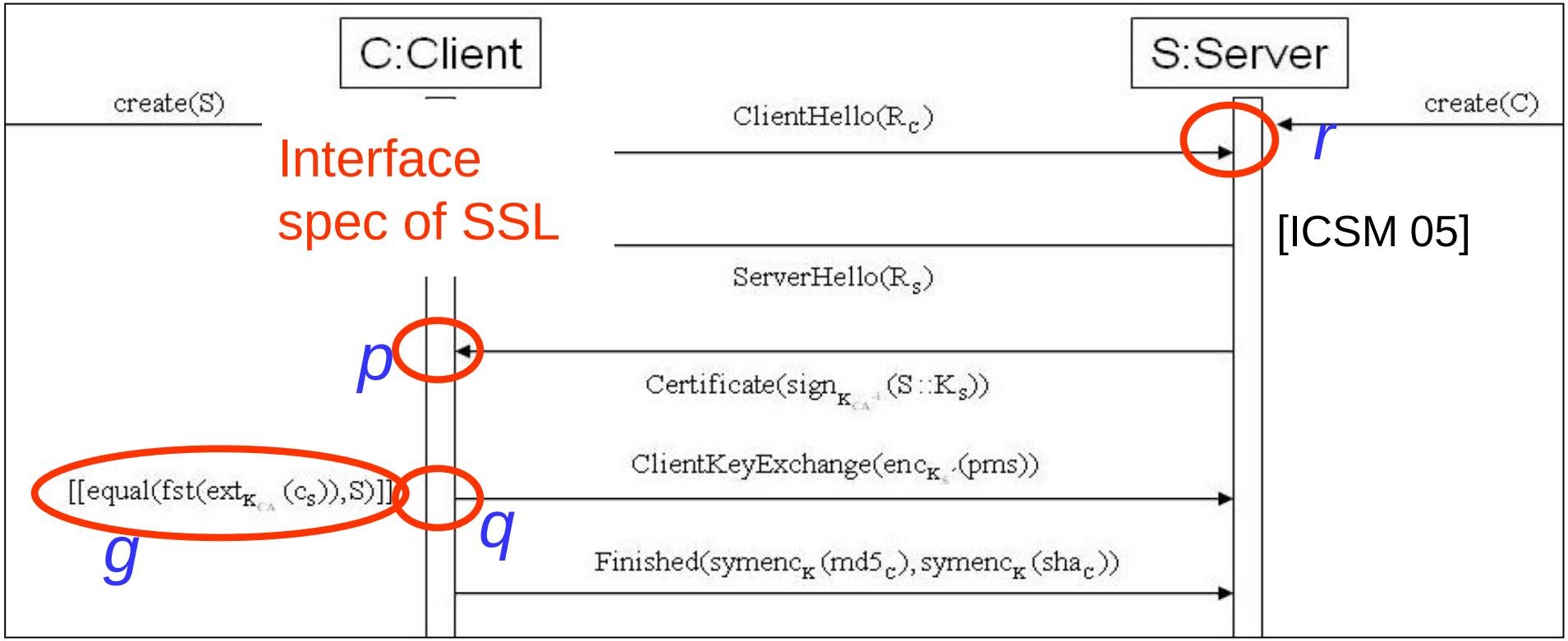
Verify Code against Models

Assumption: Have textual specification. Then:

- construct interface spec from textual spec
- analyze interface spec for security
- verify that software satisfies interface spec (using run-time verification)

JSSE / Jessie

- Java Secure Sockets Extension (JSSE) contains implementation of SSL.
- Open-source clean-room reimplementation Jessie.
- Applied our approach to fragment of Jessie (SSL handshake using RSA, verifying secrecy of exchanged secret).
- Currently extending the work to JSSE recently made open-source by Sun.



- Identify program points:
  value (*r*), receive (*p*), guard (*g*), send (*q*)

II. Check guards enforced

Parameter of the cryptographic Client-hello Message	Transferred data of the Client-Hello Message of the Jessie implementation
C	type.getValue()
Pver	major
	minor
	((gmtUnixTime >>>
	((gmtUnixTime >>> 16) & 0xFF)
	((gmtUnixTime >>> 8) & 0xFF)
	(gmtUnixTime & 0xFF)
r_c	randomBytes
	sessionId.length
Sid	sessionId
	((suites.size() << 1) >>> 8 & 0xFF)
	((suites.size() << 1) & 0xFF)
LCip	suites_1
	...
	suites_N
	comp.size()
LKomp	comp_1
	...
	comp_N

Implementation (Jessie): Identify Values

Currently do this manually using code assertions

public void write(OutputStream out) throws IOException
{ ... **out.write(randomBytes);** … }

Identify: randomBytes

(in message *ClientHello)*

2nd parameter of Random constructor called by ClientHello.write()

2nd parameter of ClientHello constructor

public void write(OutputStream out) throws IOException
{ ... **random.write(out);** ... }

ClientHello(… , Random random,)
{ ... **this.random = random;** ... }

via Handshake.write()
initialized in SSLSocket.doClientHandshake()

ClientHello clientHello = new ClientHello(...,**clientRandom**,...);

initialization of the used Random object

Random clientRandom =
new Random(...,**session.random.generateSeed(28)**);

„meaning“

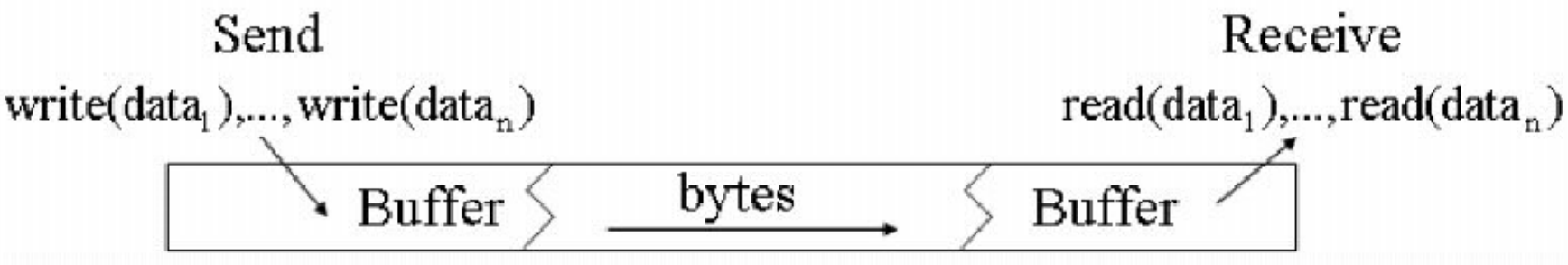
class SecureRandom (specified in: FIPS 140-2,RFC 1750) of package java.security
Function: generateSeed

technische universität dortmund

Input / Output

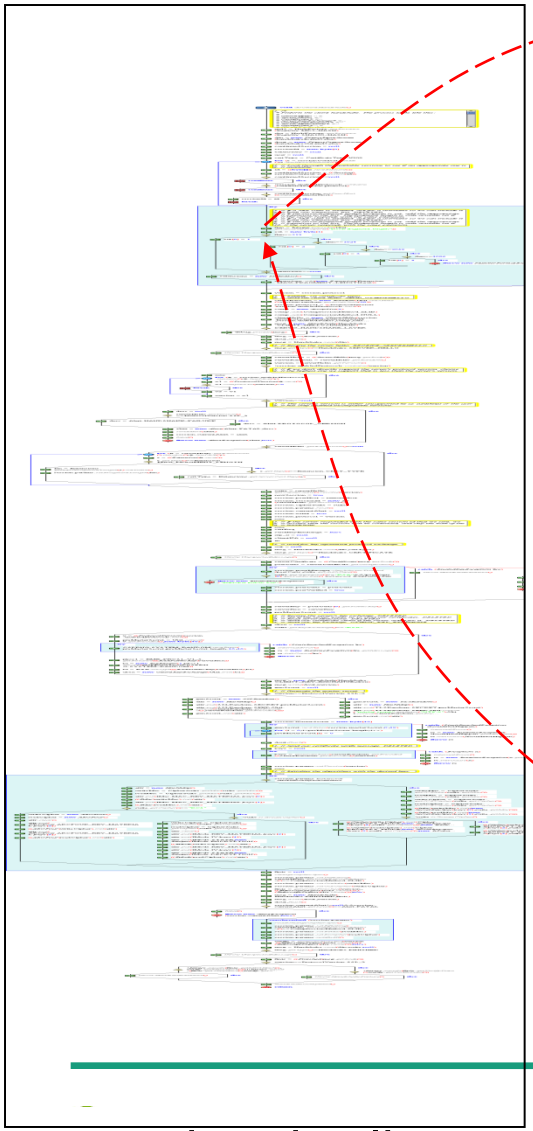
To extract input/output labels for state machine transitions, analyze input / output mechanism used in the implementation.

Many implementations (e.g. Jessie and JSSE) use buffered communication where the message objects implement read and write methods. Translate these method calls to input / output labels (need to track successive subcalls).

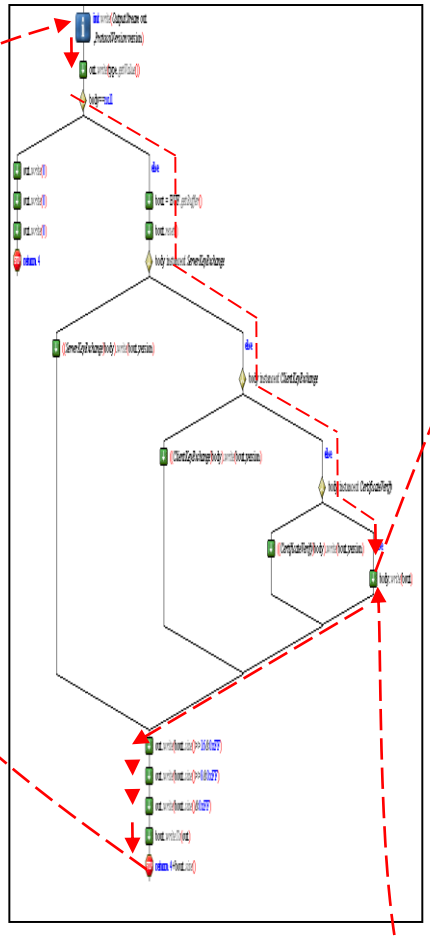


Sending Messages

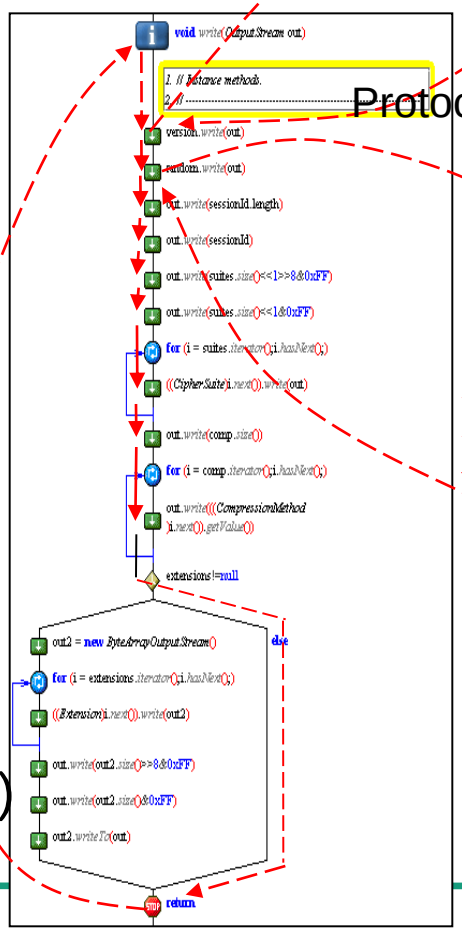
Automate this using patterns



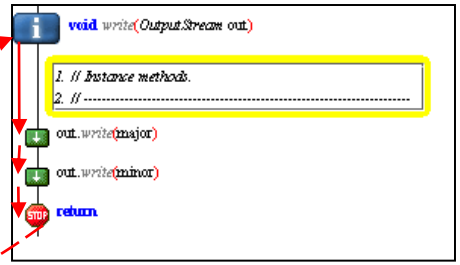
SSLSocket.doClientHandshake()



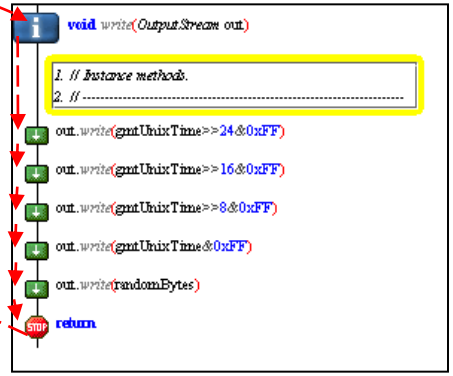
Handshake.write()



ClientHello.write()



ProtocolVersion.write()

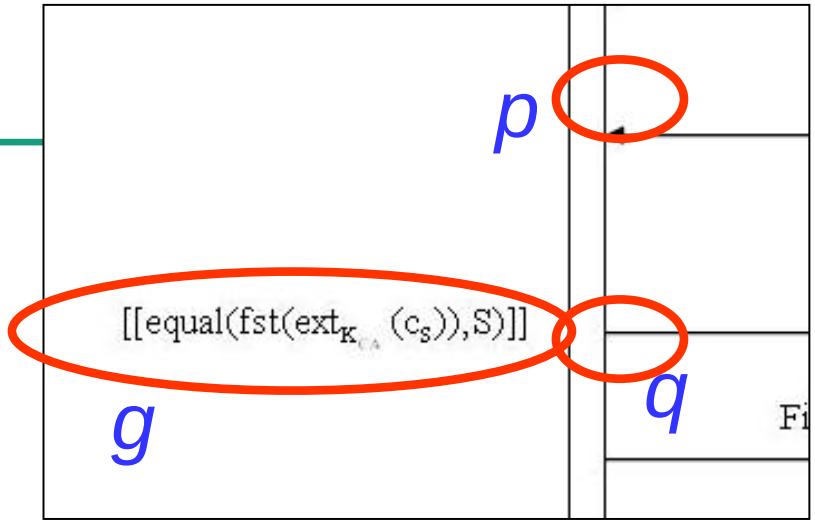


Random.write()

traverse CFG

call of OutputStream.write()

Checking Guards



Guard *g* enforced by code?

- Generate runtime check for *g* at *q* from diagram: simple + effective, but performance penalty.
- Testing against checks (symbolic crypto for inequalities).
- Automated formal local verification: conditionals between *p* and *q* logically imply *g* (using ATP for FOL).

```
public void checkServerTrusted(X509Certificate[] chain, String authType)
  throws CertificateException  {…      checkTrusted(chain, authType);    }
```

calls checkTrusted()

Guard:
checkServerTrusted()

```
private void checkTrusted(X509Certificate[] chain,
String authType) throws CertificateException
   {      ...    }
```

calls verify() for every member of certificate chain

„meaning“

```
public void verify(PublicKey key, String provider)
throws CertificateException, ...
   {      ...    }
```

calls doVerify()

java.security.Signature

- **Initializatize**
- **Update**
- **Verify**

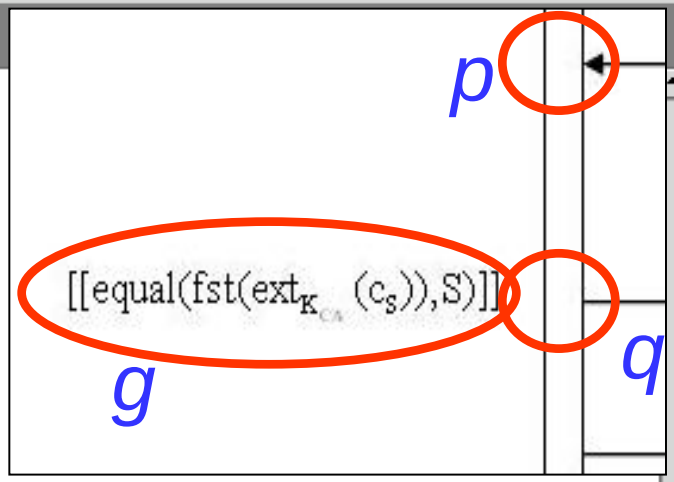
„verifies the signature“

```
private void doVerify(Signature sig,PublicKey key)
   throws CertificateException, ...
  {   ...    sig.initVerify(key);
   sig.update(tbsCertBytes);
   if (!sig.verify(signature))
  {… throw new CertificateException
       ("signature not validated"); …   }  }
```

technische universität dortmund

Execution Path Diagram for method SSLSocket.doClientHandshake()

msg = Handshake.read(din, certType);



p

catch

try

only possible way without throwing exception

g

session.trustManager.checkServerTrusted (peerCerts,suite.getAuthType());

serverKex = se

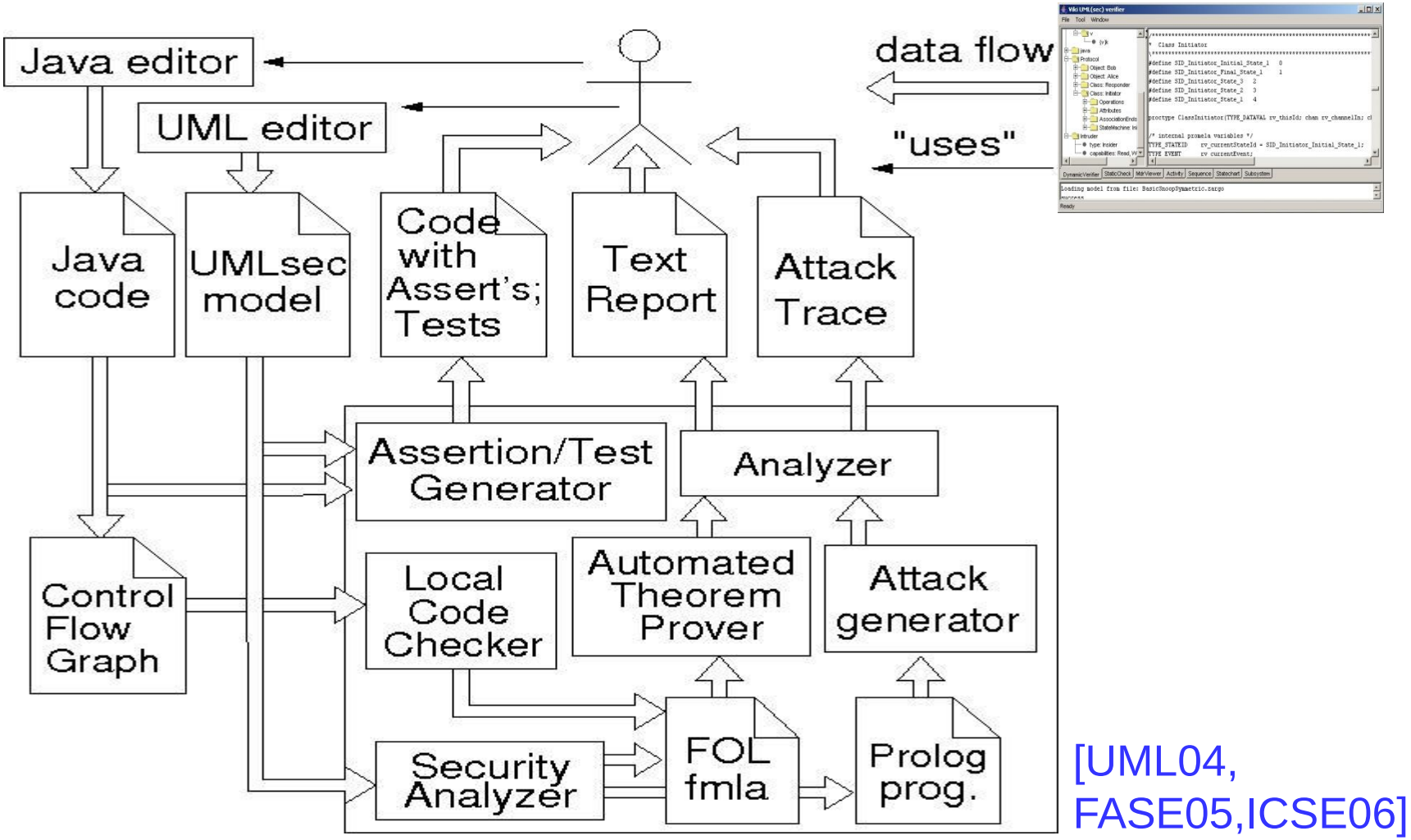
q

msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);
msg.write (dout, version);

technische u
dortmund

Class Sequence Flow

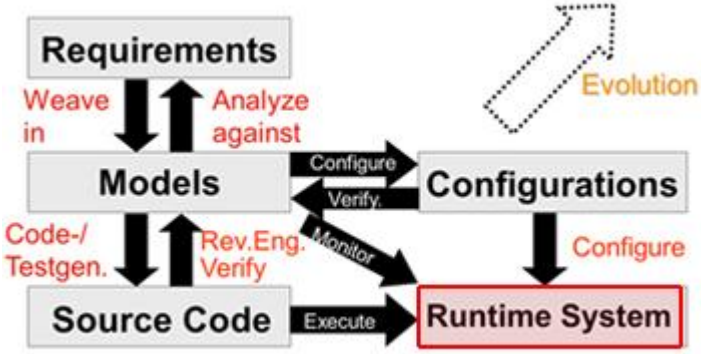
Tool Support



[UML04, FASE05,ICSE06]

Roadmap

1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications



Model-Runtime Traceability

How do I know the running implementation is still secure after deployment ?

- Does system model capture all relevant aspects about a system ?
- Are assumptions about influences from a system's operational environment reflected adequately ?
- Are the abstractions that need to be made to enable automated static verification of non-trivial systems faithful wrt the verification result ?

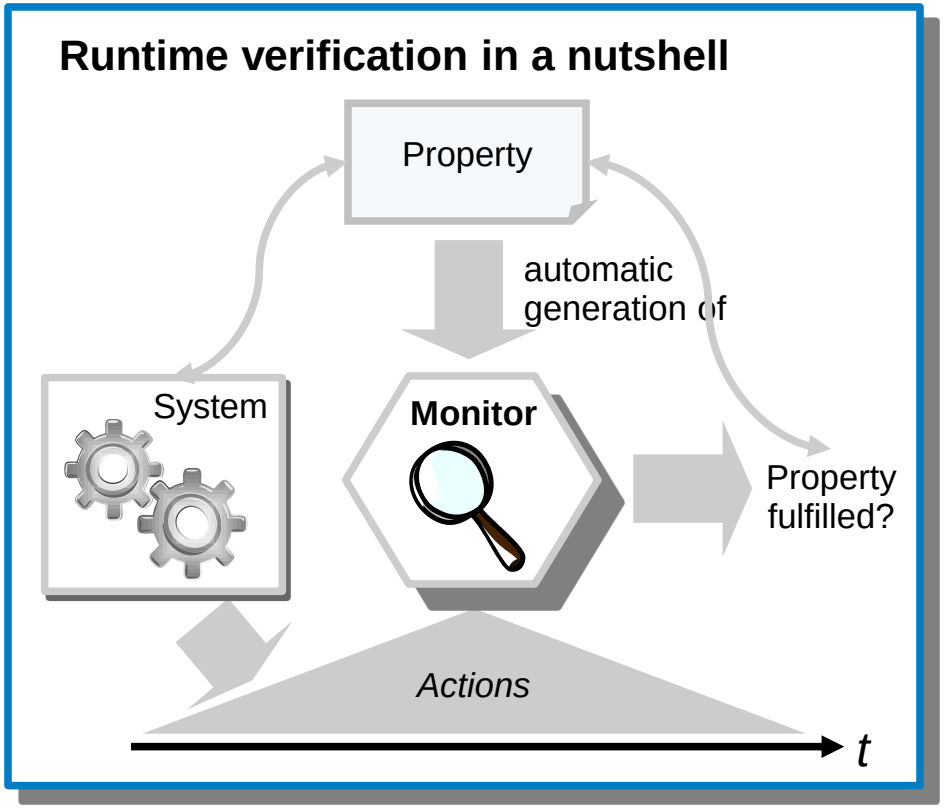
➔ Run-time verification.

Runtime Verification using Monitors

Dynamic verification technique on the actual system.

Essentially a symbiosis of model-checking and testing.

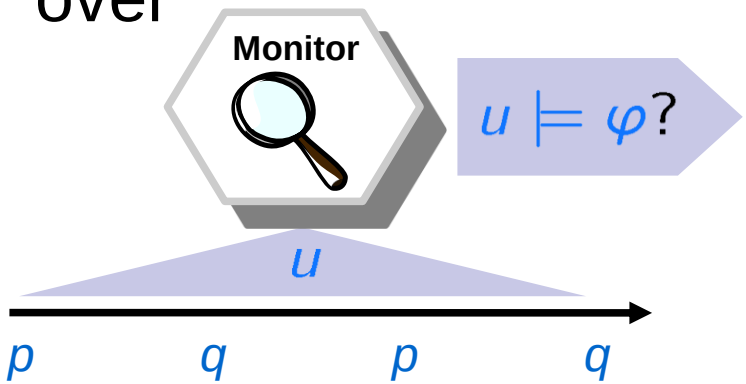
"Lazy model-checking": only check the system traces which are executed, when they are executed.



Formal underpinnings

- System (safety) property, φ , specified in terms of linear time temporal logic [Pnu77]:
  $\varphi ::= true \mid p \mid \neg p \mid \varphi \; op \; \varphi \mid \varphi \mathbf{U} \varphi \mid \mathbf{X}\varphi \quad (p \in AP)$

- Continuous interpretation of φ over sequence of system events (behaviours), $u \in (2^{AP})^*$



- ***Automatic monitor generation:*** *"Inspired" by translation of LTL to Büchi-automata*

$$\varphi \rightarrow BA_\varphi \text{ s.t. } L(BA_\varphi) = L(\varphi)$$

Semantics

$$
\begin{array}{lcl}
w, i \models true & & \\
w, i \models \neg\varphi & \Leftrightarrow & w, i \not\models \varphi \\
w, i \models p \in AP & \Leftrightarrow & p \in w(i) \\
w, i \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & w, i \models \varphi_1 \vee w, i \models \varphi_2 \\
w, i \models \varphi_1 \boldsymbol{U} \varphi_2 & \Leftrightarrow & \exists k \geq i.\ w, k \models \varphi_2 \wedge \\
 & & \forall i \leq l < k.\ w, l \models \varphi_1 \\
w, i \models \boldsymbol{X}\varphi & \Leftrightarrow & w, i+1 \models \varphi
\end{array}
$$

We write $w \models \varphi$, if and only if $w, 0 \models \varphi$, and use $w(i)$ to denote the ith element in w.

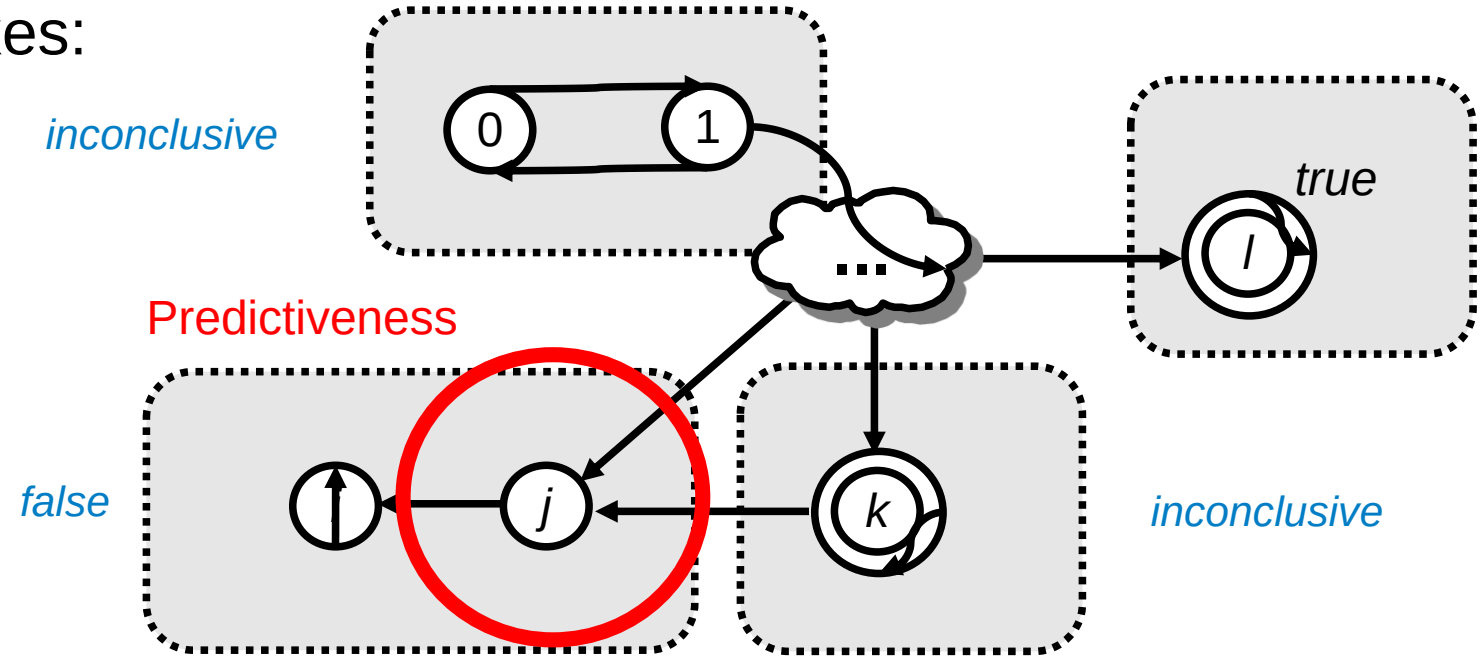
Write F phi for true U phi (“eventually phi”); G phi for not F not phi (“globally phi”); phi1 W phi2 for G phi1 or (phi1 U phi2) (weak-until)

Monitoring-friendly LTL semantics

3-valued semantics:

$$[u \models \varphi] = \begin{cases} \top & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \text{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \text{otherwise} \end{cases}$$

Gives finite-state machines for detecting *minimal* bad prefixes:



ClientKeyExchange

Client will not send out ClientKeyExchange message until has received Certificate message and check is positive, and then sends it out.

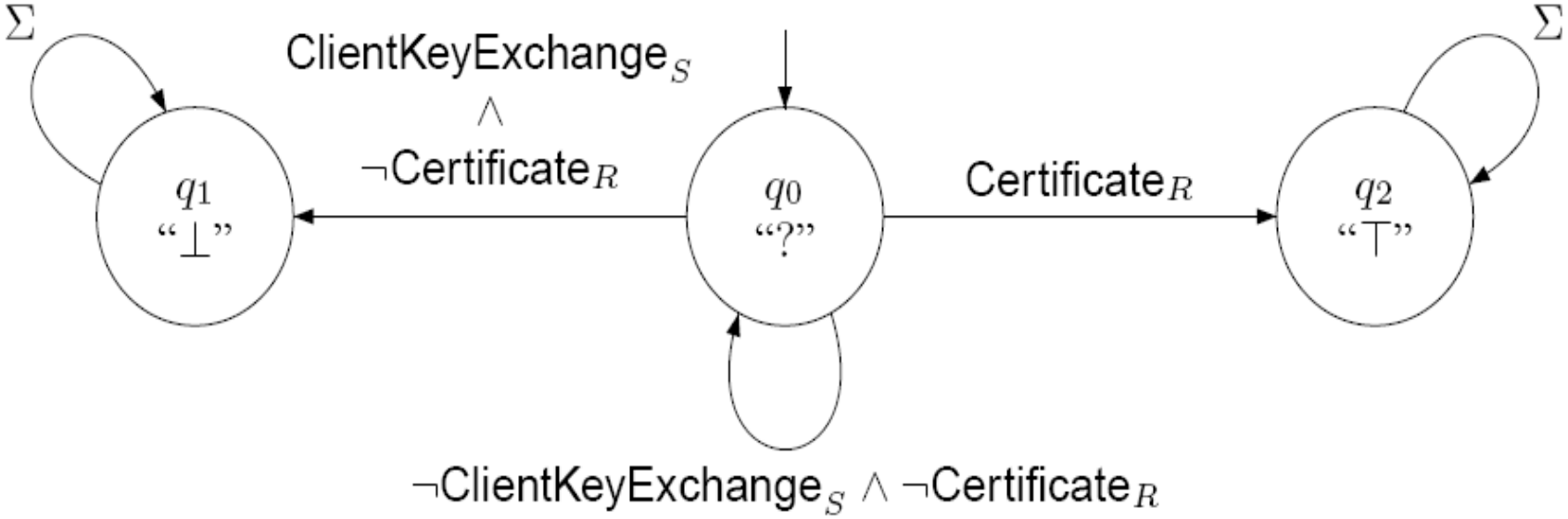


Figure 1: FSM ¬ClientKeyExchange$_S$ U Certificate$_R$.

Client Transport Data

Client will not send any transport data before has checked that MD5 hash received in Server`s Finished message is equal to MD5 created by Client (and correspondingly for SHA hash).

$$\varphi_3 = \neg Data\mathbf{W}((MD5(\mathsf{Finished}_R) = MD5(\mathsf{Finished}_S)),$$

not co-safety but safety

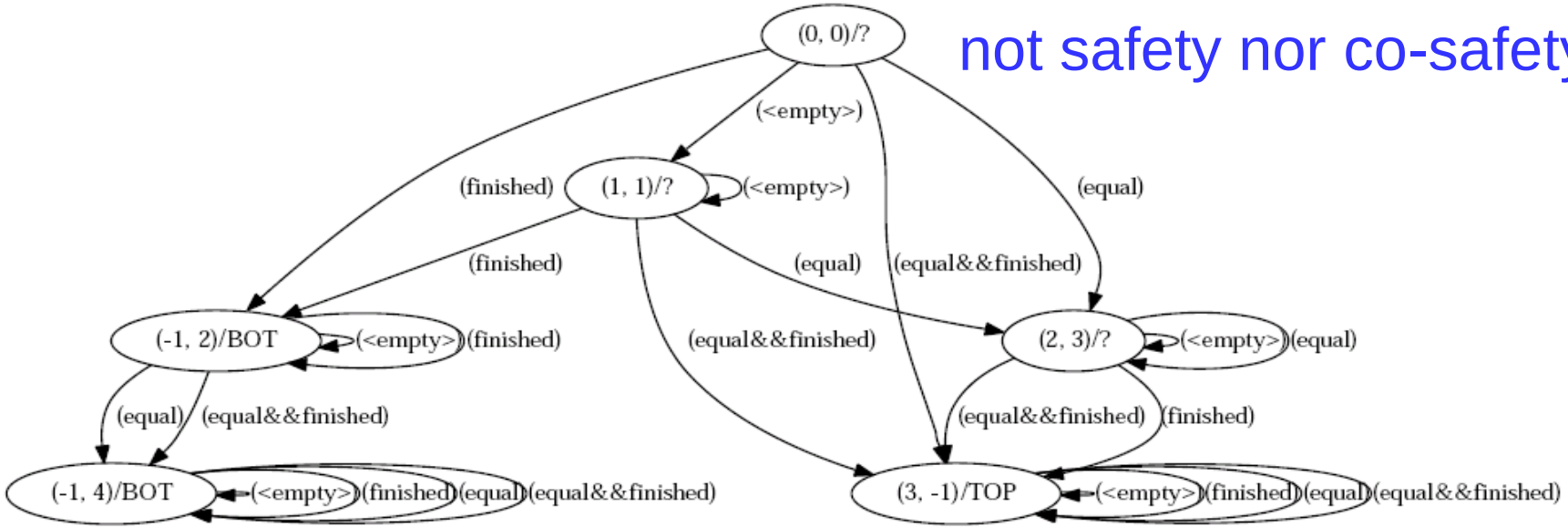
Server Finished

Server will not send Finished message before MD5 received in Client`s Finished message equal to MD5 created by server. Then sends out eventually.

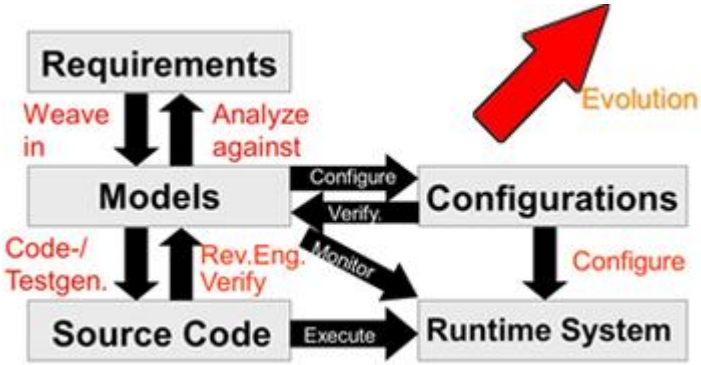
NB: Improves on Schneider's security automata.

$$\varphi_2 = (\neg \text{finished } \mathbf{W} \text{ equal} \wedge (\mathbf{F} \text{ equal} \Rightarrow \mathbf{F} \text{ finished}))$$

not safety nor co-safety

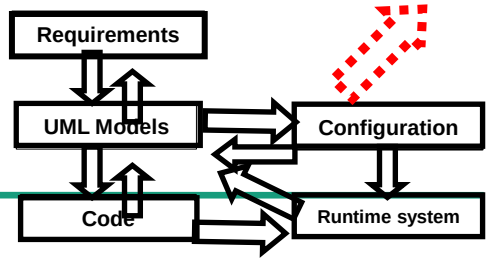


Roadmap



1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

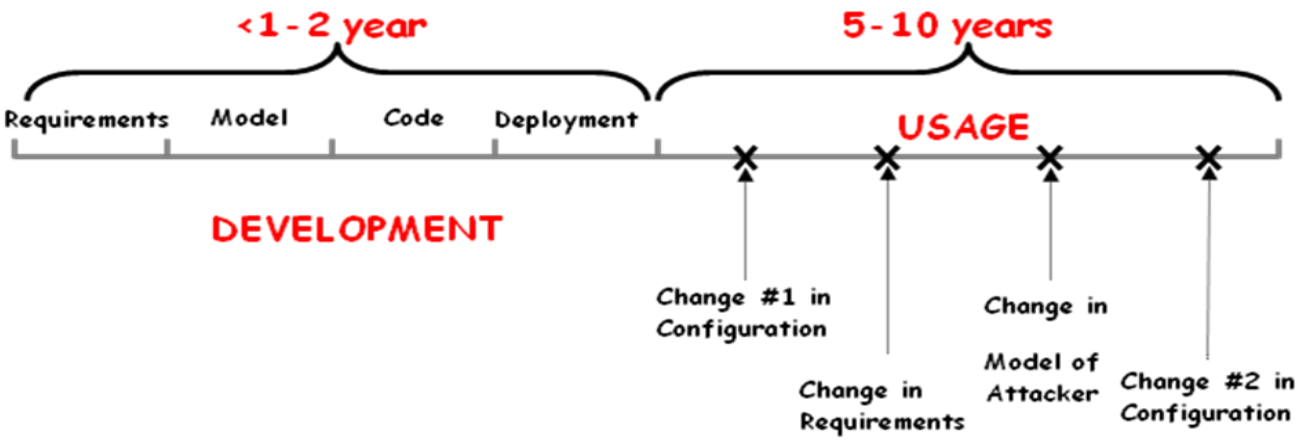
The Forgotten End of the System Life-cycle



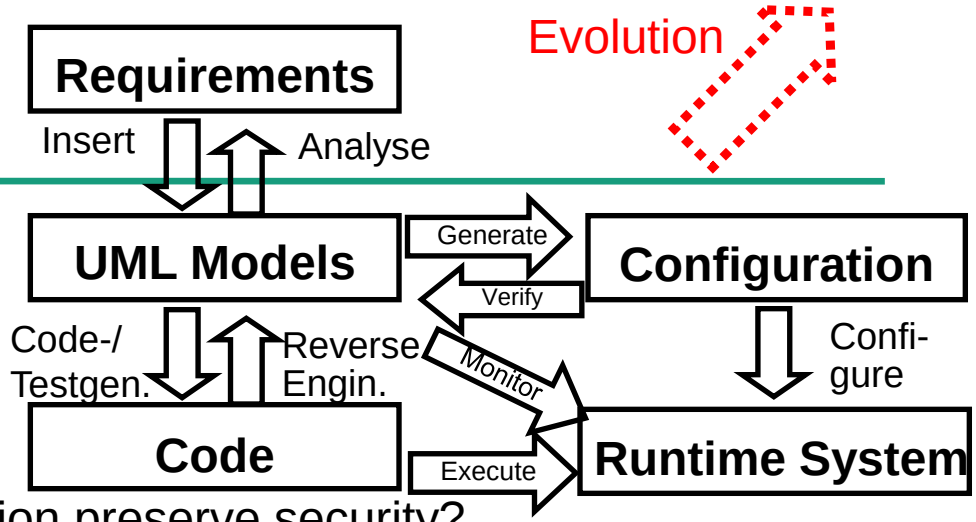
Challenges:

- Software lifetime often longer than intended (cf. Year-2000-Bug).
- Systems evolve during their lifetime.
- In practice evolution is difficult to handle [cf. HVB example].

Problem: Critical requirements (e.g. security) preserved ?



Challenge: Evolution



Each artifact may evolve.

To reduce costs, reuse verification results as far as possible.

=> Under which conditions does evolution preserve security?

Even better: examine possible future evolution for effects on security.

- Check *beforehand* whether potential evolution will preserve security.
- Choose an architecture during the design phase which will support future evolution best wrt. security.

=> Evolution as first-class modeling concept in UMLsec.

Trade-off: flexibility of evolution vs. preservation of security.

[NB: analogous problem: Software-product lines.]

Types of Change

Change can arise from

- Need to update, tailor, delete or introduce new security feature(s) (security-driven changes)
- System evolution
  - Updates, modifications, deletions and additions to existing functional features
  - Introduction of new functional features
- Changes in environment, attacker models etc.

Change may arise in any of the life-cycle phases

UMLsec can be used to model security relevant changes in all life-cycle phases

Particular interesting instance: the evolution mechanism itself (e.g. Software versioning, software update (cf GP)) – is it itself functioning securely ?

Practical example: different SSL versions in HVB architecture.

How to deal with change?

All artefacts on our overview picture can change.

Want to reuse assurance results as much as possible.

=> 1) Preservation of security under change ?

Need mapping between old and new artefact for this.

Also: Want to take into consideration possible future changes at development stage. Certain security analyses before allowing the future change to happen:

- ✓ **Existing security property destroyed due to the change?**
- ✓ **Other security-related artefacts affected by the change?**

=> 2) Change as first order citizen in modelling and analysis. Again need 1) for that (and also provides above mapping).

Changing the Model vs. Modeling Changes

What If…

- You want to change all things of a type?

  Tedious search for every instance

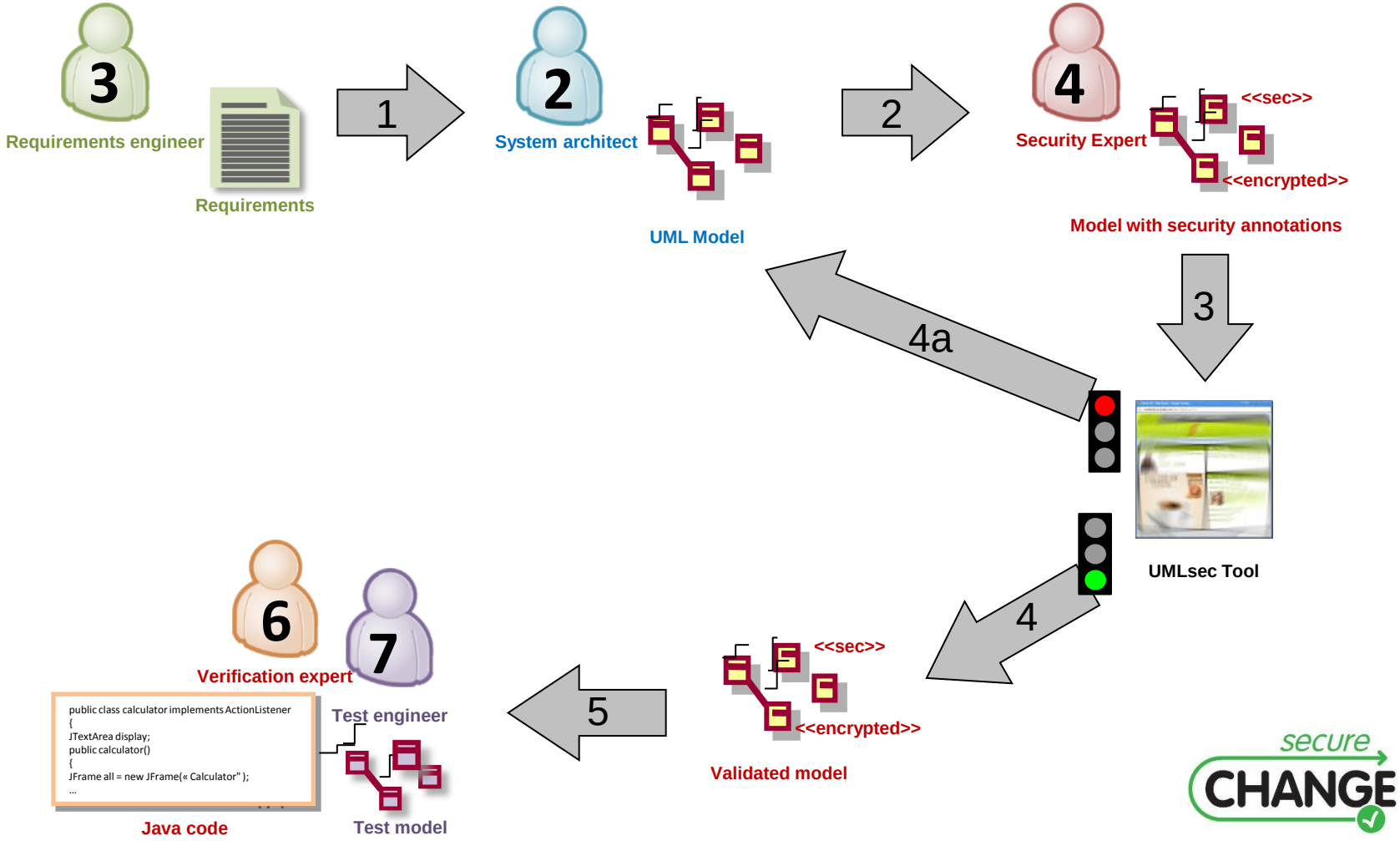
- You want to describe different variants?

  Multiply the model files

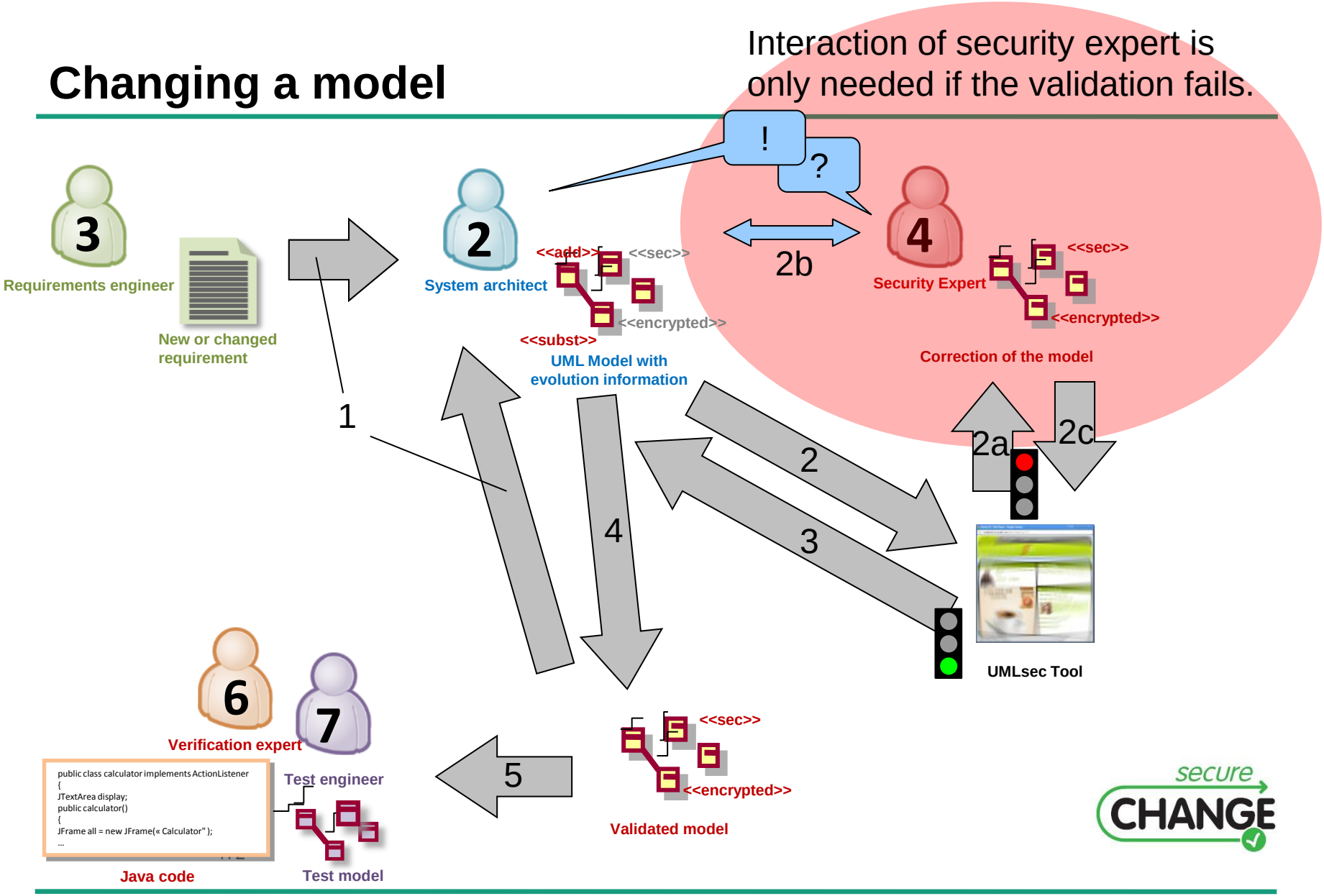
- Very little has actually changed, but the model is huge?

  Complete analysis takes time

Initial creation of a model



Changing a model



Change of specification models

- small-step (atomic) changes: identified atomic changes that span the complete space of possible changes, determine preconditions under which security properties are preserved
- big-step (refactoring) changes: consider several generally applicable big-steps changes

Atomic Change: Deletion

p>p': p leaks more know-ledge than p'

p preserves secrecy of s assuming C:

$\psi(P) \wedge C \not\vdash \mathsf{knows}(s)$

Theorem 1 *Assume that the program p' evolved from the program p where p and p' are related as in the following case distinctions. Assume we are given the public knowledge $\mathcal{P}$ and the previous adversary knowledge $\mathcal{A}$.*

$p = E$, $p' = \varepsilon$ *(where $E \in$ **Exp** and ε is the empty program): This implies $p \succcurlyeq p'$.*

$p =$ *either p' or p'': This implies $p \succcurlyeq p'$ and $p \succcurlyeq p''$.*

$p =$ *if $E = E'$ then p' else p'': For any expression $X \in$ **Exp** such that p preserves the secrecy of X:*
p' preserves the secrecy of X assuming $E = E'$ and
p'' preserves the secrecy of X assuming $E \neq E'$.

$p =$ *case E of key do p' else p'': For any expression $X \in$ **Exp** such that p preserves the secrecy of X:*
*p' preserves the secrecy of X assuming $E \in$ **Keys** and*
*p'' preserves the secrecy of X assuming $E \notin$ **Keys**.*

$p =$ *case E of $x :: y$ do p' else p'': For any expression $X \in$ **Exp** such that p preserves the secrecy of X:*
p' preserves the secrecy of X assuming $\exists x, y. E = x :: y$ and p'' preserves the secrecy of X assuming $\neg\exists x, y. E = x :: y$.

Atomic Change: Insertion

Theorem 2 *Assume that the program p evolved from the program p' where p and p' are related as in the following case distinctions.*

$p = E$, $p' = \varepsilon$ *(where $E \in$ **Exp** and ε is the empty program): For any expression X, we have* $\psi(p) \vdash \mathsf{knows}(X)$ *iff* $\psi(p') \wedge [\forall i_1, \ldots, i_n.[\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \mathsf{knows}(E(i_1, \ldots, i_n))]] \vdash \mathsf{knows}(X)$.

$p =$ *either p' or p'': For any expression X, we have* $\psi(p) \vdash \mathsf{knows}(X)$ *iff* $\psi(p') \wedge \psi(p'') \vdash \mathsf{knows}(X)$.

$p =$ *if $E = E'$ then p' else p'': For any expression X, we have* $\psi(p) \vdash \mathsf{knows}(X)$ *iff*

$$[\forall i_1, \ldots, i_n.[\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow (E(i_1, \ldots, i_n) = E'(i_1, \ldots, i_n) \Rightarrow \phi(p')) \wedge (E(i_1, \ldots, i_n) \neq E'(i_1, \ldots, i_n) \Rightarrow \phi(p'))]] \vdash \mathsf{knows}(X).$$

$p =$ *case E of key do p' else p'': For any expression X, we have* $\psi(p) \vdash \mathsf{knows}(X)$ *iff*

$$[\forall i_1, \ldots, i_n.[\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow (\mathsf{key}(E(i_1, \ldots, i_n)) \Rightarrow \phi(p)) \wedge (\neg\mathsf{key}(E(i_1, \ldots, i_n)) \Rightarrow \phi(p'))]] \vdash \mathsf{knows}(X).$$

$p =$ *case E of $x :: y$ do p' else p'': For any expression X, we have* $\psi(p) \vdash \mathsf{knows}(X)$ *iff*

$$[\forall i_1, \ldots, i_n.[\mathsf{knows}(i_1) \wedge \ldots \wedge \mathsf{knows}(i_n) \Rightarrow \forall h, t.(E(i_1, \ldots, i_n) = h :: t \Rightarrow \phi(p[h/x, t/y])) \wedge (\neg\exists h, t.E(i_1, \ldots, i_n) = h :: t \Rightarrow \phi(p'))]] \vdash \mathsf{knows}(X).$$

Big-step Change: Refinement

One way to explicitly specify possibilities for future change is by underspecification: Change by one instantiation to the other.

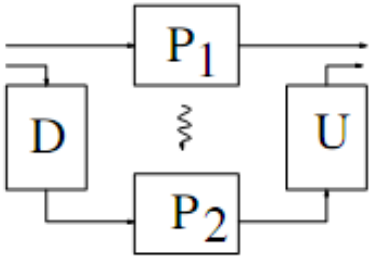
Ideally refinement should preserve security: Verify abstract spec, extends to instantiations.

Unfortunately doesn't work with all approaches in the literature.

Our notion of model refinement preserves security requirements (cf. earlier theorem).

Other big-Step Changes

- Restriction of interface
- Refactoring using additional components
- Change up to set of input/output behaviours



In each case: Theorem which identifies pre-conditions on when this preserves security (cf. earlier).

Modular Change via Rely-Guarantee

Can permit future changes in a modular way by formulating rely-guarantee conditions on components and showing that these imply the security requirements.

Then components can be changed arbitrarily as long as the changed component still satisfies the rely-guarantee condition.

Have general results showing when this is admissible.

Generating Rely-Guarantee Conditions

For program fragment *p*, generate set of statements *derive(L,C,E)* such that adversary knowledge is contained in every set *K* such that:

- for every list *l* of values for the variables in *L* that satisfy the conditions in *C*:
  K contains the value constructed by instantiating the variables in the expression *E* with the values from *l*

Can then change components as long as still satisfy these rely-guarantee conditions.

Change of the Attacker Model

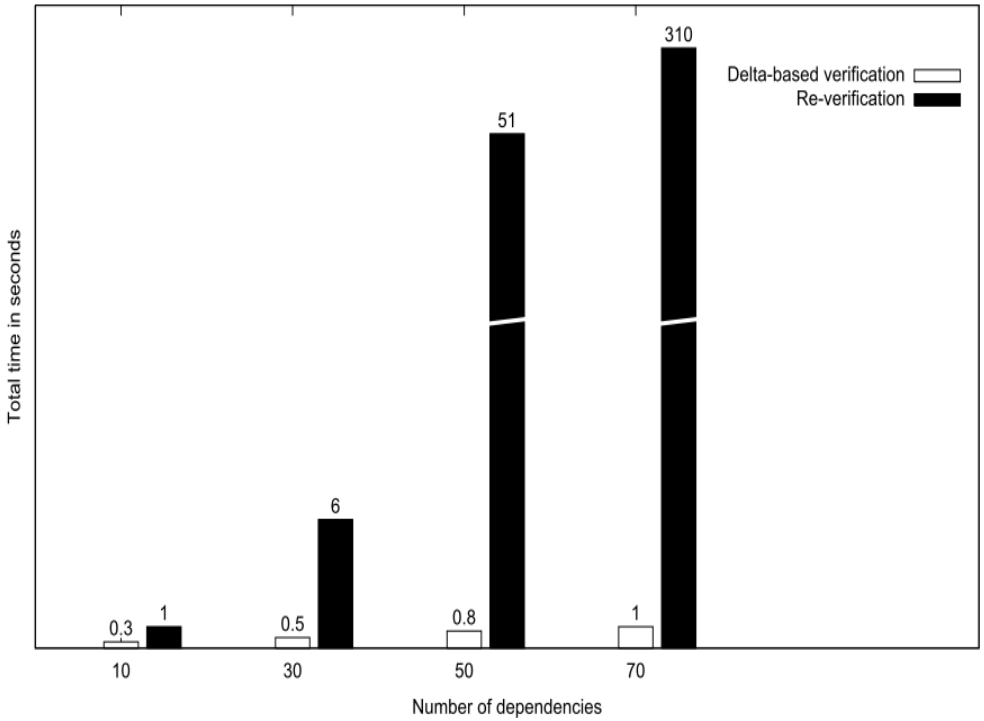
Change of attacker / threat model results in changes in the formula on the previous slide.

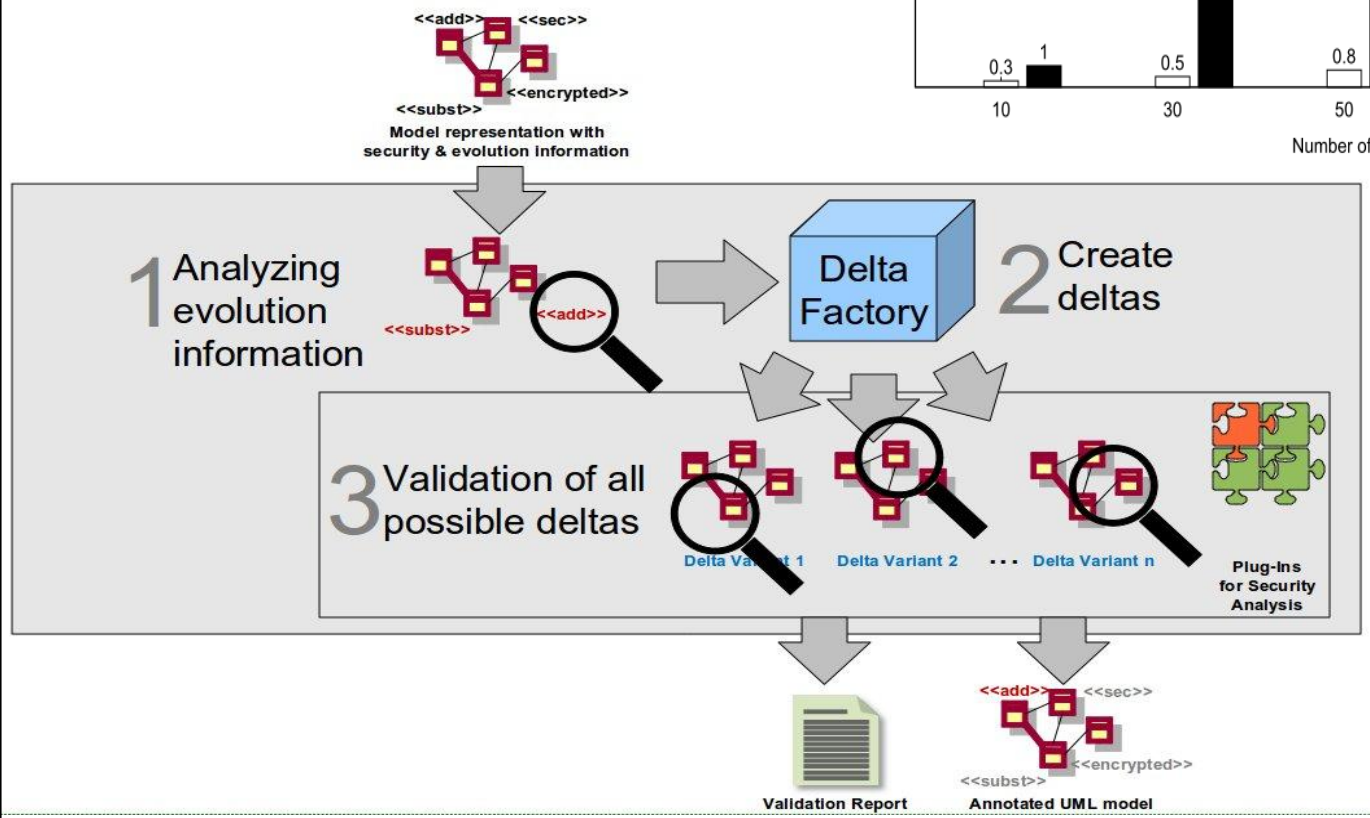
Have different attacker models (default, internal, ...).

Investigate to which extent security verification results can be reused between different attacker models.

Tool Support: System-Evolution

- Implemented evolution-based analysis within tool-support.
- Resulting performance gain





Jan Jürjens. 2011. Automated security hardening for evolving UML models. 33rd Int. Conf. on Software Engineering (ICSE '11). ACM.

A. Bauer, J. Jürjens, Y. Yu: Run-Time Security Traceability for Evolving Systems. Computer Journal 54(1): 58-87 (2011)

Fraunhofer ISST

Maintaining Model-Code Traceability under Evolution

Code evolution can potentially multiply testing effort.

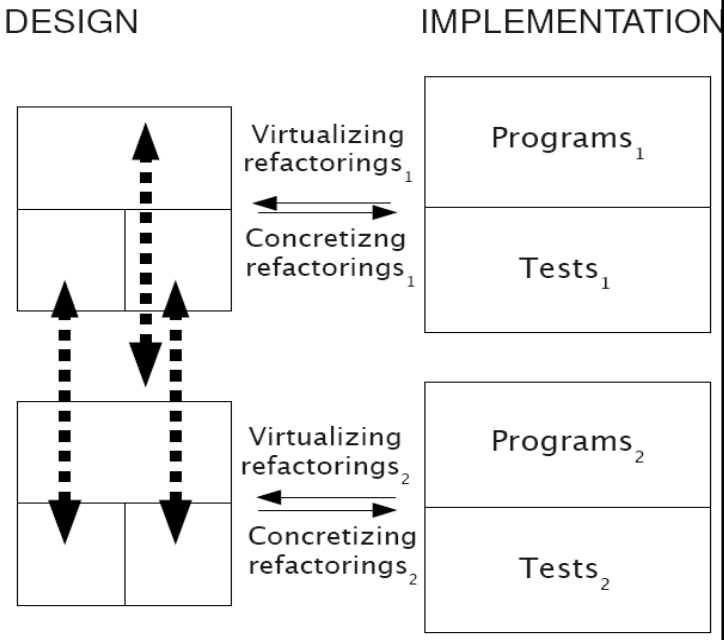
Need to re-verify code parts that changed, re-do integration testing etc.

Particular problem when testing for sophisticated properties (such as security) since requires particular effort.

➔ Want to automatically trace code evolution to model level so can automatically reuse earlier tests.

[Bauer, Jurjens, Yu 09]

Model-Code Co-Evolution



Basic observation: Most system changes can be reduced to two kinds:

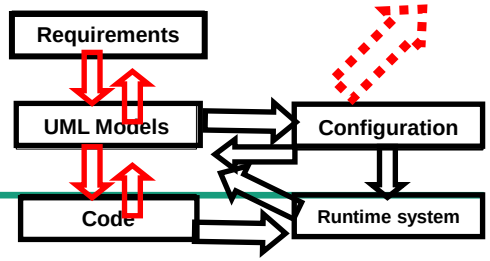
- Adding / removing parts of the system.
- Basic refactoring operations to hold system parts together despite changes.

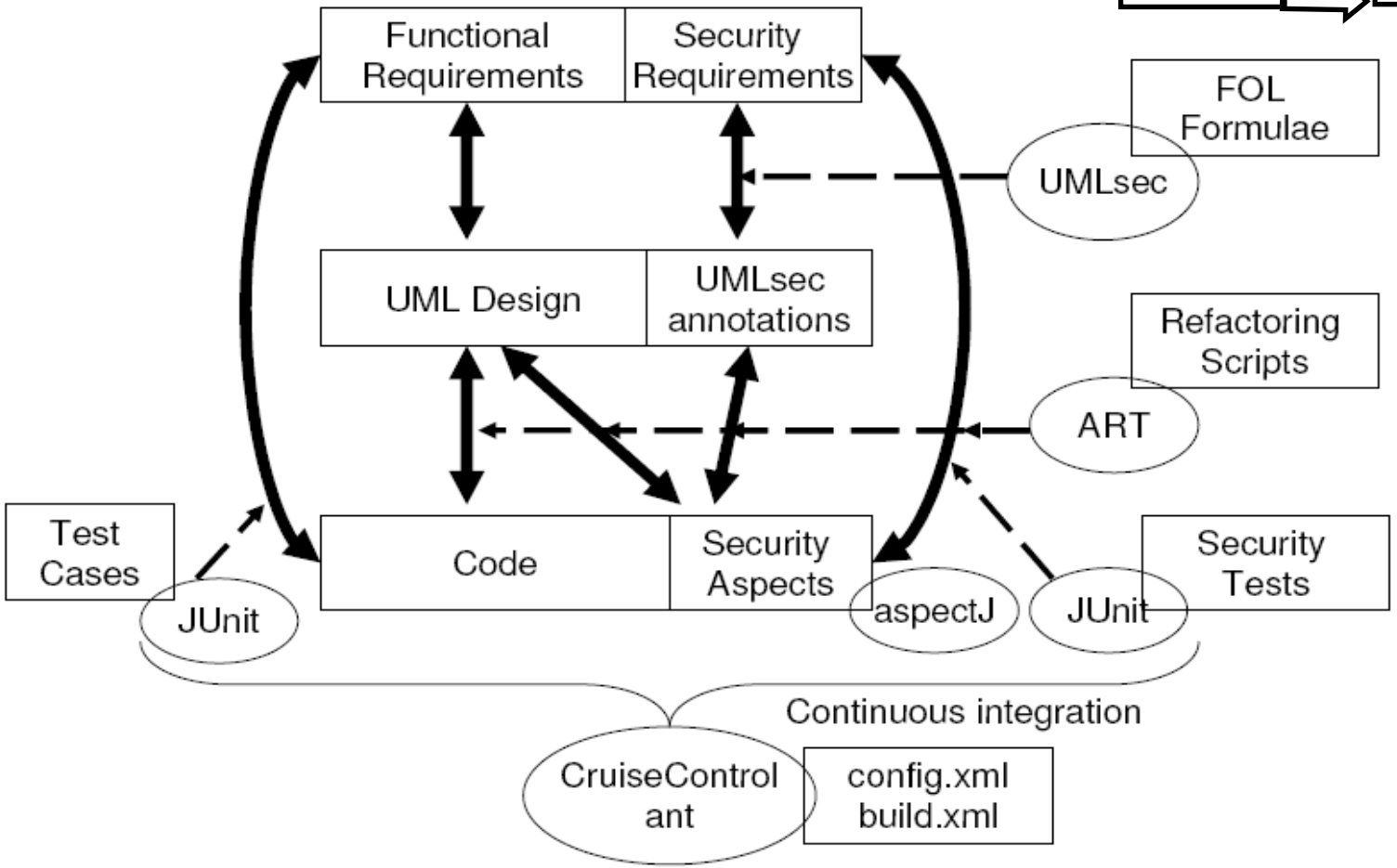
When adding / removing code parts we need to assume that the corresponding models are also added / removed.

For evolution by refactoring can achieve automated model-code traceability e.g. using Eclipse Refactoring Language Toolkit (LTK) / XML based refactoring scripts.

Maintain model-code synchrony using continuous integration scripts (with CruiseControl / Apache Ant).

Secure Evolution: Tool support





[ASE'07, ICSM'08, ASE'08, Computer Journal '10]

Preserving requirements-code traceability by refactoring.

Java Secure Sockets Extension

Applied our approach to a series of implementations of the Java Secure Sockets Extension library:

Jessie 1.0.0 - Jessie 1.0.1 - JSSE 1.6

Demonstrated that our model-code co-evolution approach is robust even across major software changes.

Symbols	Program entities	Identif.	Refactoring op.
1. C	clientHello	C	rename.type
2. S	serverHello	S	rename.type
3. P_{ver}	session.protocol version	P_ver	extract.temp
4. R_C	clientRandom	R_C	rename.local.variable
R_S	serverRandom	R_S	rename.local.variable
5. S_{id}	sessionId	S_id	rename.field
	sessionId	S_id	rename.local.variable
6. Ciph[ ]	session.enabledSuites	Ciph	extract.temp
7. Comp[ ]	comp	Comp	extract.temp
8. Veri	Lines 1518–1557	Veri	extract.method

Messages in sequence	op.	diff	Time (sec)
S1: $C \rightarrow S : (P_{\text{ver}}, R_C, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$	7	31	13.891
S2. $S \rightarrow C : (P_{\text{ver}}, R_S, S_{\text{id}}, \text{Ciph}[\,], \text{Comp}[\,])$	5	20	9.437
S3. $S \rightarrow C : \text{Certificate}[\text{X509Cert}_s]$	2	2	1.474
S4. $C : \text{Veri}(\text{X509Cert}_s)$	2	2	3.854
...	...	...	...
Total of 7 messages and 3 checks	27	86	40.303

Roadmap

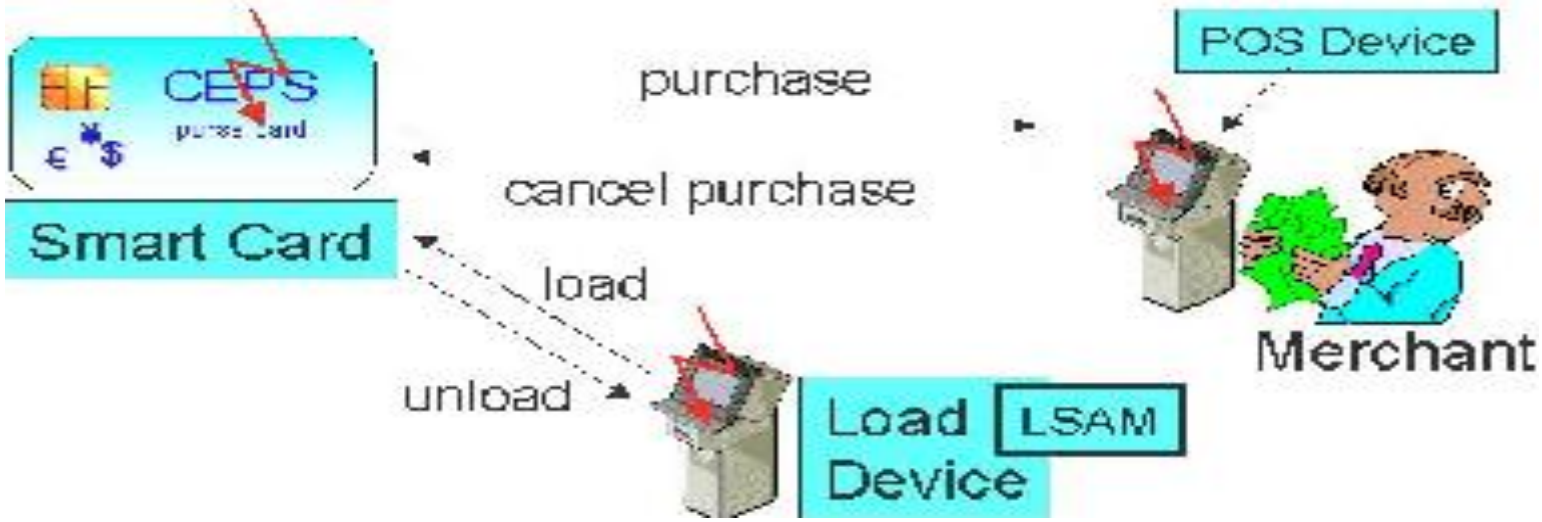
1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

Common Electronic Purse Specifications

Global electronic purse standard (90% of market).

Smart card contains account balance. Chip secures transactions with crypto.

More fraud protection than credit cards (transaction-bound authorization).



Load Protocol

Unlinked, cash-based load transaction (on-line).

Load value onto card using cash at load device.

Load device contains Load Security Application Module (LSAM): secure data processing and storage.

Card account balance adjusted; transaction data logged and sent to issuer for financial settlement.

Uses symmetric cryptography.

Load «data security»

Card «critical»
{secrecy={K_{CI}}}
{integrity={K_{CI},cep,nt,rc_{nt}}}

cep, nt, rc_{nt} : Data; K_{CI} : Keys

Init(lda,m)
Credit(s2,rl)

LSAM «critical»
{secrecy={K_{LI}}}
{integrity={K_{LI},lda,n,rl_n,$r2l_n$,r_n,m_n}}

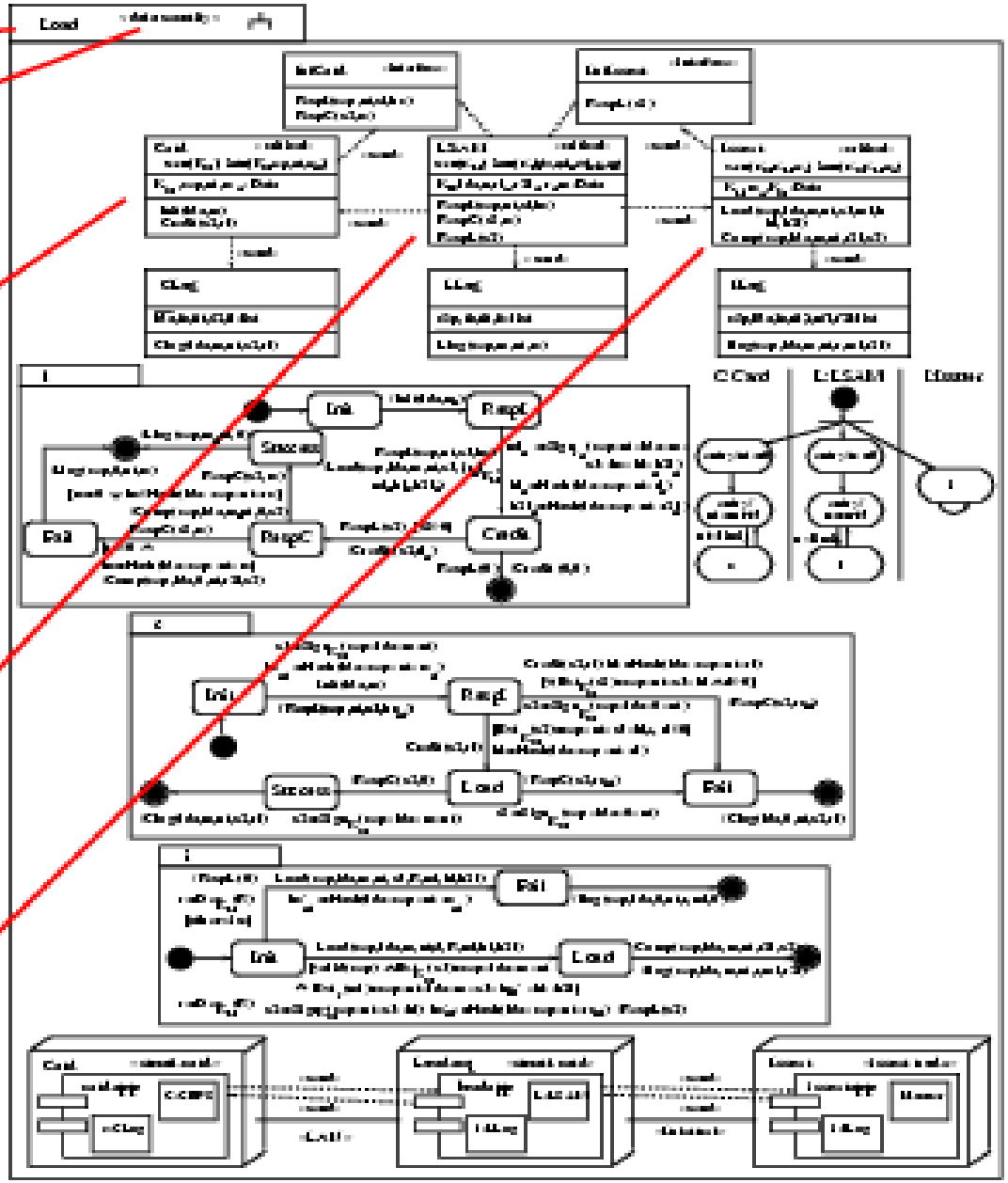
lda, n, rl_n, $r2l_n$, m_n : Data
K_{LI}, r_n : Keys

Resp1(cep,nt,sl,hc)
RespC(s3,rc)
RespL(s2)

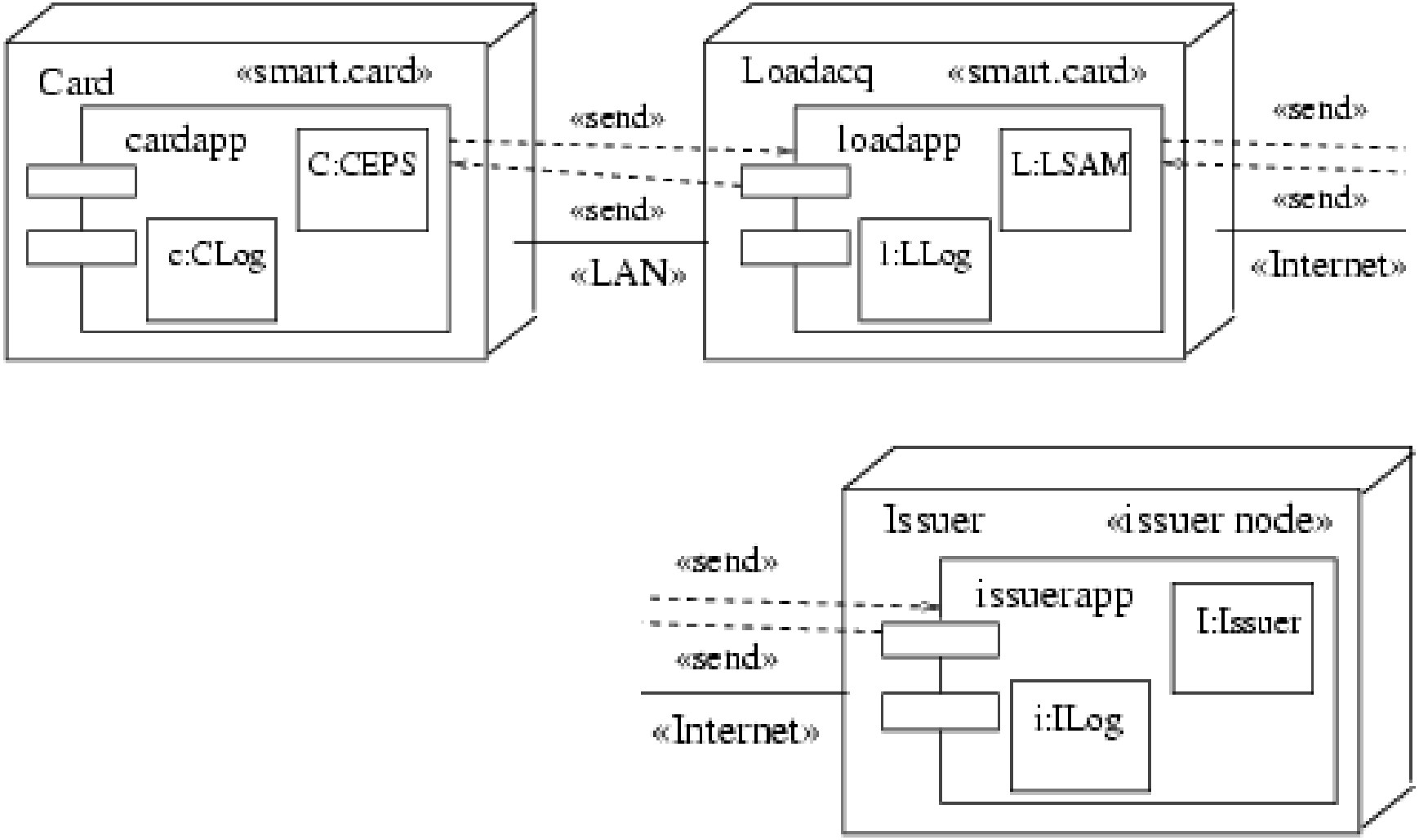
Issuer «critical»
{secrecy={K_{CI},K_{LI},rc_{nt}}}
{integrity={K_{CI},K_{LI},rc_{nt}}}

rc_{nt} : Data; K_{LI},K_{CI} : Keys

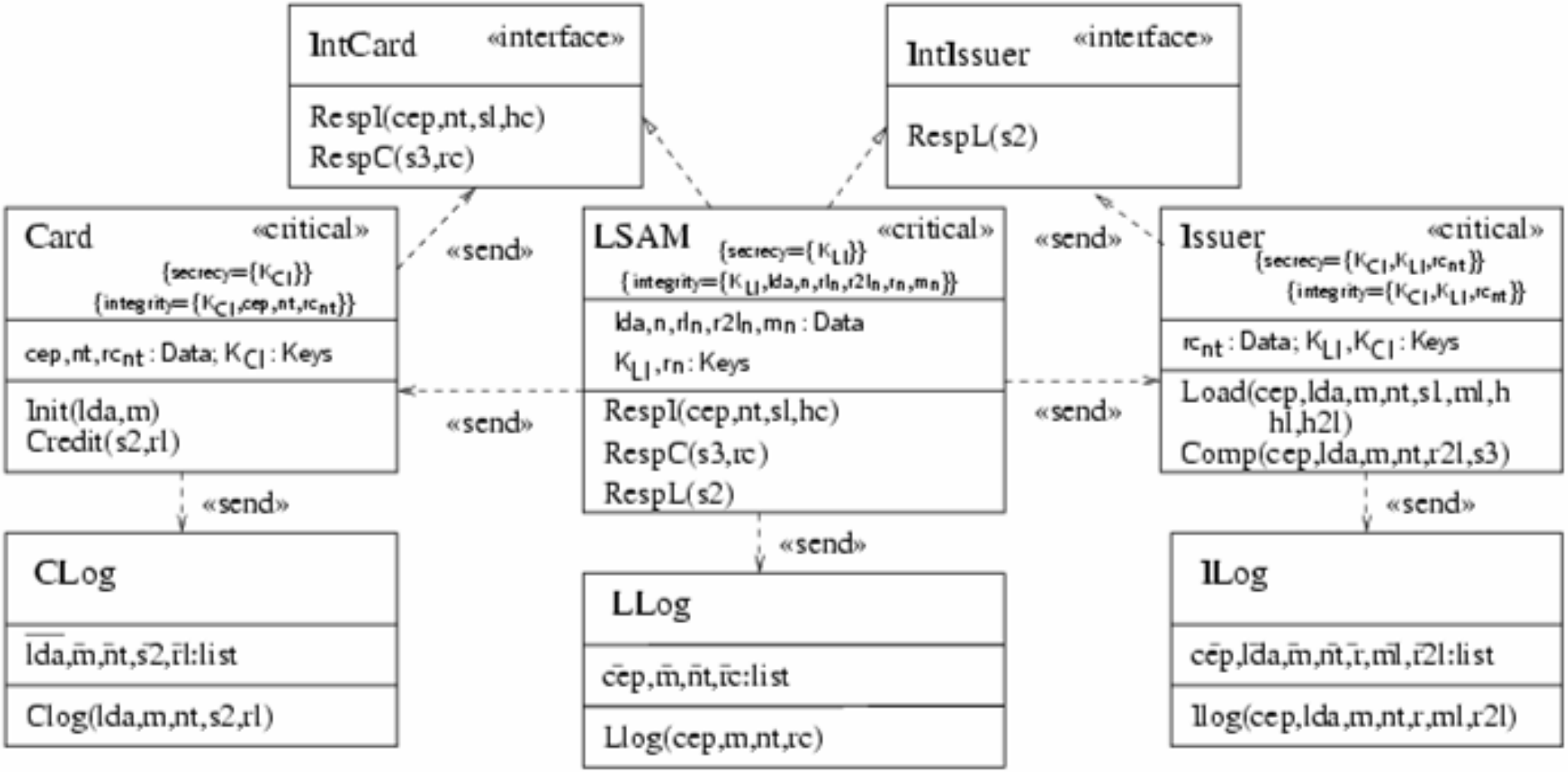
Load(cep,lda,m,nt,s1,ml,h
hl,h2l)
Comp(cep,lda,m,nt,r2l,s3)



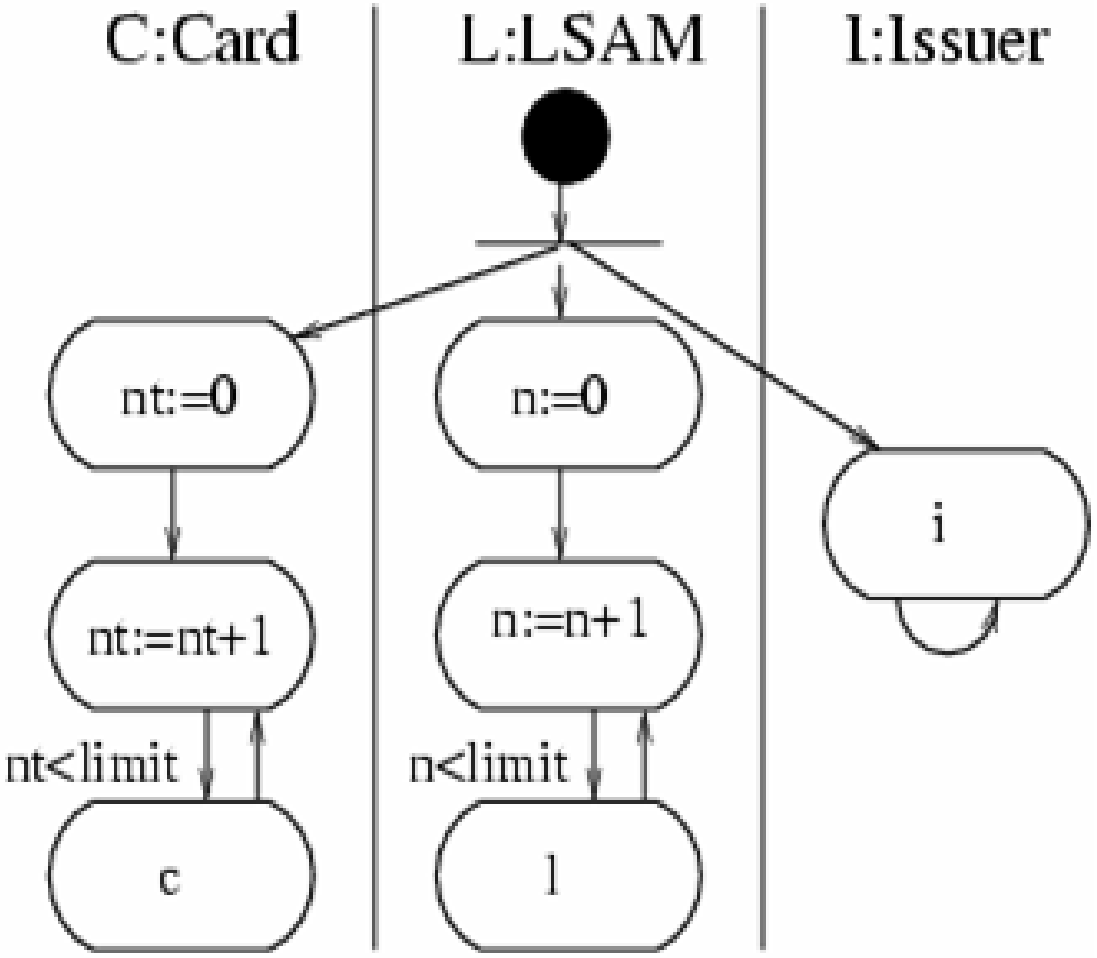
Load Protocol: Physical View



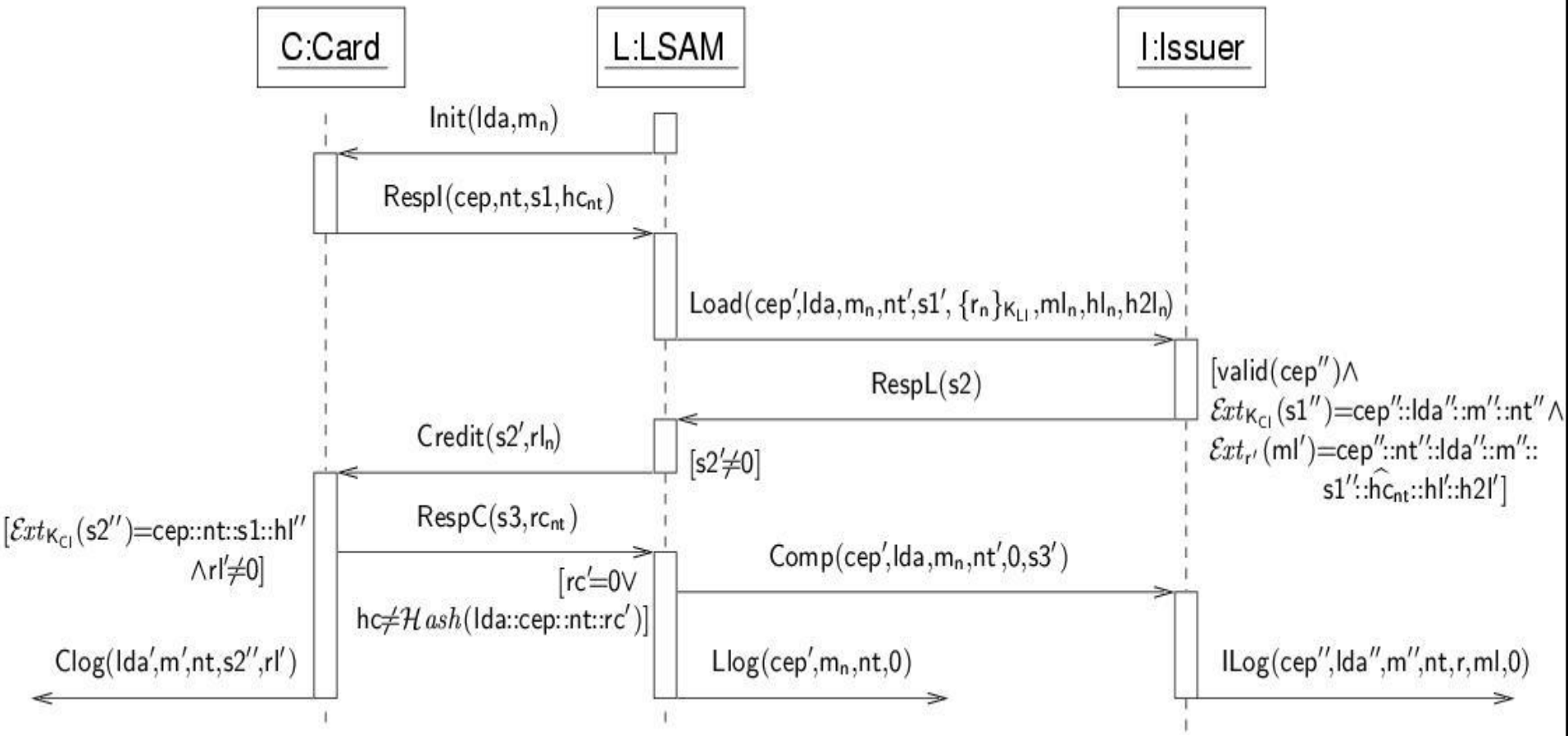
Load Protocol: Structural View



Load Protocol: Coordination View



Load Protocol: Interaction View



Security Threat Model

Card, LSAM, issuer security module assumed tamper-resistant.

Intercept communication links, replace components.

Possible attack motivations:

- **Cardholder**: charge without pay
- **Load acquirer**: keep cardholder's money
- **Card issuer**: demand money from load acquirer

May coincide or collude.

Audit Security

No direct communication between card and cardholder. Manipulate load device display.

Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception).

Security Conditions (informal)

Cardholder security: If card appears to have been loaded with *m* according to its logs, cardholder can prove to card Issuer that a load acquirer owes *m* to card issuer.

Load acquirer security: Load acquirer has to pay *m* to card issuer only if load acquirer has received *m* from cardholder.

Card issuer security: Sum of balances of cardholder and load acquirer remains unchanged by transaction.

Load Acquirer Security

Suppose card issuer *I* possesses $ml_n = Sign_{rn}(cep::nt::lda::m_n::s1::hc_{nt}::hl_n::h2l_n)$ and card *C* possesses rl_n, where $hl_n = Hash\ (lda::cep::nt::rl_n)$.

Then after execution either of following hold:

Llog(cep, lda, m_n, nt) has been sent to l:LLog (so load acquirer L has received and retains m_n in cash) or

Llog $(cep,\ lda,\ 0,\ nt)$ has been sent to l : LLog (so L returns m_n to cardholder) and L has received rc_{nt} with $hc_{nt} = Hash(lda::cep::nt::rc_{nt})$ (negating ml_n).

"ml_n provides guarantee that load acquirer owes transaction amount to card issuer" (CEPS)

Flaw

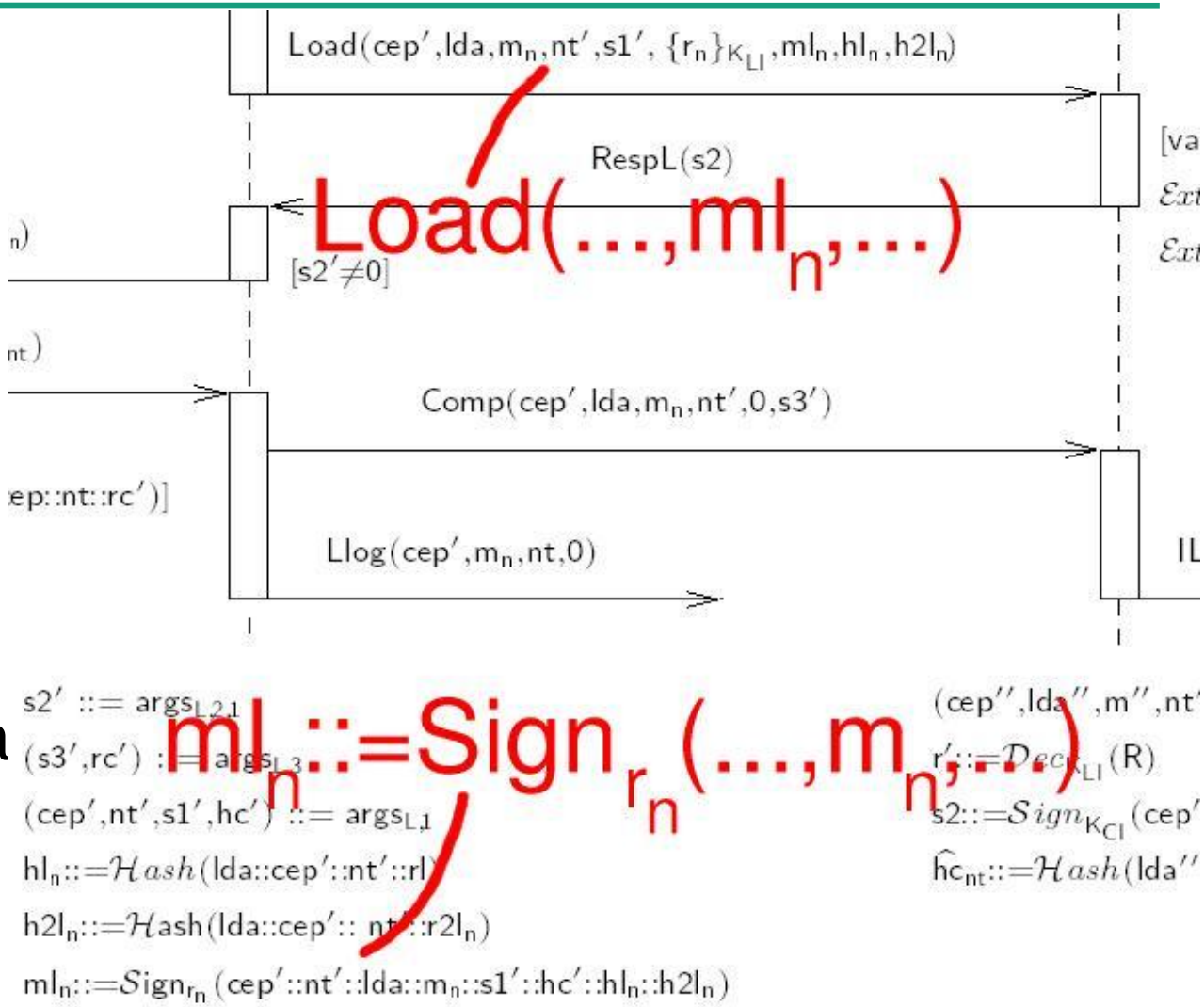
L does not provide load acquirer security against adversaries of type insider.

Why ?

Flaw

ml_n: „Proof“ for bank that load machine received money.

But: r_n shared between bank a load machine.



Correction

Modification: use asymmetric key in ml_n, include signature certifying hc_{nt}.

Verify this version wrt. above conditions.

Roadmap

1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

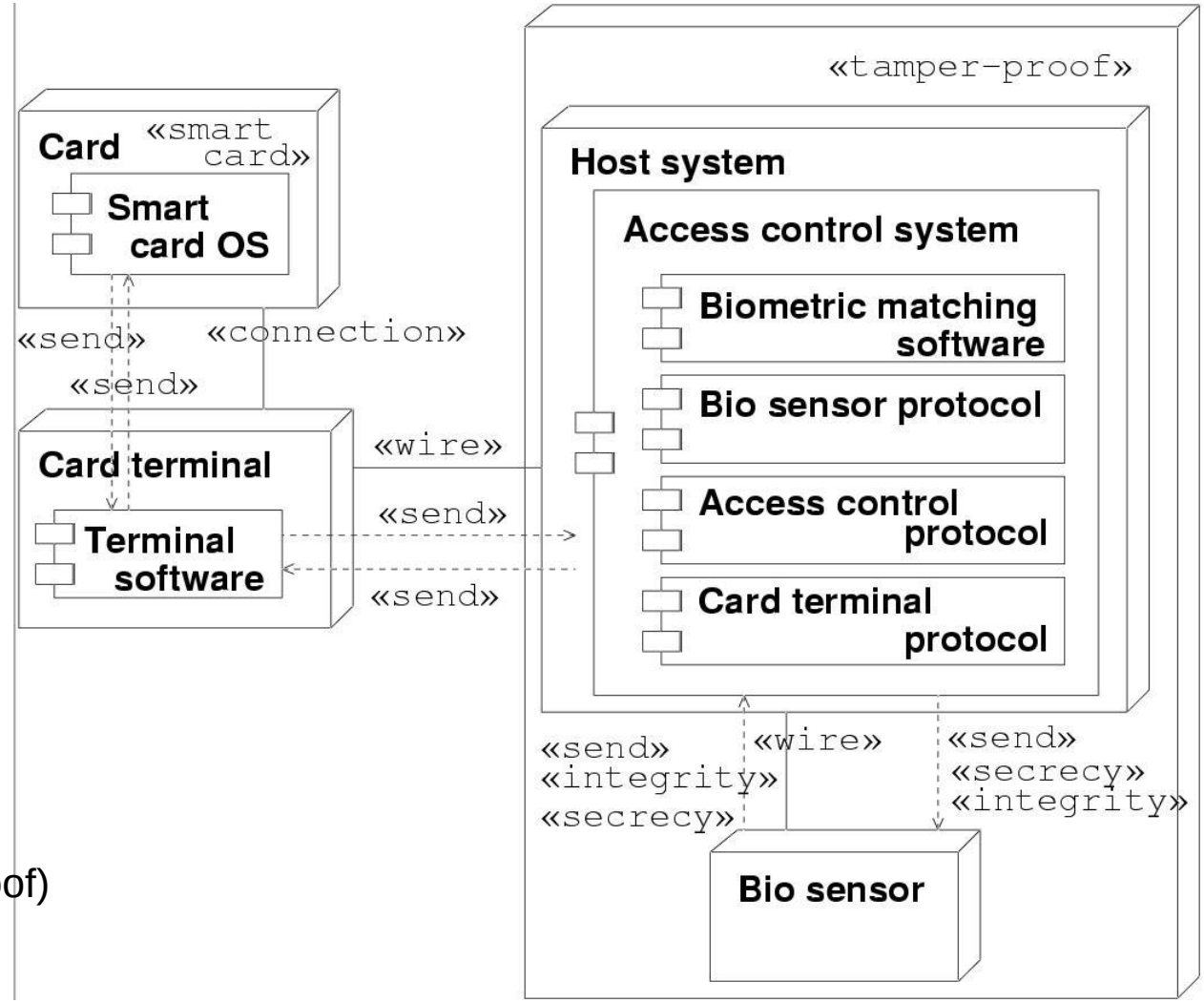
Biometric Authentication System

In development by large German telecommunication company.

In joint project, use presented security analysis tools at given UML specification.

So far, have discovered three major attacks against subsequently improved versions (misuse counter circumvented by dropping / replaying messages, smart-card insufficiently authenticated by recombing sessions).

Architecture



$Threats_{insider}$(connection)

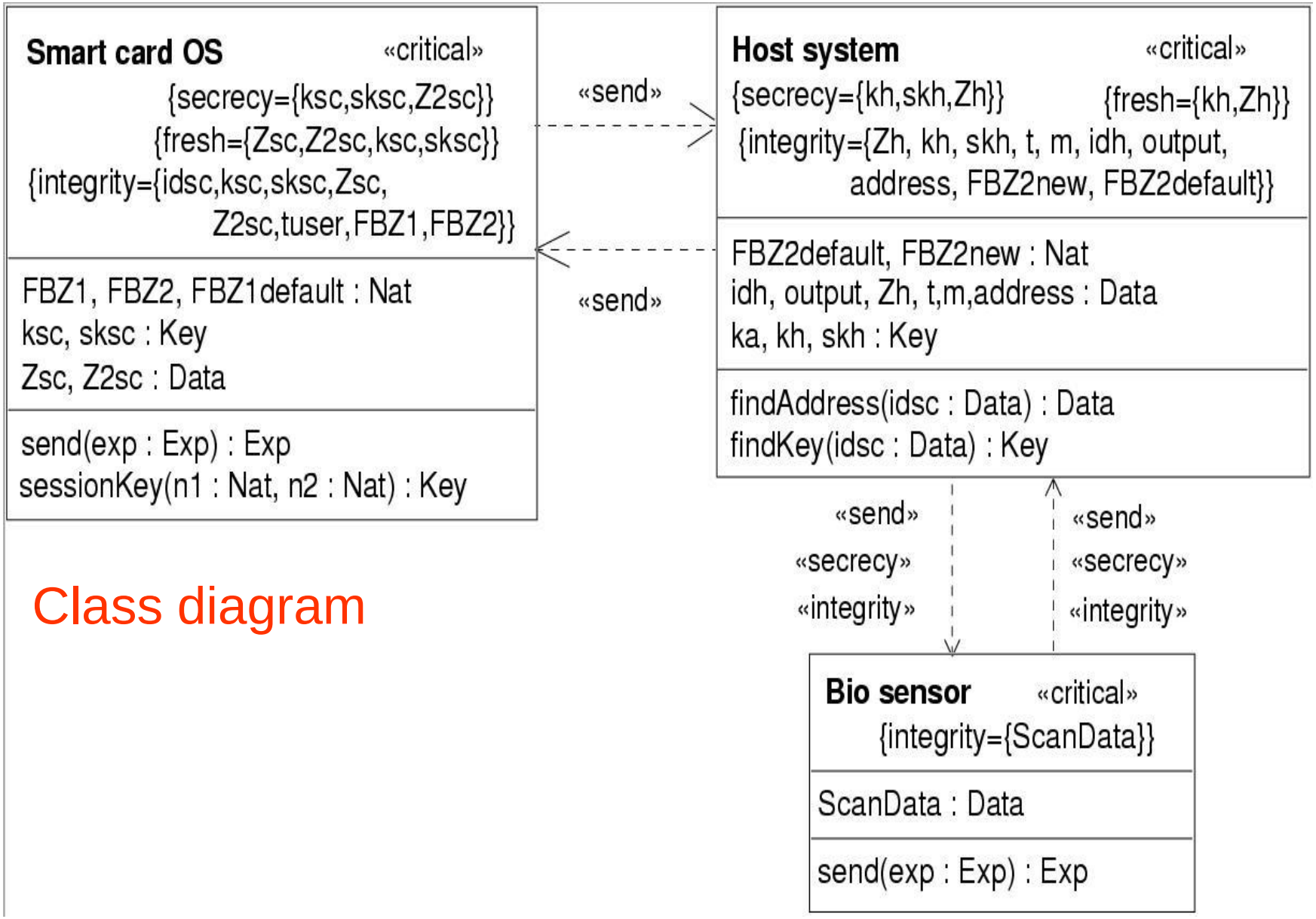
= $Threats_{insider}$ (wire)

= {read, write, delete}

$Threats_{insider}$(smartcard)

= $Threats_{insider}$ (tamper-proof)

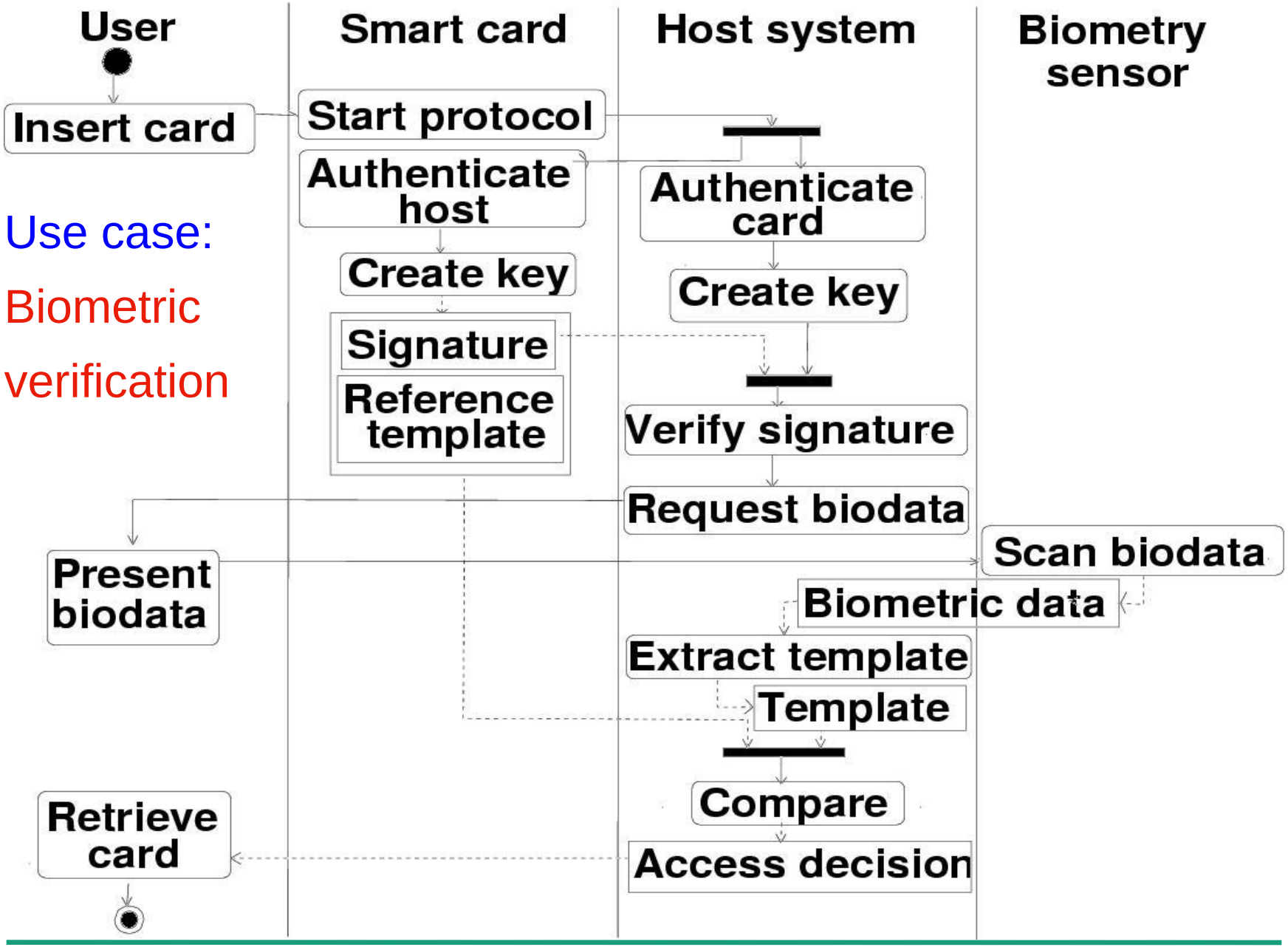
= Ø



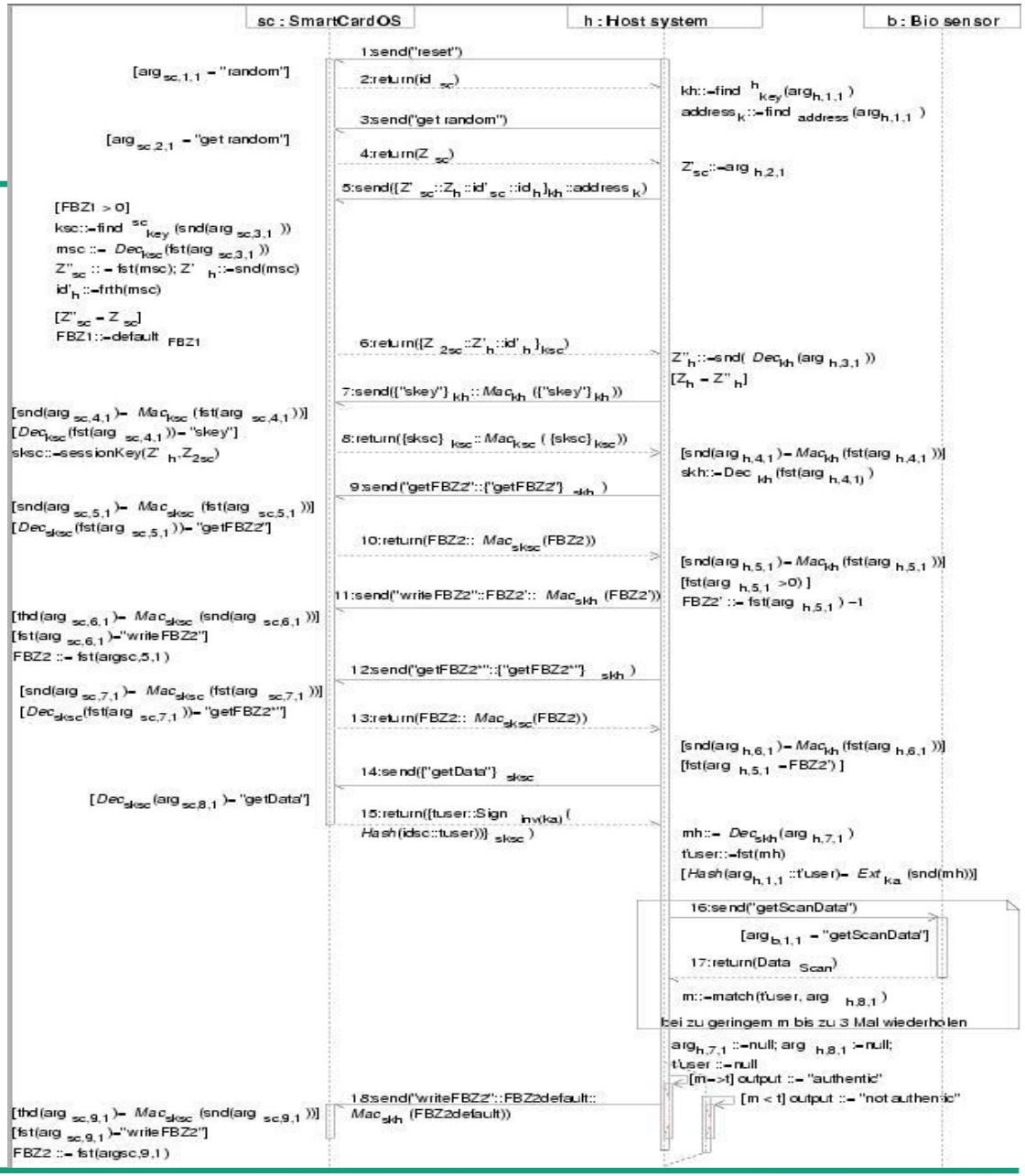
Class diagram

Use case:

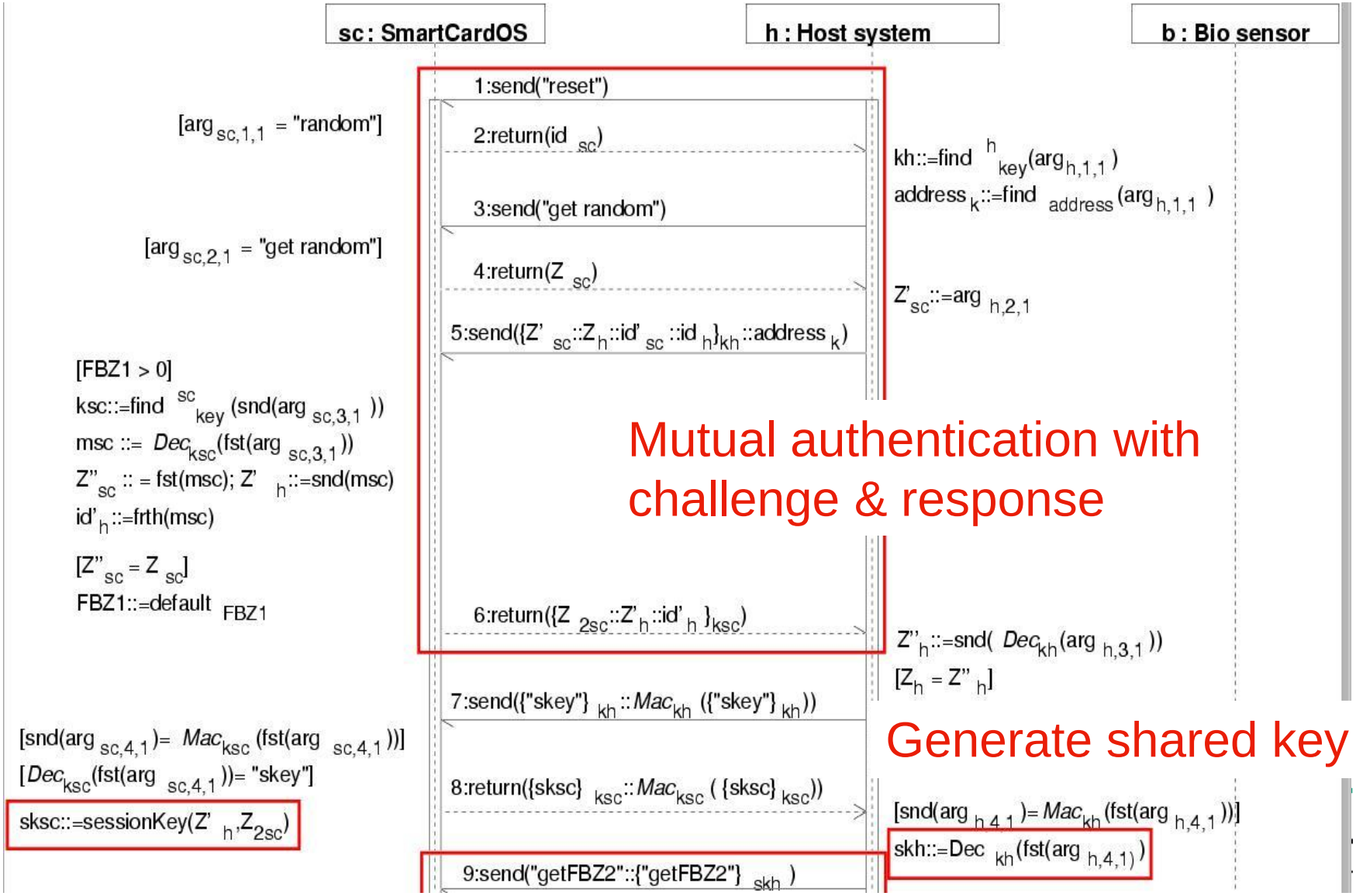
Biometric verification



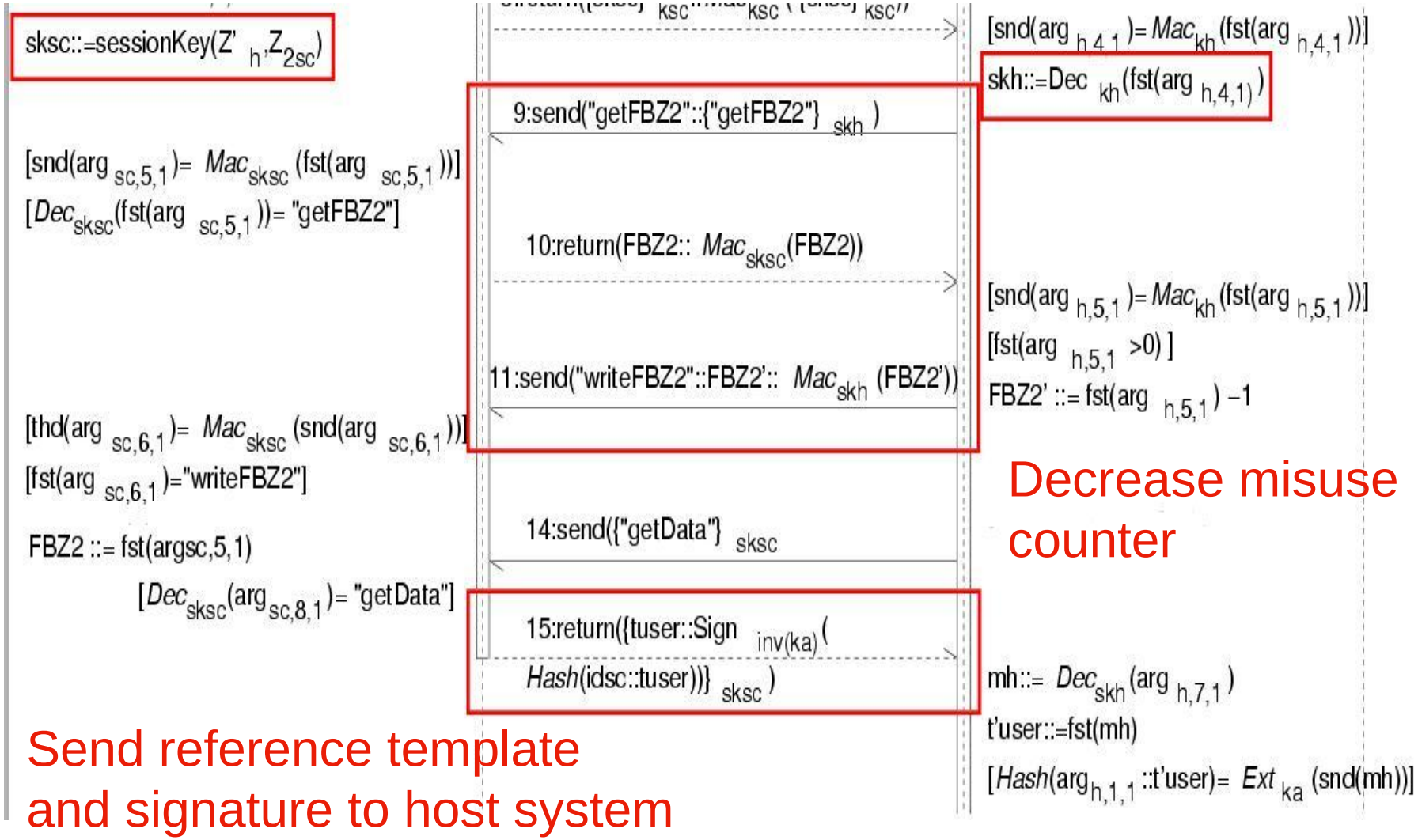
Protocol


sc : SmartCardOS
h : Host system
b : Bio sensor
1:send("reset")
[arg sc,1,1 = "random"]
2:return(id sc)
kh:=find h key(arg h,1,1)
address k:=find address(arg h,1,1)
3:send("get random")
[arg sc,2,1 = "get random"]
4:return(Z sc)
Z'sc:=arg h,2,1
5:send([Z' sc::Zh::id'sc::idh}kh::address k)
[FBZ1 > 0]
ksc:=find sc key(snd(arg sc,3,1))
msc :=- Dec ksc(fst(arg sc,3,1))
Z''sc :=- fst(msc); Z' h:=snd(msc)
id'h:=frth(msc)
[Z''sc = Z sc]
FBZ1:=default FBZ1
6:return([Z 2sc::Z'h::id'h}ksc)
Z''h:=snd(Dec kh(arg h,3,1))
[Zh = Z''h]
7:send(["skey"] kh::Mac kh(["skey"] kh))
[snd(arg sc,4,1)= Mac ksc(fst(arg sc,4,1))]
[Dec ksc(fst(arg sc,4,1))= "skey"]
sksc:=sessionKey(Z' h,Z2sc)
8:return({sksc} ksc::Mac ksc({sksc} ksc))
[snd(arg h,4,1)= Mac kh(fst(arg h,4,1))]
skh:=Dec kh(fst(arg h,4,1))
9:send("getFBZ2"::["getFBZ2"] skh)
[snd(arg sc,5,1)= Mac sksc(fst(arg sc,5,1))]
[Dec sksc(fst(arg sc,5,1))= "getFBZ2"]
10:return(FBZ2:: Mac sksc(FBZ2))
[snd(arg h,5,1)= Mac kh(fst(arg h,5,1))]
[fst(arg h,5,1 >0)]
FBZ2' :=- fst(arg h,5,1) -1
11:send("writeFBZ2"::FBZ2':: Mac skh(FBZ2'))
[thd(arg sc,6,1)= Mac sksc(snd(arg sc,6,1))]
[fst(arg sc,6,1)="writeFBZ2"]
FBZ2 :=- fst(argsc,5,1)
12:send("getFBZ2"::["getFBZ2"] skh)
[snd(arg sc,7,1)= Mac sksc(fst(arg sc,7,1))]
[Dec sksc(fst(arg sc,7,1))= "getFBZ2"]
13:return(FBZ2:: Mac sksc(FBZ2))
[snd(arg h,6,1)= Mac kh(fst(arg h,6,1))]
[fst(arg h,5,1 =FBZ2')]
14:send(["getData"] sksc
[Dec sksc(arg sc,8,1)= "getData"]
15:return({tuser::Sign inv(ka)(Hash(idsc::tuser))} sksc)
mh:=- Dec skh(arg h,7,1)
tuser:=fst(mh)
[Hash(arg h,1,1::tuser)= Ext ka(snd(mh))]
16:send("getScanData")
[arg b,1,1 = "getScanData"]
17:return(Data Scan)
m:=match(tuser, arg h,8,1)
bei zu geringem m bis zu 3 Mal wiederholen
arg h,7,1 :=null; arg h,8,1 :=null;
tuser :=null
[m->t] output :=- "authentic"
[m < t] output :=- "not authentic"
18:send("writeFBZ2"::FBZ2default::Mac skh(FBZ2default))
[thd(arg sc,9,1)= Mac sksc(snd(arg sc,9,1))]
[fst(arg sc,9,1)="writeFBZ2"]
FBZ2 :=- fst(argsc,9,1)

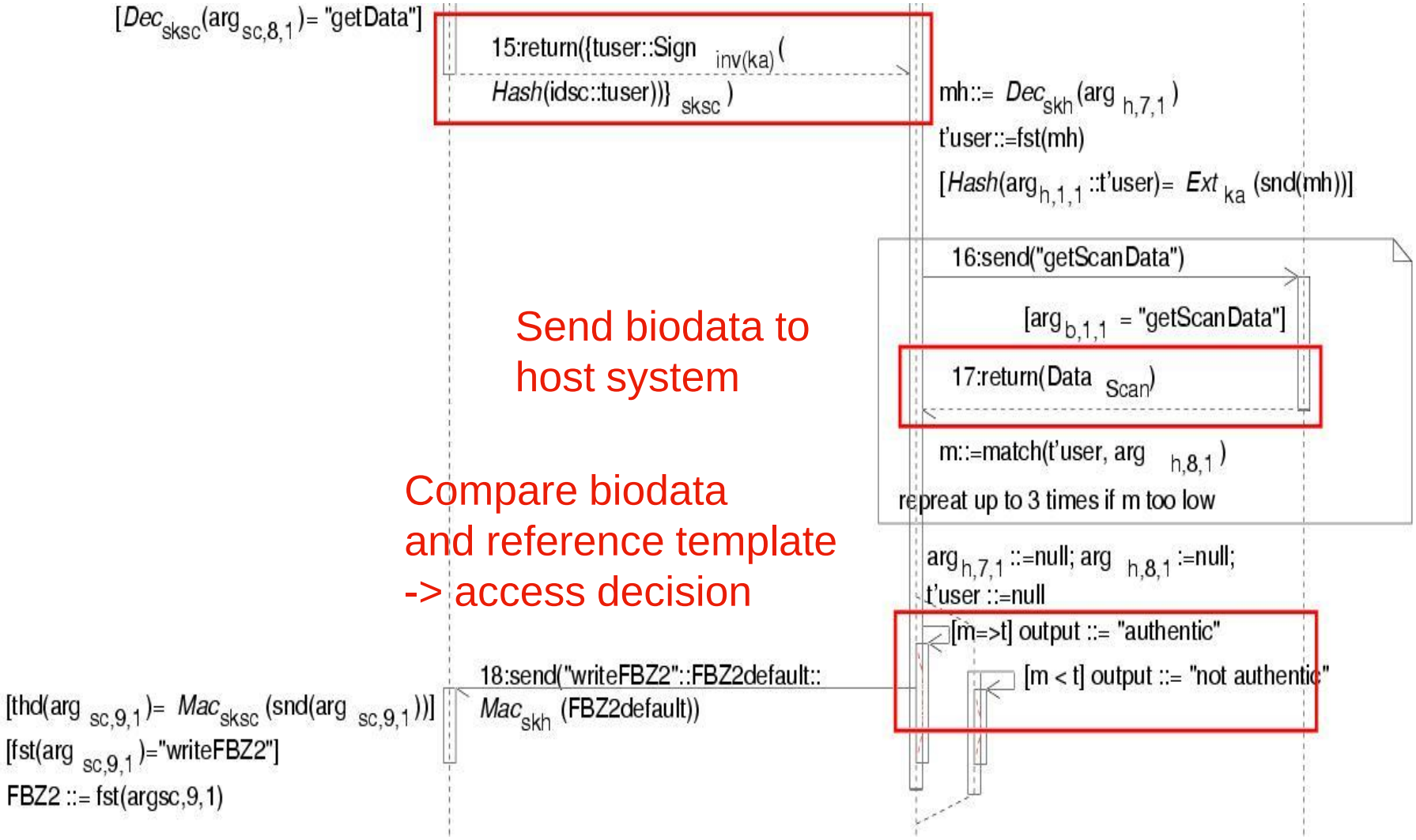
Authentic. Protocol Part 1



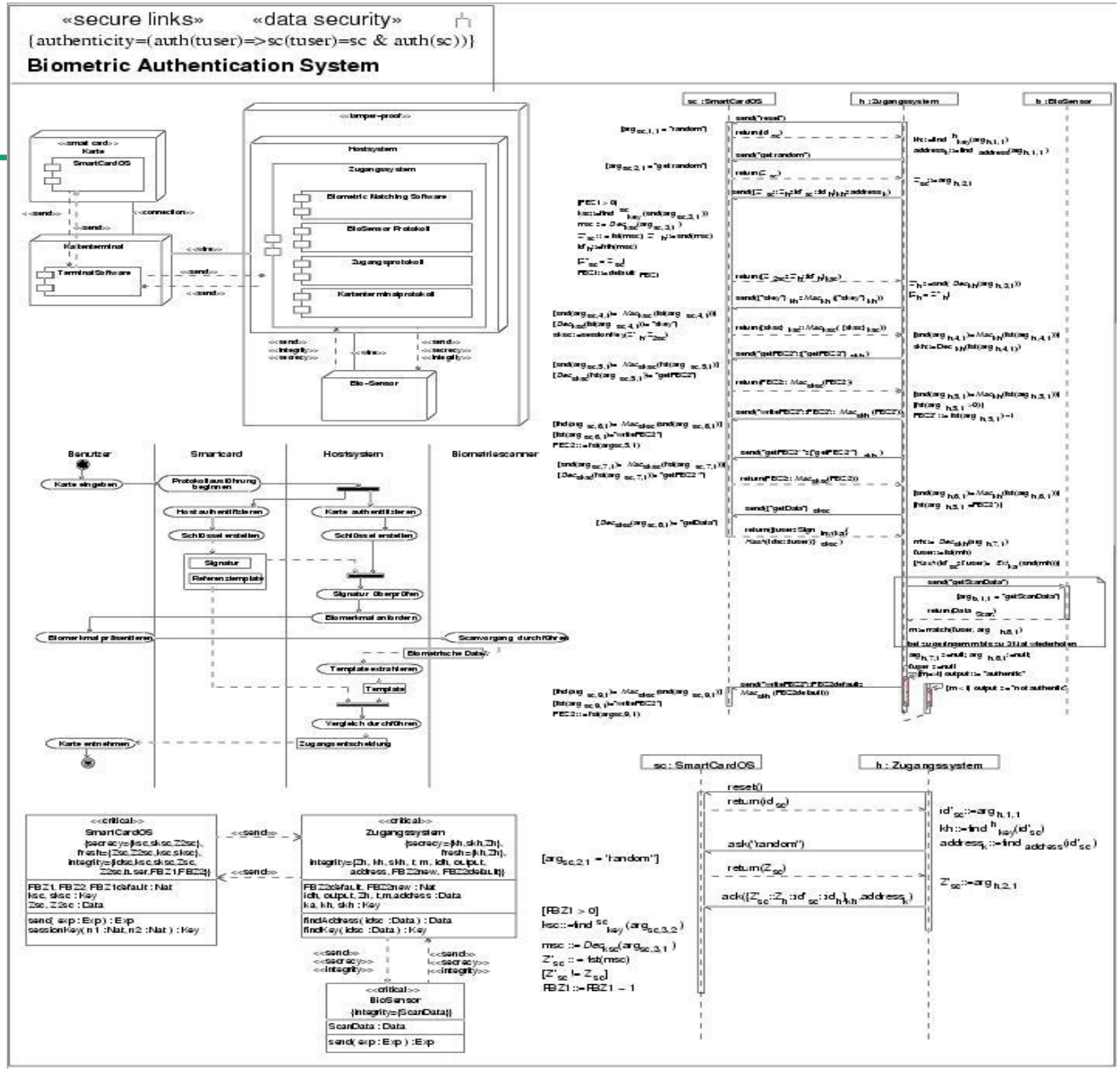
Authentication Protocol Part 2



Authentication Protocol Part 3



Big Picture


«secure links» «data security»
{authenticity=(auth(tuser)=>sc(tuser)=sc & auth(sc))}
Biometric Authentication System
Smart card
Karte
SmartCardOS
Kartenterminal
Terminal Software
Hostsystem
Zugangssystem
Biometric Matching Software
BioSensor Protokoll
Zugangsprotokoll
Kartenterminalprotokoll
Bio-Sensor
Benutzer
Smartcard
Hostsystem
Biometriescanner
Karte eingeben
Protokollausführung beginnen
Host authentifizieren
Schlüssel erstellen
Signatur
Referenztemplate
Karte authentifizieren
Signatur überprüfen
Biomerkmal anfordern
Biomerkmal präsentieren
Scanvorgang durchführen
Biometrische Daten
Template extrahieren
Template
Vergleich durchführen
Zugangsentscheidung
Karte entnehmen
sc : SmartCardOS
h : Zugangssystem
b : BioSensor
reset()
ask('random')
SmartCardOS
Zugangssystem
BioSensor

Translation to First-order Logic II

Message order not enforced by smart card (!).

Connection from smart card

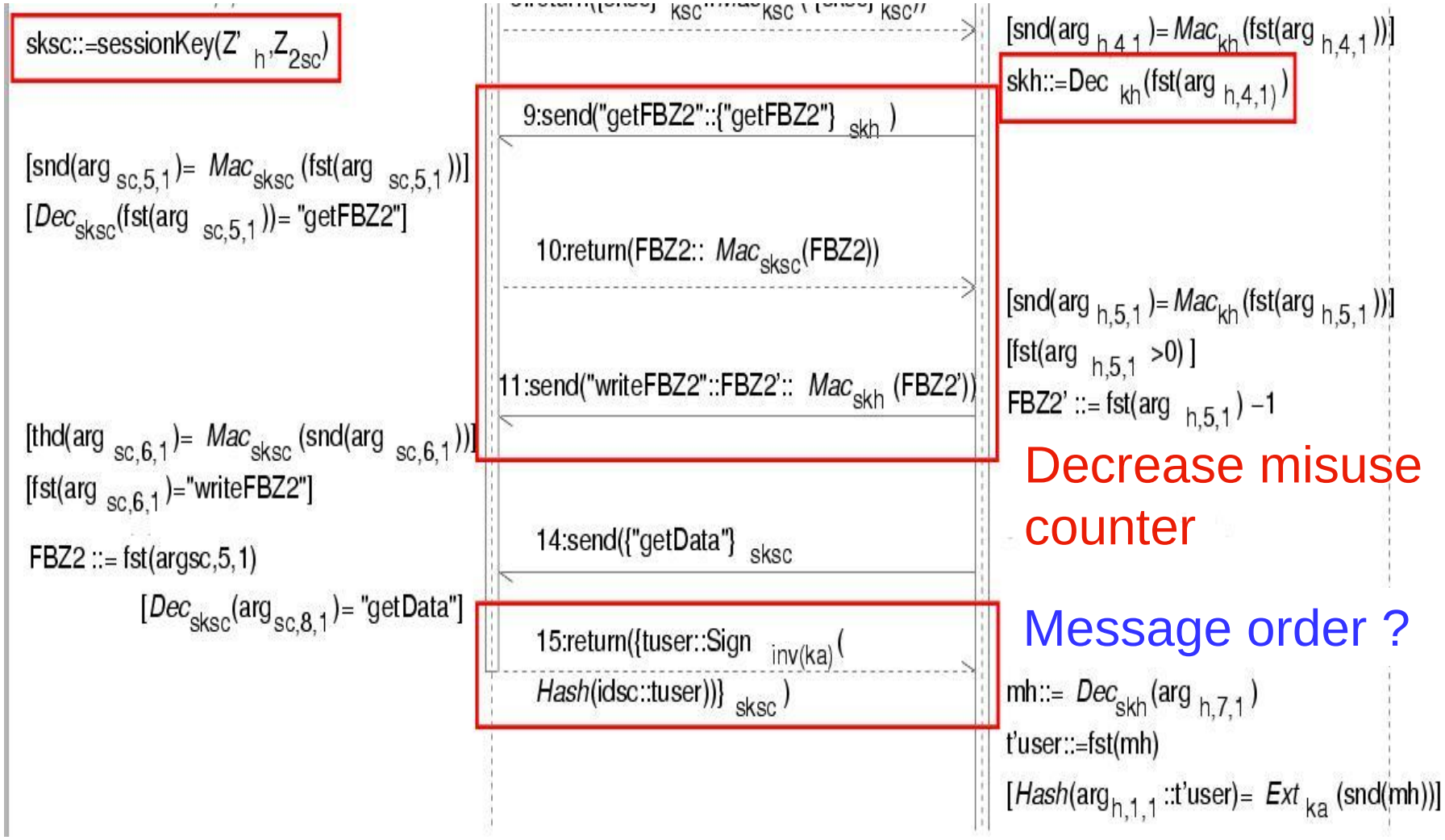
$TR1=(in(msg\_in),cond(msg\_in),out(msg\_out))$

followed by $TR2$ gives predicate

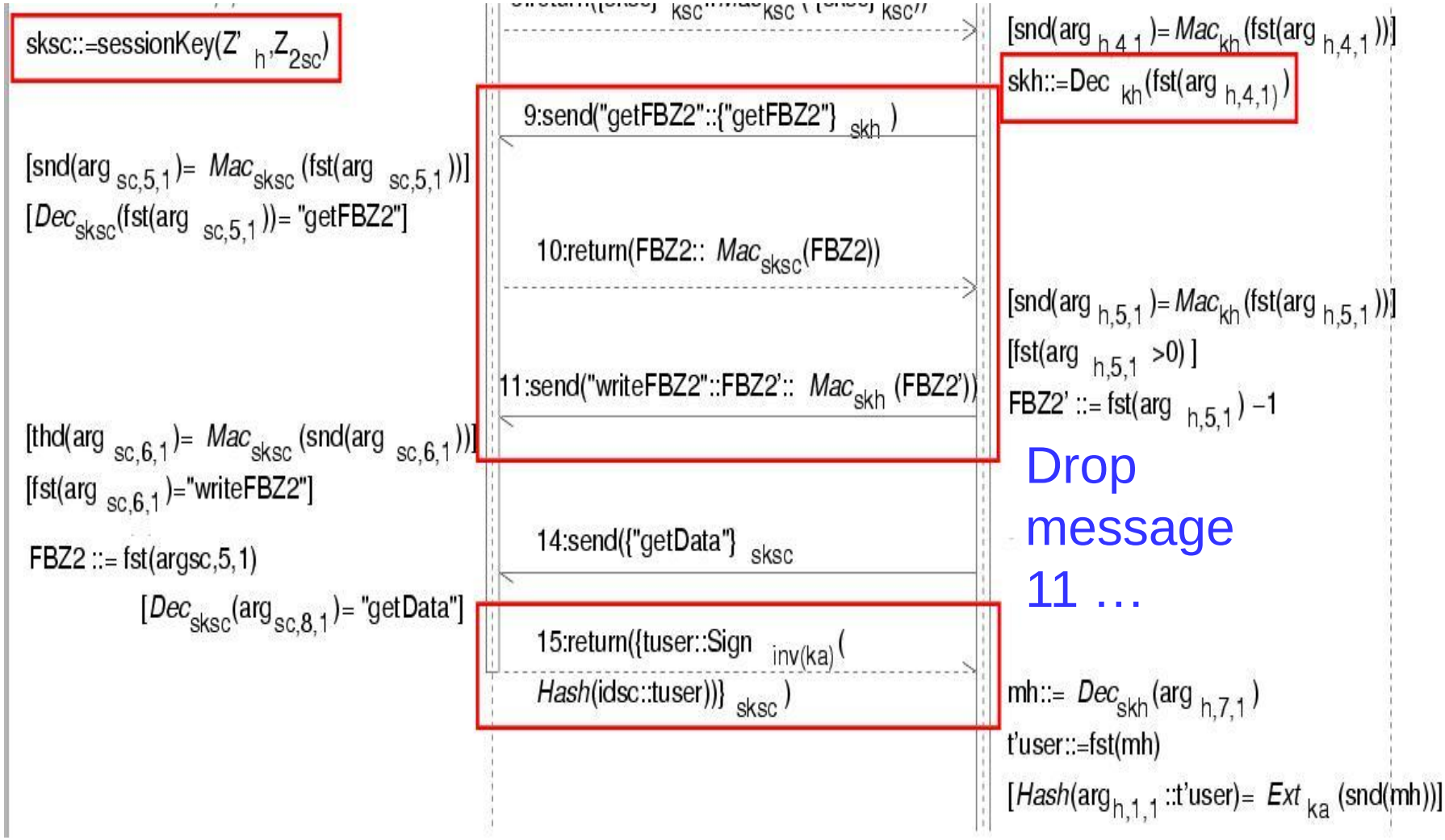
$PRED(TR1)=$

$$\forall msg\_in.\ [knows(msg\_in) \wedge cond(msg\_in) \Rightarrow knows(msg\_out)]$$
$$\wedge\ PRED(TR2)$$

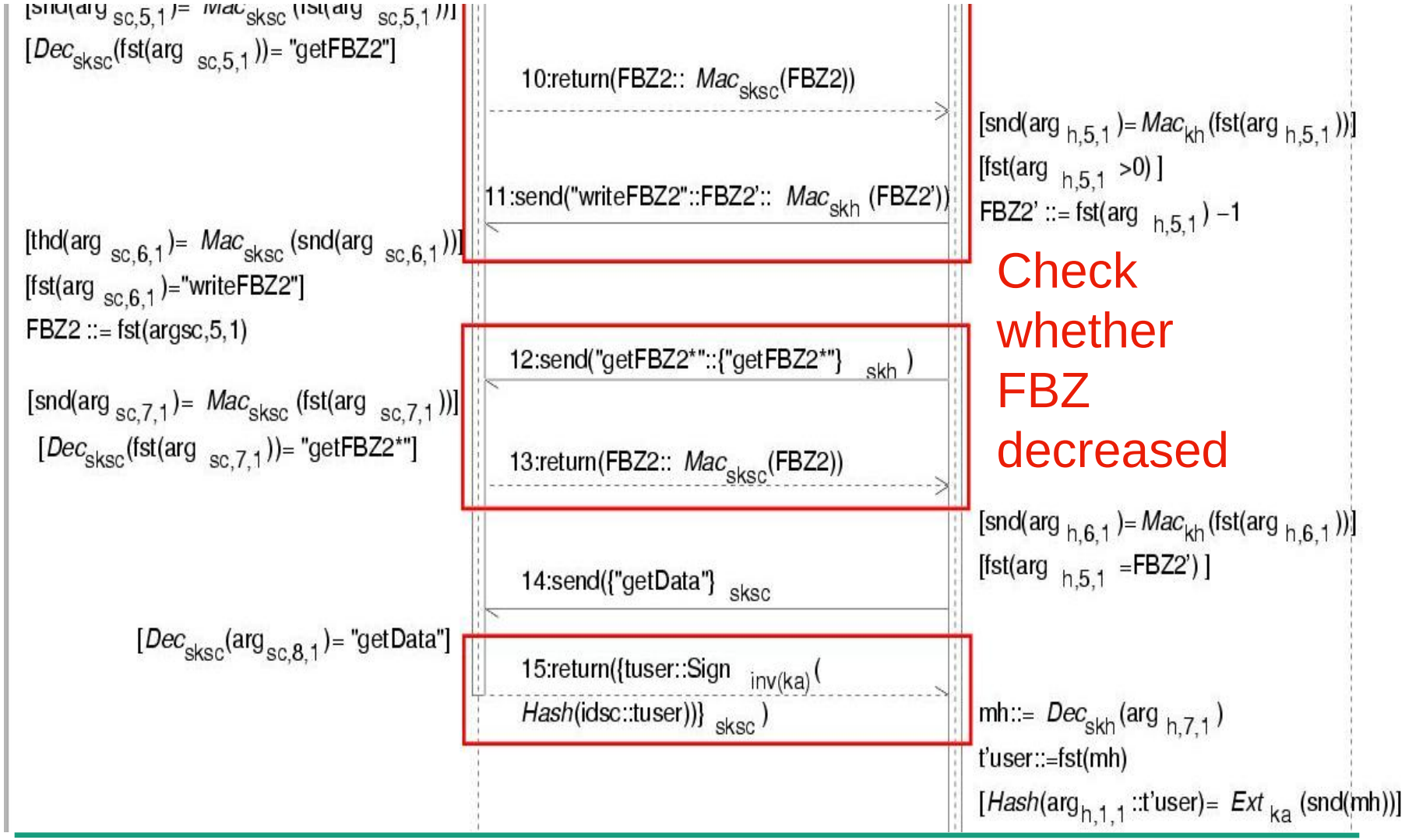
Authent. Protocol Pt. 2: Problem ?



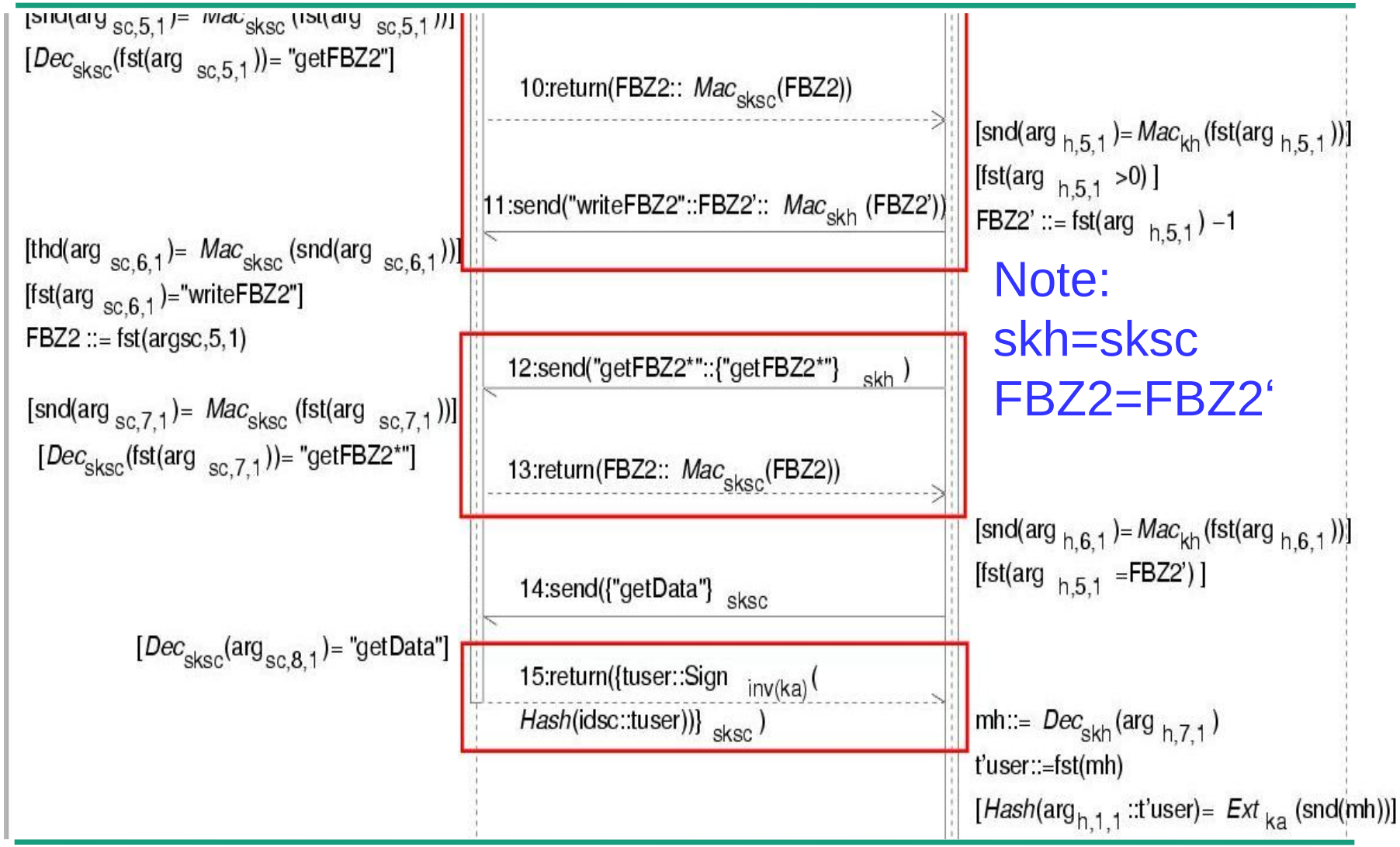
Authent. Protocol Pt. 2: Problem.



Authent. Protocol Pt. 2: Improvement

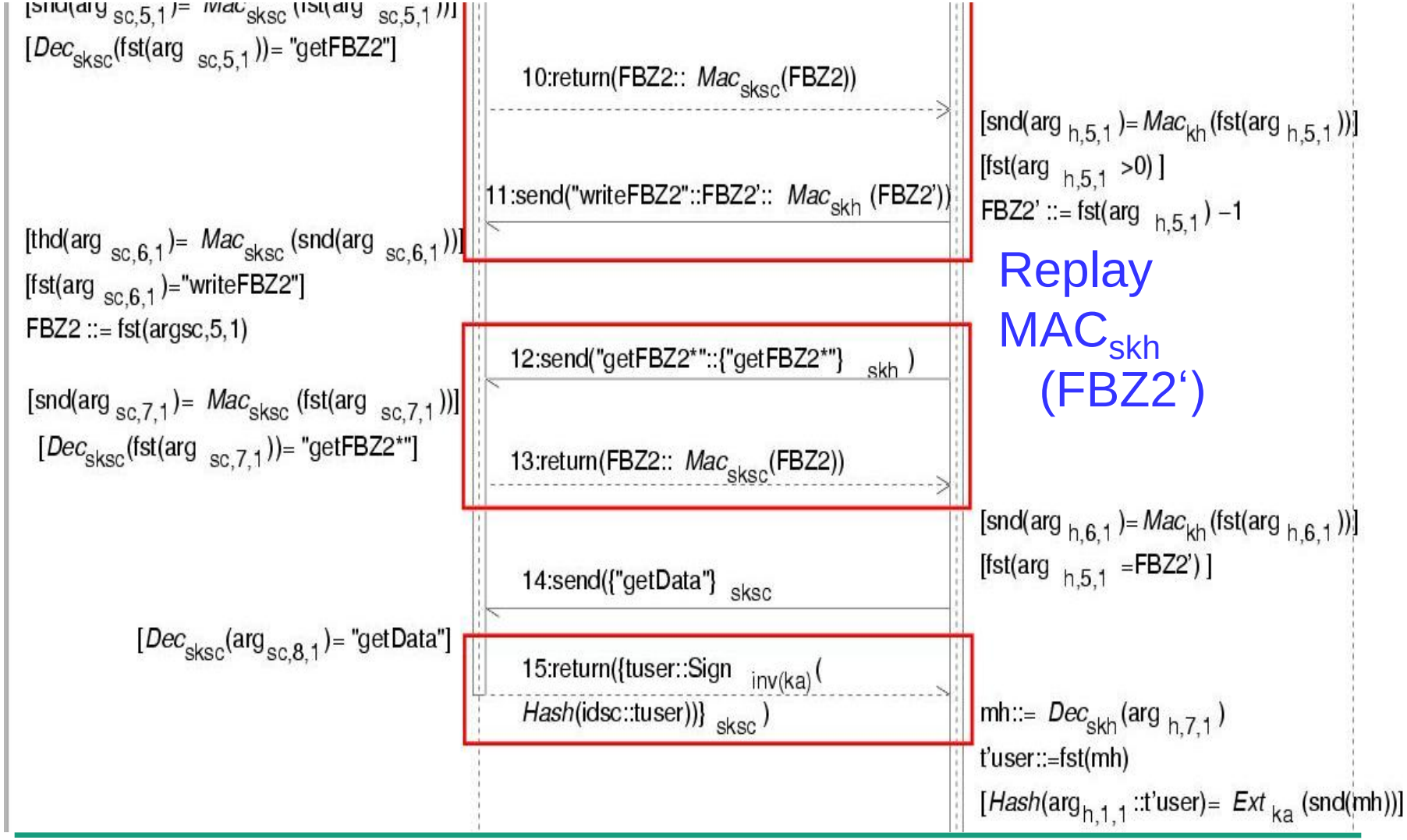


Authent. Prot. Pt. 2: Improvement ?

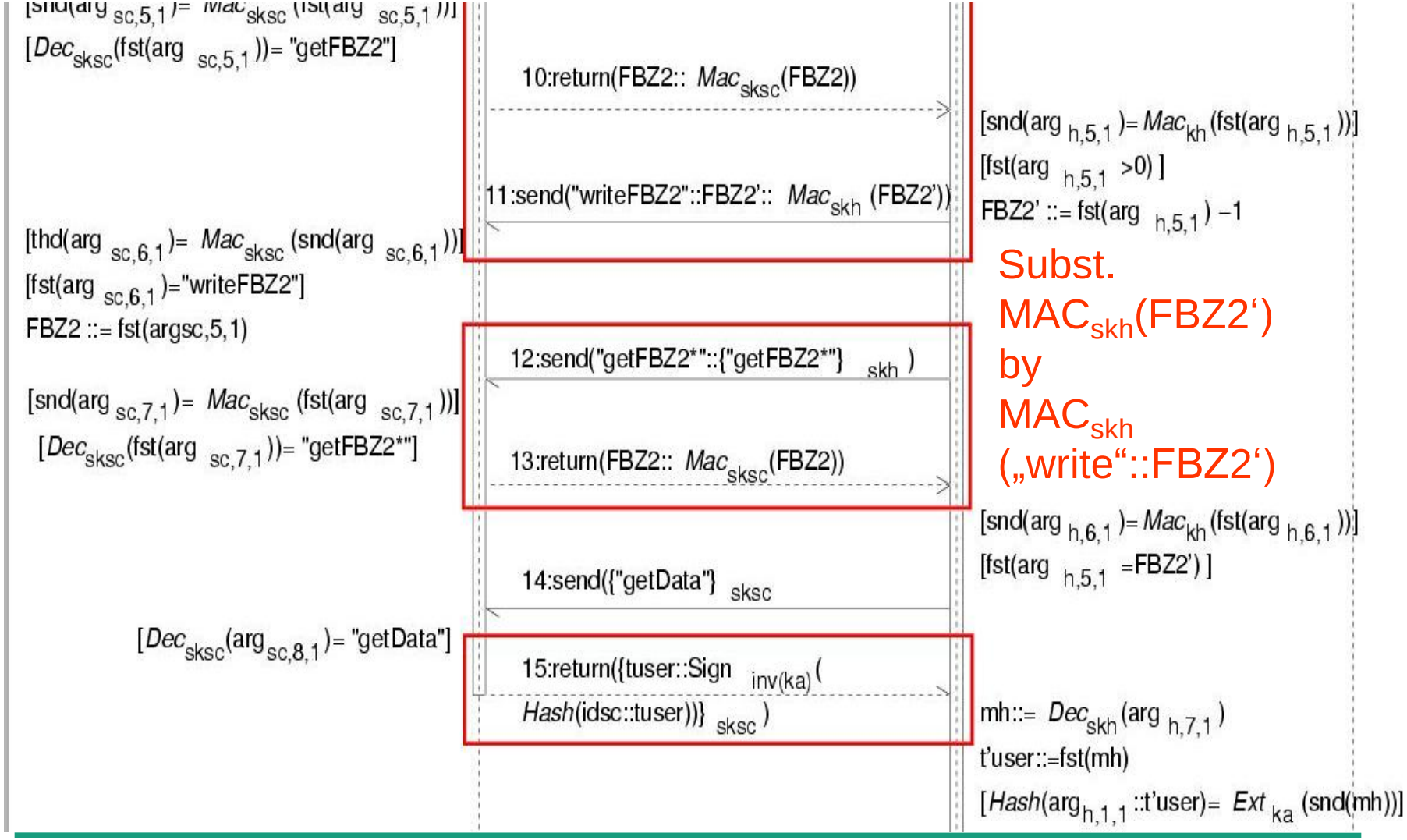


Note:
skh=sksc
FBZ2=FBZ2‘

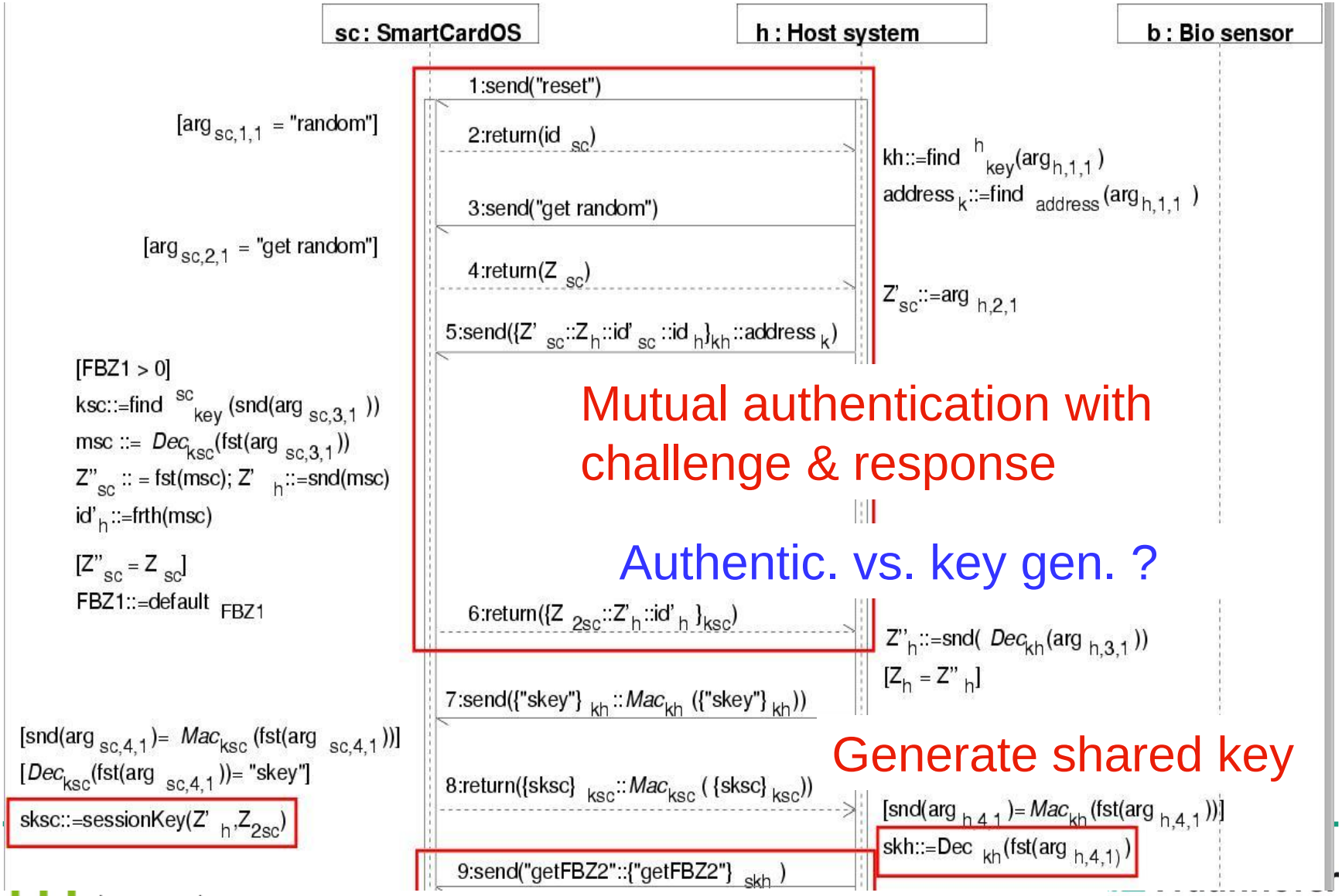
Authent. Prot. Pt. 2: Problem



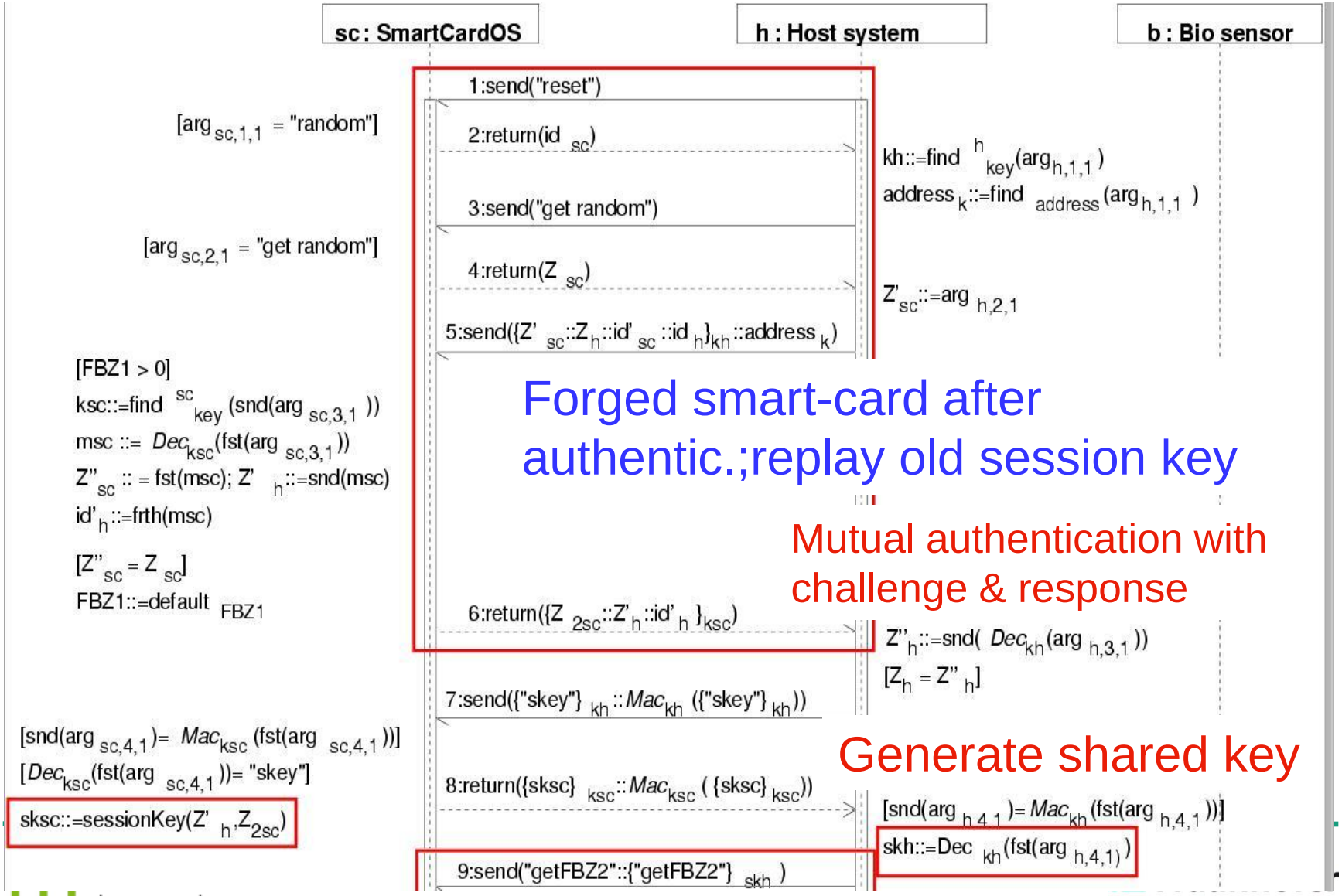
Authent. Prot. Pt. 2: Improvement (?)



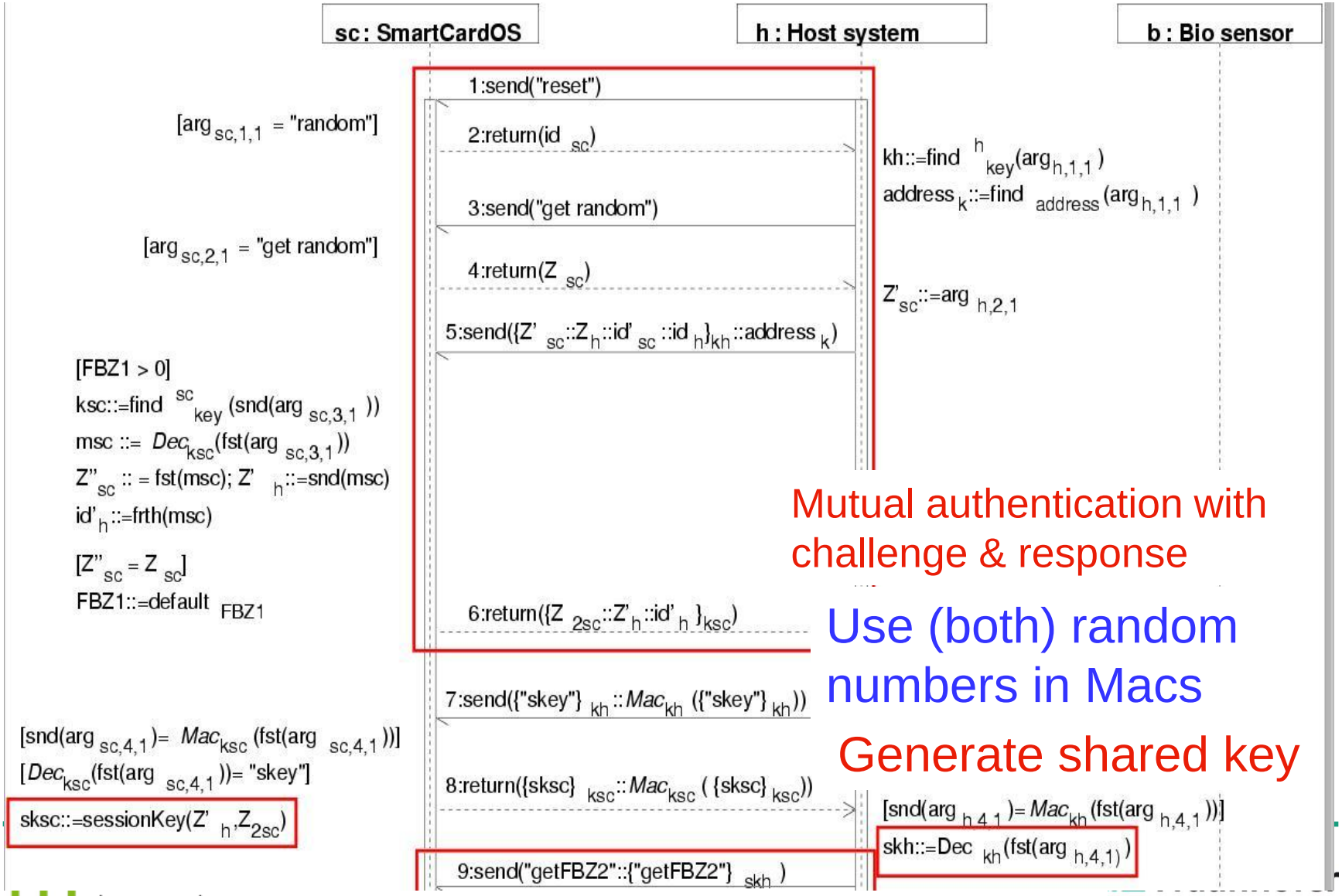
Authentic. Protocol Part 1: Problem ?



Authentic. Protocol Part 1: Problem.



Authentic. Protocol Part 1: Improvement (?)



Roadmap

1. Motivation and Overview
2. Specifying Security with UMLsec
3. Security Architectures
4. Cryptographic Protocols
5. Formal Security Analysis, General Results
6. Models versus Code
7. Run-time Verification
8. Software Evolution
9. Electronic Purses, Biometric Authentication
10. Other Applications

Bank Application

Security analysis of web-based banking application, to be put to commercial use (clients fill out and sign digital order forms).

Layered security protocol (first layer: SSL protocol, second layer: client authentication protocol)

Security requirements:

- confidentiality
- authenticity

Layered Security Architectures

Layer *on top* uses security *below*.

client authenticity

→

confidentiality, integrity, server authenticity

→ ?

=

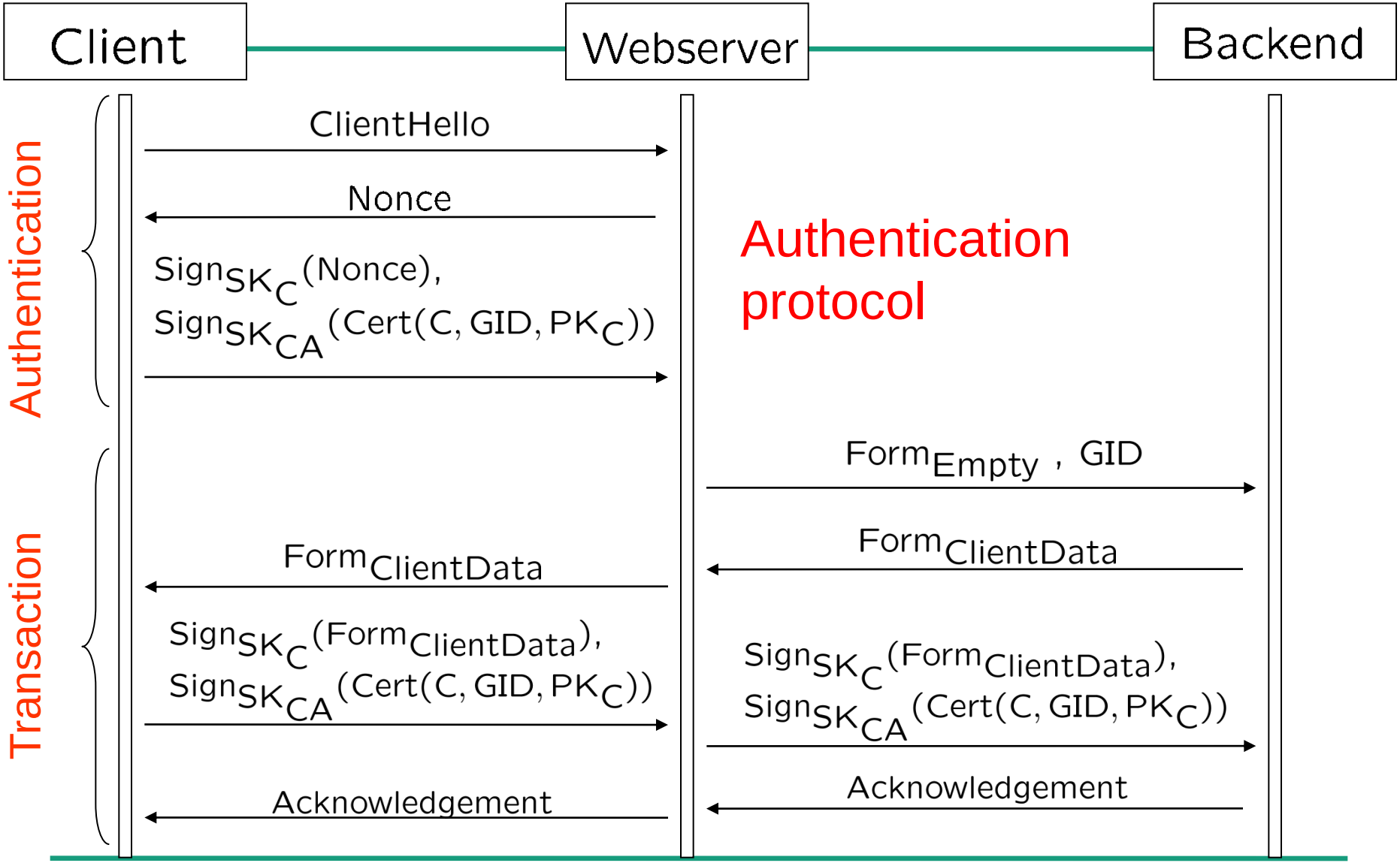
confidentiality, … + client authenticity

→

Security properties additive ?

The Application II

- Two layer architecture.
- When user logs on, an SSL-connection is established (first layer).
  - Provides secrecy, integrity, server authentication but no client authentication (this version).
- Custom-made protocol on top of SSL for client authentication.
- Session key generated by SSL used to encrypt messages on second layer.


Client
Webserver
Backend
Authentication
Transaction
ClientHello
Nonce
$Sign_{SK_C}(Nonce)$,
$Sign_{SK_{CA}}(Cert(C, GID, PK_C))$
Authentication protocol
$Form_{Empty}$, GID
$Form_{ClientData}$
$Form_{ClientData}$
$Sign_{SK_C}(Form_{ClientData})$,
$Sign_{SK_{CA}}(Cert(C, GID, PK_C))$
$Sign_{SK_C}(Form_{ClientData})$,
$Sign_{SK_{CA}}(Cert(C, GID, PK_C))$
Acknowledgement
Acknowledgement

Layered Security Protocol

- Adjust adversary model to account for SSL security properties.
- Justify that specialized adversary model wrt. top-level protocol is as powerful as generic adversary wrt. protocol composition.
- Verify top-level protocol wrt. specialized adversary.
- Implies verification of protocol composition.

Insight

Layered Security Architectures indeed additive wrt. security properties in this particular case.

Generalize to classes of systems and security requirements !?

Further Applications

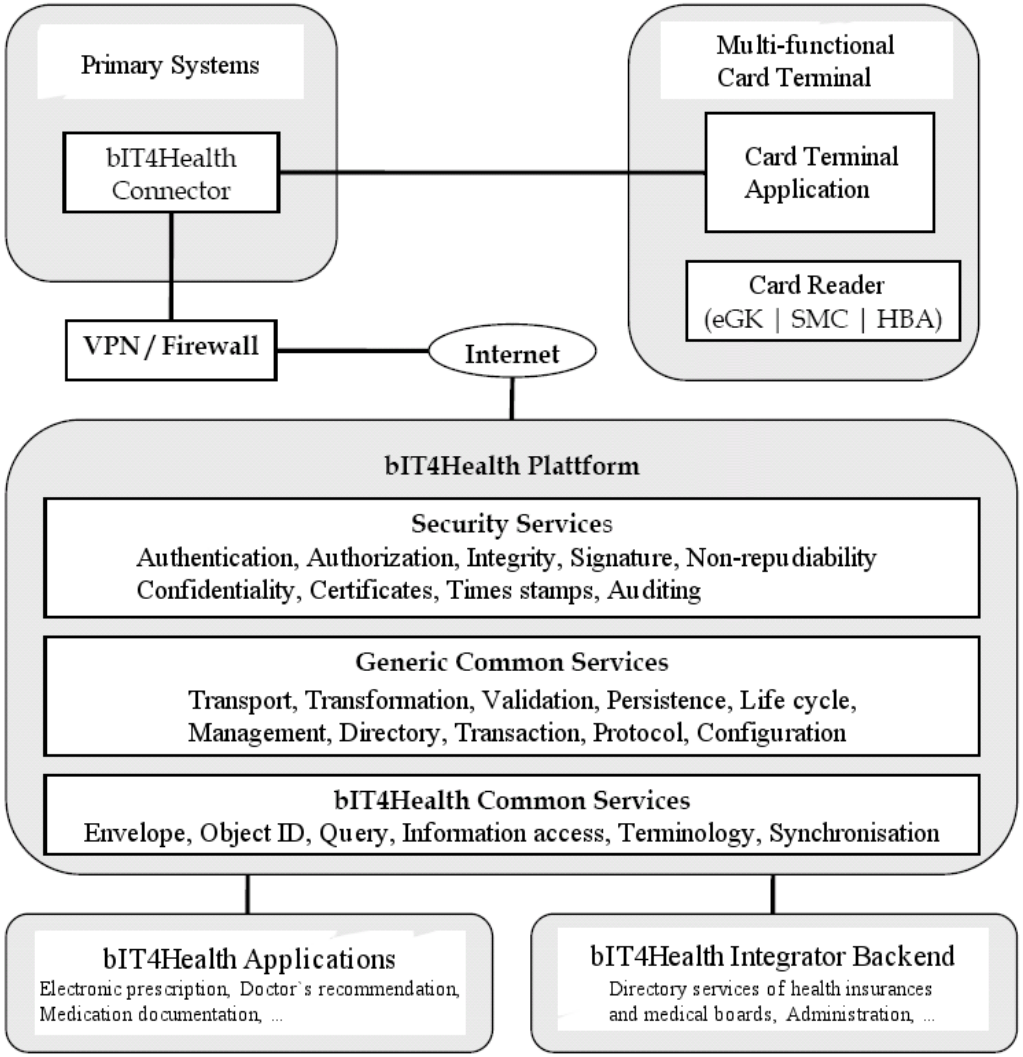
- German Health-card: Architecture analyzed with UMLsec, some weaknesses found [Jour. Meth. Inform. Medicine 08]
- Internal information system [ICSE 07]
- Internet bank architecture at HypoVereinsbank [SAFECOMP 03]
- Common Electronic Purse Specifications (Global standard for electronic purses): several weaknesses found [IFIPSEC 01, ASE 01]
- Biometric authentication systems: several weaknesses found [ACSAC 05, Models 09]
- Health information systems [Caise 09]
- Return-on-Security Investment Analysis
- Analysis of digital signature architecture
- IT security risk modelling
- Smart-card software update platform

Ongoing:

- Cloud-user security analysis
- Cloud-provider security analysis
- Security economics analysis

German Health Card Architecture

- Analyzed architecture against security requirements using UMLsec
- Detected several security weaknesses in the architecture



[Meth. Inform. Medicine 08]

Application: Mobile Communikation at O_2

UMLsec-based security analysis of regulations for the use of mobile devices at O_2 (Germany)

Extracted 62 security requirements from the security policy.

21 business-process relevant requirements, modelled in 8 activity diagrams, using the UMLsec-stereotypes <<fair exchange>> and <<provable>>.

10 data security requirements (confidentiality, integrity), modelled in deployment diagrams.

3 requirements on role-based access control (RBAC).

15 requirements regarding the protection of network services and use of firewalls and antivirus-software (modelled using extension of UMLsec)

13 requirements could not be modelled directly in UMLsec.

J. Jürjens, J. Schreck, P. Bartmann. 2008. Model-based security analysis for mobile communications. 30th International Conference on Software engineering (ICSE '08). ACM

Nr.	Sicherheitsanforderungen	<<Secure Links>>, Secure Links with XML	<<Fair Exchange>>	<<Provable>>	SecrecyIntegrity	TPTP-Datei z. Analyse v. Netzwerk-architekturen
1.9	Authentifizierung des Benutzers (Mitarbeiters) gegenüber dem Endgerät durch Chipkarten			X		
1.10	Verschlüsselung der auf den mobilen Endgeräten befindlichen Daten	X				
1.14	Keine zum Fernzugang parallele Verbindungen in andere Netze - Vermeidung der Kopplung mit unsicheren Netzen durch Umgehung der Firewall					X
1.26	Starke Verschlüsselung der Verbindung zwischen Endgerät und Fernzugangs-Server	X				
1.37	Bei O_2 übliche Virenschutzprogramme auf den Endgeräten					X
1.38	Aktualisierung des Virenschutzes über Fernzugang					

Validation Example:
Internal information system at BMW

MetaSearch Engine: personalized search in corporate intranet (password-protected).

Some documents are very security-critical. [ICSE 07]

Over 1.000 potential users, 280.000 documents, 20.000 requests per day.

Seamlessly integrated into enterprise security architecture.

Provides security services for applications (user authentication, role-based access, global single-sign-on), starting point for further security services.

Successfully analyzed with UMLsec.

BMW Group

Some Empirical Results

Is model-based quality assurance worthwhile compared to classical QA techniques (e.g. testing)?

[Testcom '05]

1) Static Analysis vs. Code Review: Industrial software at O2 (Germany) examined for errors. Result:

- Static-analysis only finds certain error classes, but very reliably.
- Most important aim: reduce “false positive”-rate.

2) Model-checking vs. Simulation / Tests: door control (in coop. w. BMW). Typical error classes:

[Models '08]

- Simulation / testing finds many “simple” errors fast and effectively (e.g. incorrect transition priority: few min.)
- Model-checking also finds obscure errors (e.g. race conditions), but with additional effort (1-2 days for LTL formula).

Model-based Security: Some Milestones

2001: UMLsec: UML profile for security modelling (*Jürjens*)
Model-based security testing with AutoFocus (*Wimmel, Jürjens*)

2002: Secure UML: Modelling RBAC with UML (*Basin et al.*)
Hypermedia security modeling with Ariadne (*Aedo, Diaz et al.*)
Aspect-oriented Security Modelling (*France et al.*)
Model-based IT security risk assessment (*Stølen et al.*)
Interactive theorem proving of UML models for security (*Haneberg, Reif et al.*)

2003: Formal verification for UML models of access control (*Koch, Parisi-Presicce*)

2004: Automated verification tools for UMLsec (*Shabalin et al.*)
Actor-centric modeling of user rights with UML (*Breu et al.*)
Extending OCL for secure database development (*Fernández-Medina et al.*)

2005: First book on model-based security published (in English)

2007: Security monitors for UML policy models (*Massacci et al.*)

2008: Executable misuse cases for security concerns (*Whittle et al.*)

2009: Model-based security vs performance evaluation (*Woodside et al.*)
First book on model-based security in Chinese

2010: From requirements to UMLsec models (*Houmb et al.; Islam et al.; Mouratidis et al.*)
Security monitoring for UMLsec models (*Bauer et al.; Pironti et al.*)
… : Model-based security monitor generation for embedded systems (*Schürr et al.*)

Model-based Security: Where are we today?

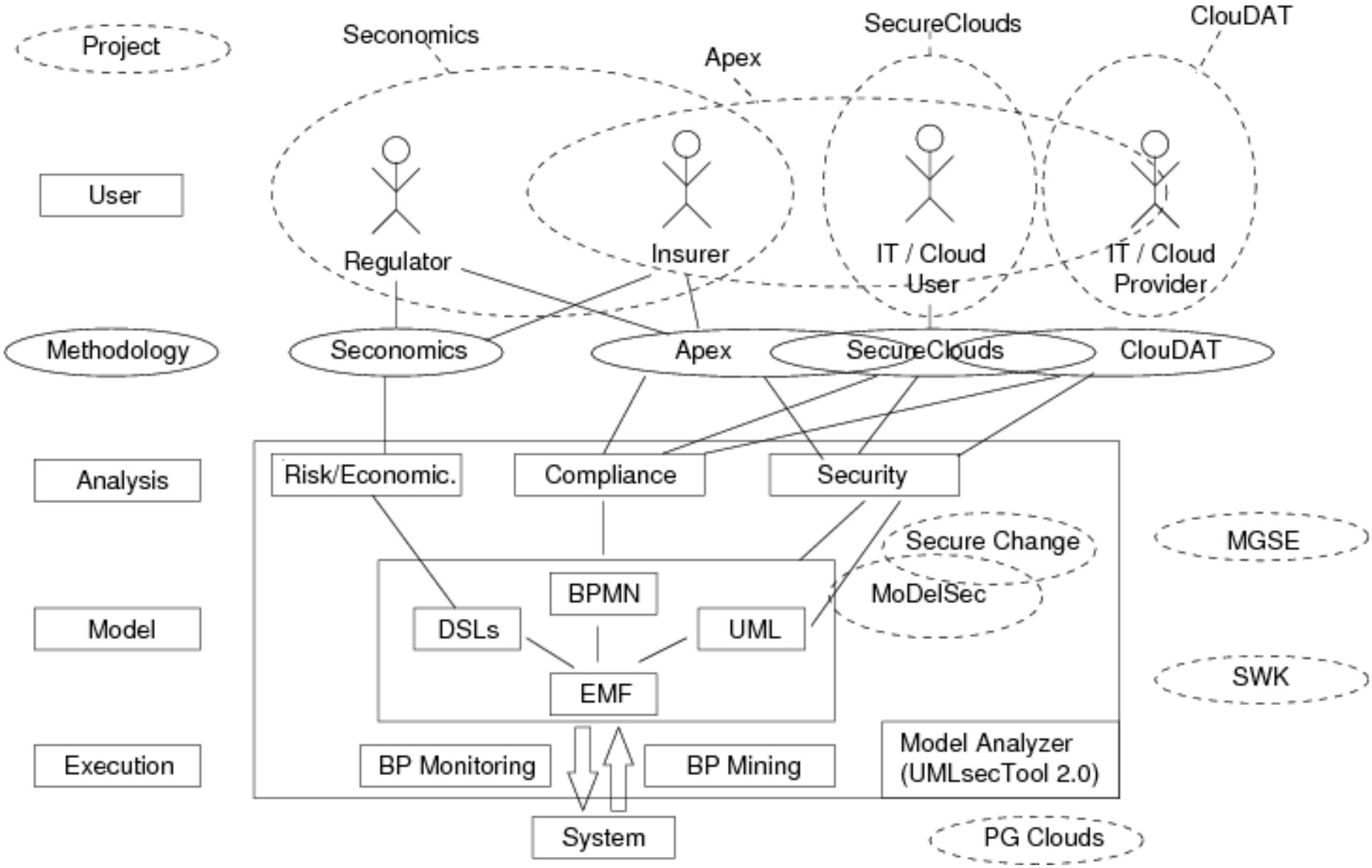
Companies are increasingly active in Model-based Security
(e.g. Interactive Objects, ObjectSecurity, Thales (Security DSML), Foundstone (McAfee), …)

2005: Model-based Security is part of the "Build Security In" body of knowledge of US Dep. Homeland Security.

2007: "Model-Driven Security: Enabling a Real-Time, Adaptive Security Infrastructure" (Gartner, 21 September 2007):

"Model-driven security is embryonic, but it will have a significant impact as information security infrastructures become increasingly real time, automated, and ***adaptive to changes*** in organizations and their environments." [http://www.gartner.com/DisplayDocument?id=525109]

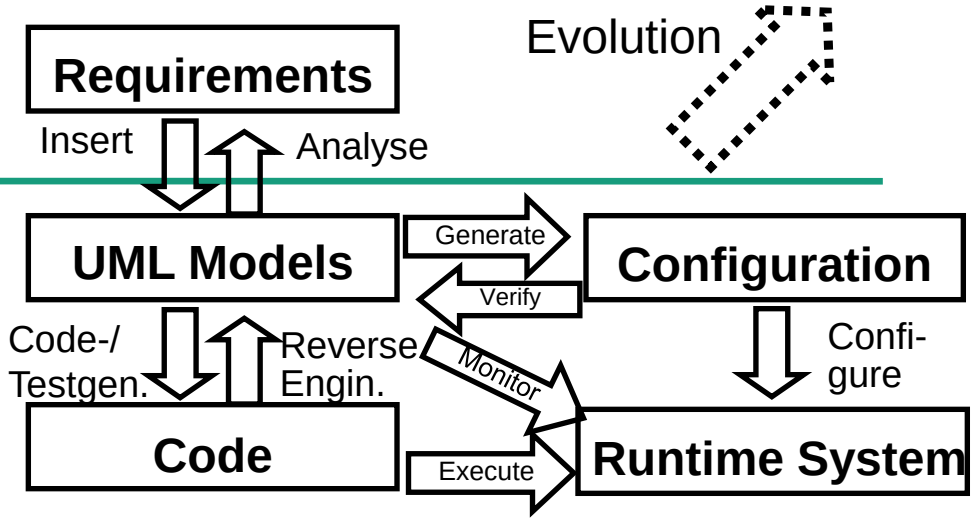
Some current projects



Model-based Security: Open Problems

- Secure evolution: first steps under way, more needs be done.
- Sound integration and pervasive verification:
  - between system abstraction hierarchies (applications down to hardware)
  - between system lifecycles phases
- General challenges in model-based development (general problems, but instantiated to security particularly interesting / challenging / important):
  - Scientific work: model repositories ?
  - Industrial usage: RoI of modelling ?
  - Usability and scalability of the modelling notations and associated tools ?
    - How to represent complex information (such as security information) within visual diagrams in an understandable and usable way ?

UMLsec: Summary



Model based security engineering with UMLsec:

- Model-based development with UML
- Automatic security analysis of software artifacts:
  - UML Models, Java / C programs, configuration data
- Successful applications in industry.

Questions?

More information (papers, slides, tools etc.): http://jan.jurjens.de