Tools and Techniques for Model-based Testing with UML

Jan Jürjens

Computing Department
The Open University

J.Jurjens@open.ac.uk

http://www.umlsec.org

Testing Critical and Embedded Systems

Very challenging.

For high level of assurance, would need full coverage (test every possible execution).

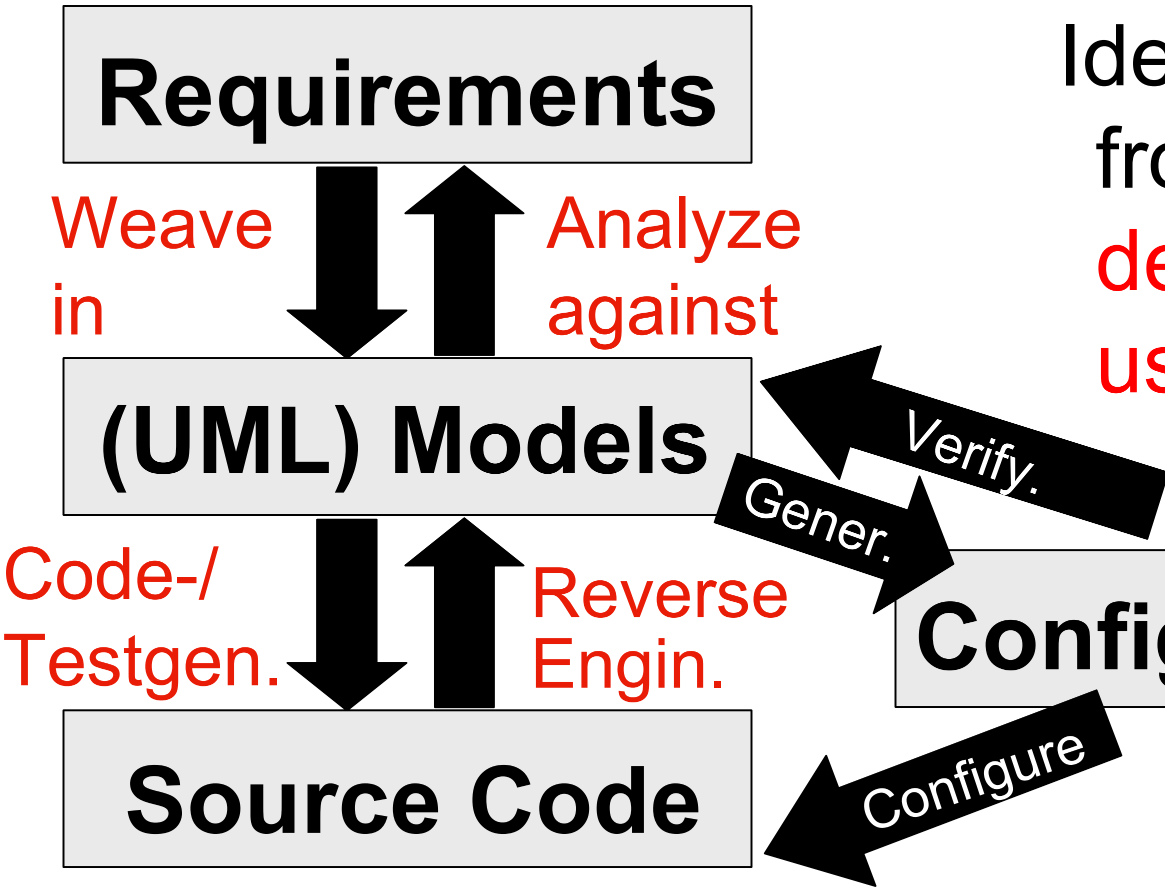
Usually infeasible (especially reactive systems).

Have heuristics for trade-off between development effort and reliability.

Need to ask yourself:

- How complete is the heuristic ?
- How can I validate it ?

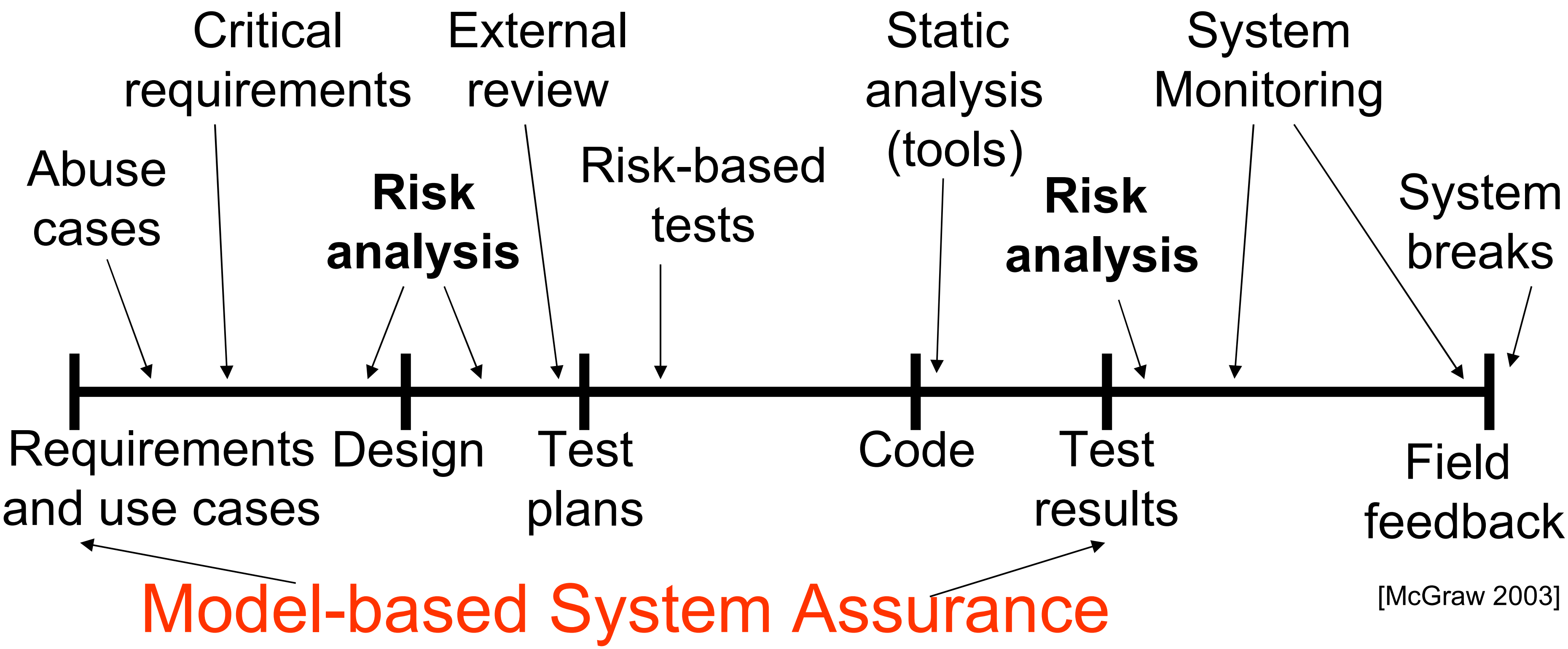
Model-based System Assurance



Idea: Extract models from artefacts in development and use of software.

➔ Tool-supported, theoretically sound, efficient automated design & analysis.

Critical System Lifecycle



Design: Encapsulate prudent engineering rules.

Analysis: Formally based, automated, efficient tools.

Note: emphasis on high-level requirements.

System Assurance: Model or Code ?

Model:

+ earlier (less expensive to fix flaws)

+ more abstract ➔ more efficient

- more abstract ➔ may miss flaws

- programmers may introduce flaws

- even code generators, if not formally verified

Code:

+ „the real thing“ (which is executed)

➔ Do both where feasible.

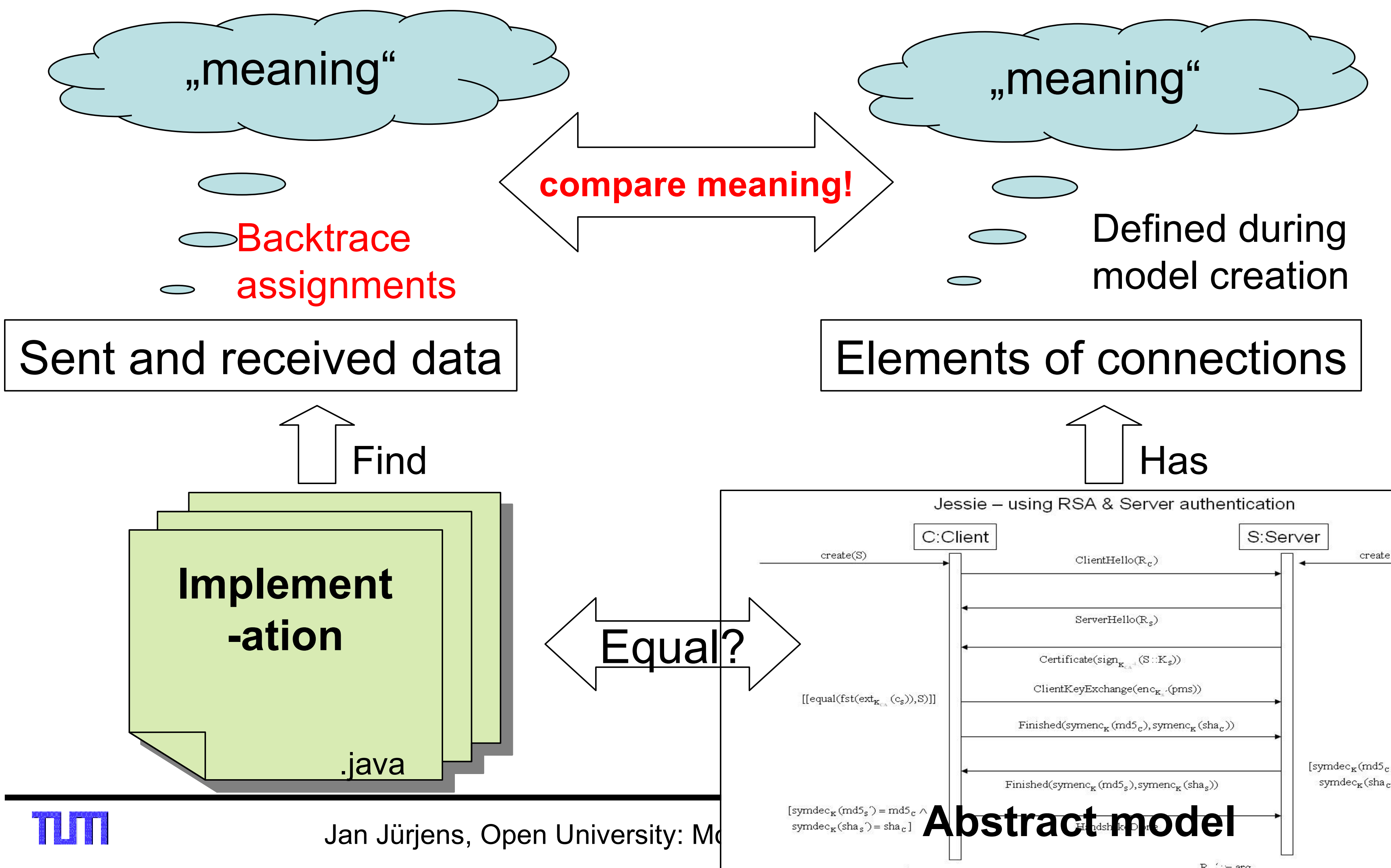
Verify Code against Models

Assumption: Have textual specification.
Then:

- construct interface spec from textual spec
- analyze interface spec for critical requirements
- verify that software satisfies interface spec

Model vs. Implementation

[with David Kirscheneder]



How to Verify Code Against Models

Model-based Testing (e.g. based on Real-time UML). Advantages:

- Precise measures for completeness.
- Can be formally validated.

Two complementary strategies:

- Conformance testing
- Testing for criticality requirements

Conformance Testing

Classical approach in model-based test-generation (much literature).

Can be superfluous when using code-generation [except to check your code-generator, but only once and for all].

Works independently of real-time requirements.

Conformance Testing: Caveats

- Complete test-coverage still infeasible (although can measure coverage).
- Can only test code against what is contained in model. Usually, model more abstract than code. May lead to „blind spots“.

For both reasons, may miss critical test-cases. Want: „criticality testing“.

Criticality Testing: Strategies

Internal: Ensure test-case selection from models does not miss critical cases: Select according to information on criticality.

External: Test code against possible environment interaction generated from parts of the model (e.g. deployment diagram with information on physical environment).

Criticality Testing

Shortcoming of classical model-based test-generation (conformance testing) motivates „criticality testing".

Goal: model-based test-generation adequate for critical real-time systems.

Internal Criticality Testing

Need behavioral semantics of used specification language (precise enough to be understood by a tool).

Here: semantics for simplified fragment of UML in „pseudo-code“ (ASMs).

Select test-cases according to criticality annotations in the class diagrams.

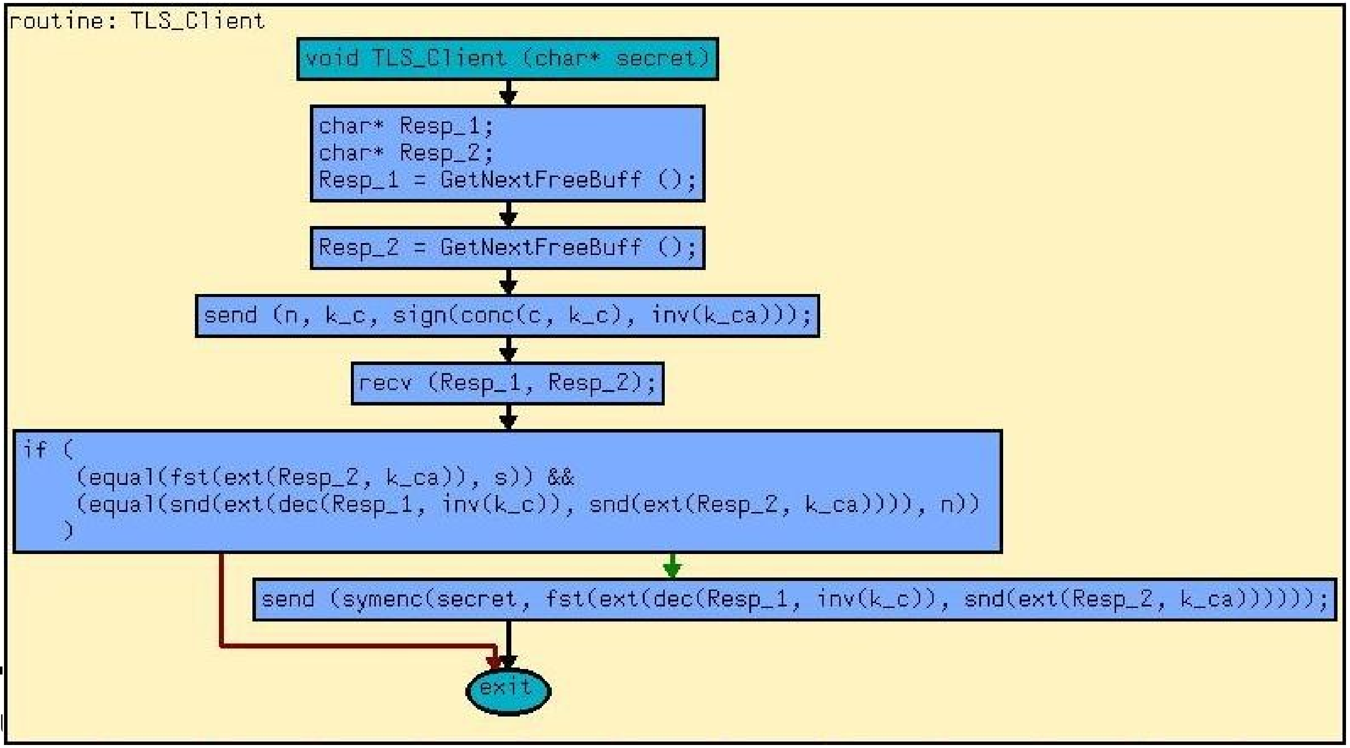
Test-cases: critical selections of intended behavior of the system.

External Criticality Testing

Generate test-sequences representing the environment behaviour from the criticality information in the deployment diagrams.

Automated White-Box Testing

- Generate control flow graph.
- Analyze for criticality requirements.
- Use to generate critical test-cases.



Model-based Testing with UML

Meaning of diagrams stated informally in (OMG 2003).

Ambiguities problem for

- tool support
- establishing behavioral properties (safety, security)

Need precise semantics for used part of UML, especially to ensure security requirements.

Tool-support: Pragmatics

Commercial modelling tools: so far mainly syntactic checks and code-generation.

Goal: sophisticated analysis. Solution:

- Draw UML models with editor.
- Save UML models as XMI (XML dialect).
- Connect to verification tools (automated theorem prover, model-checker …), e.g. using XMI Data Binding.

CSDUML Framework: Features

Framework for analysis plug-ins to access UML models on conceptual level over various UI's.

Exposes a set of commands. Has internal state (preserved between command calls).

Framework and analysis tools accessible and available at http://www.umlsec.org .

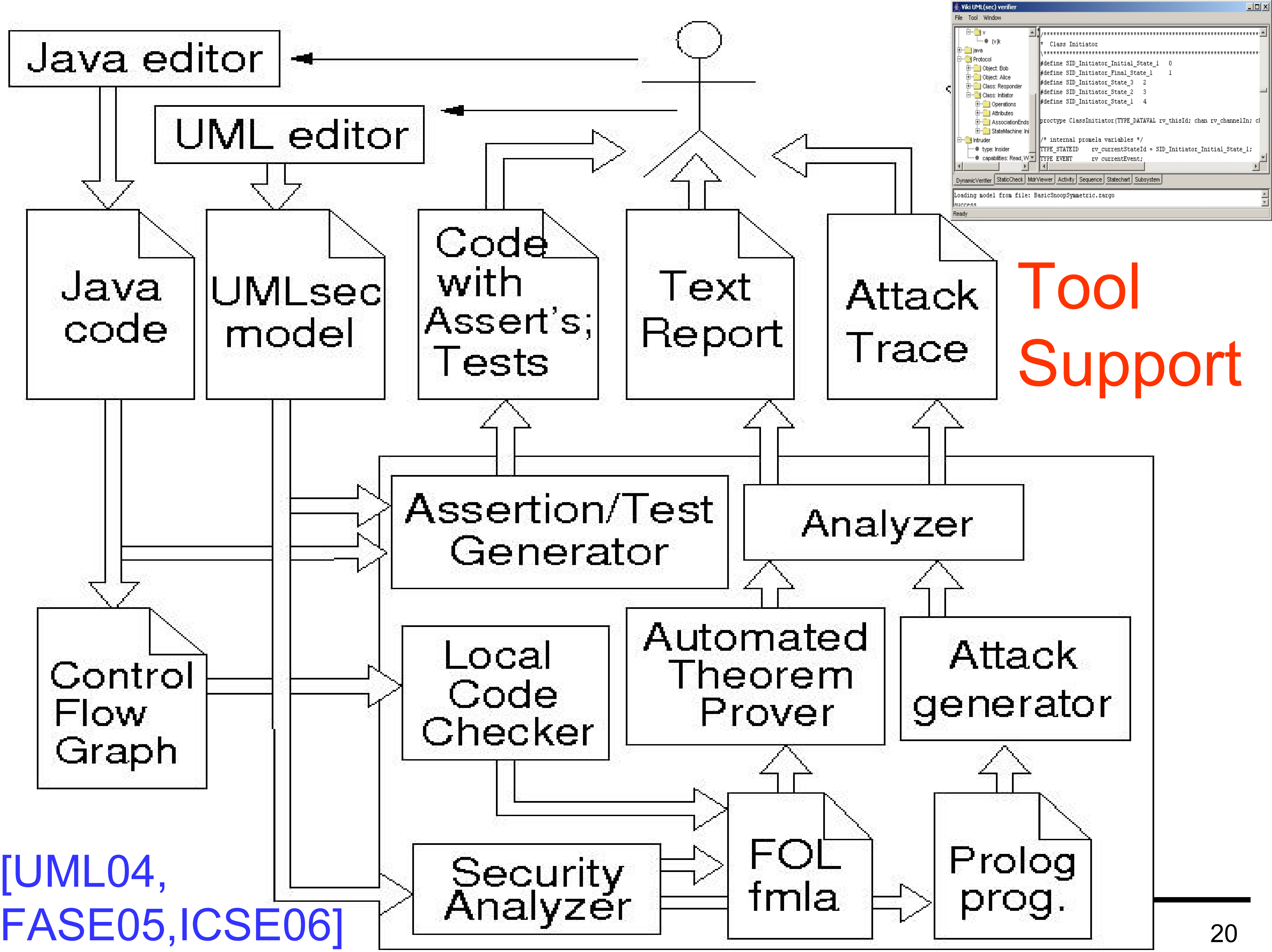
Upload UML model (as .xmi file) on website. Analyse model for included critical requirements. Download report and UML model with highlighted weaknesses.

Tool Support

For example:

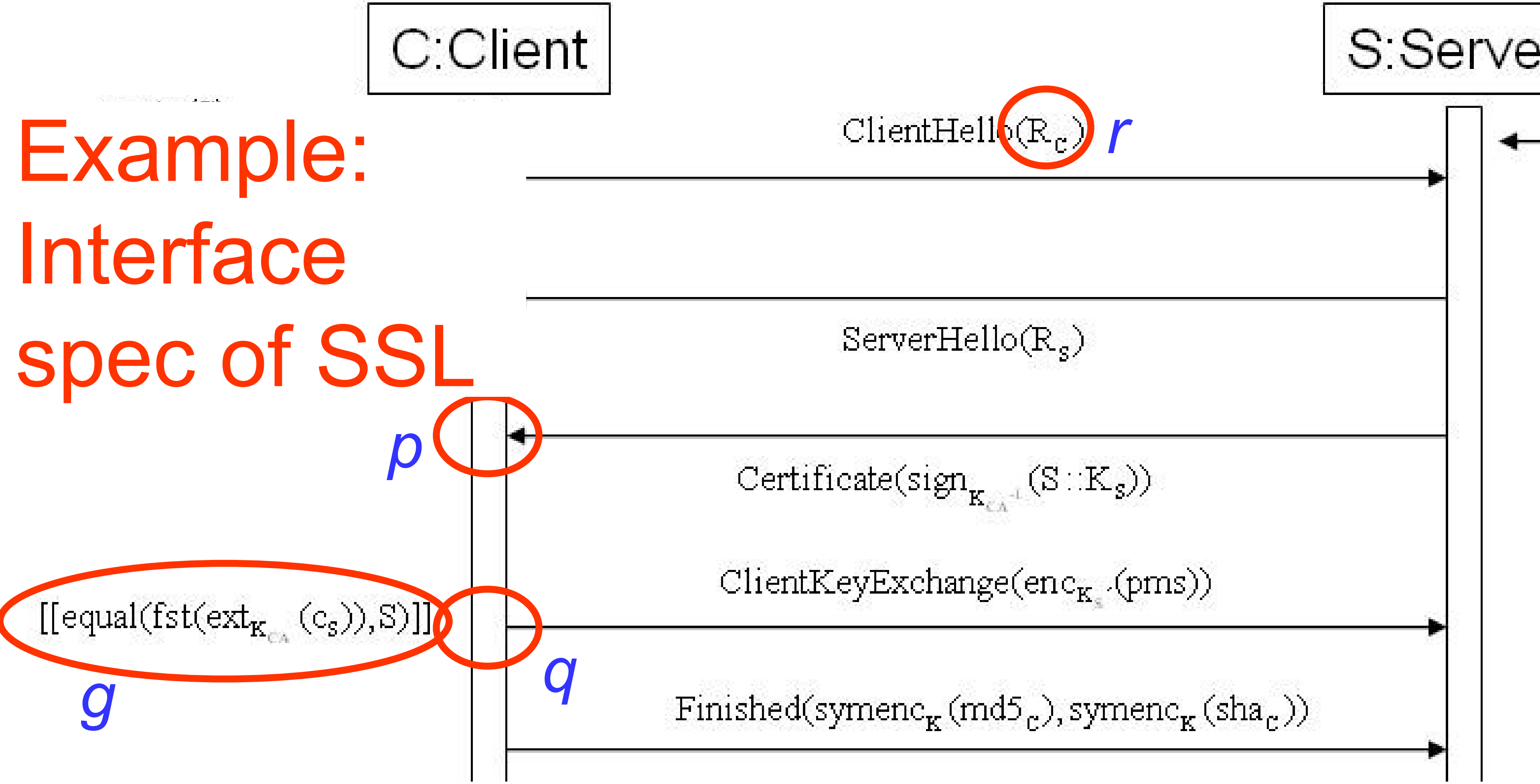
- consistency checks
- mechanical analysis of complicated requirements on model level (bindings to model-checkers, constraint solvers, automated theorem provers, …)
- code generation
- test-sequence generation
- configuration data analysis against UML.

Tool Support



[UML04, FASE05,ICSE06]

Example: Interface spec of SSL



I) Identify program points:

value (*r*), receive (*p*), guard (*g*), send (*q*)

II) Check guards enforced

Implementation of SSL: Identify Values

Parameter der kryptographischen ClientHello Nachricht	**Effektiv übertragene Daten der ClientHello Nachricht der Jessie Implementierung**
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

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

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

Identify: randomBytes

(in message *ClientHello*)

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

2nd parameter of ClientHello constructor

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

via Handshake.write()

initialized in SSLSocket.doClientHandshake()

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

initialization of the used Random object

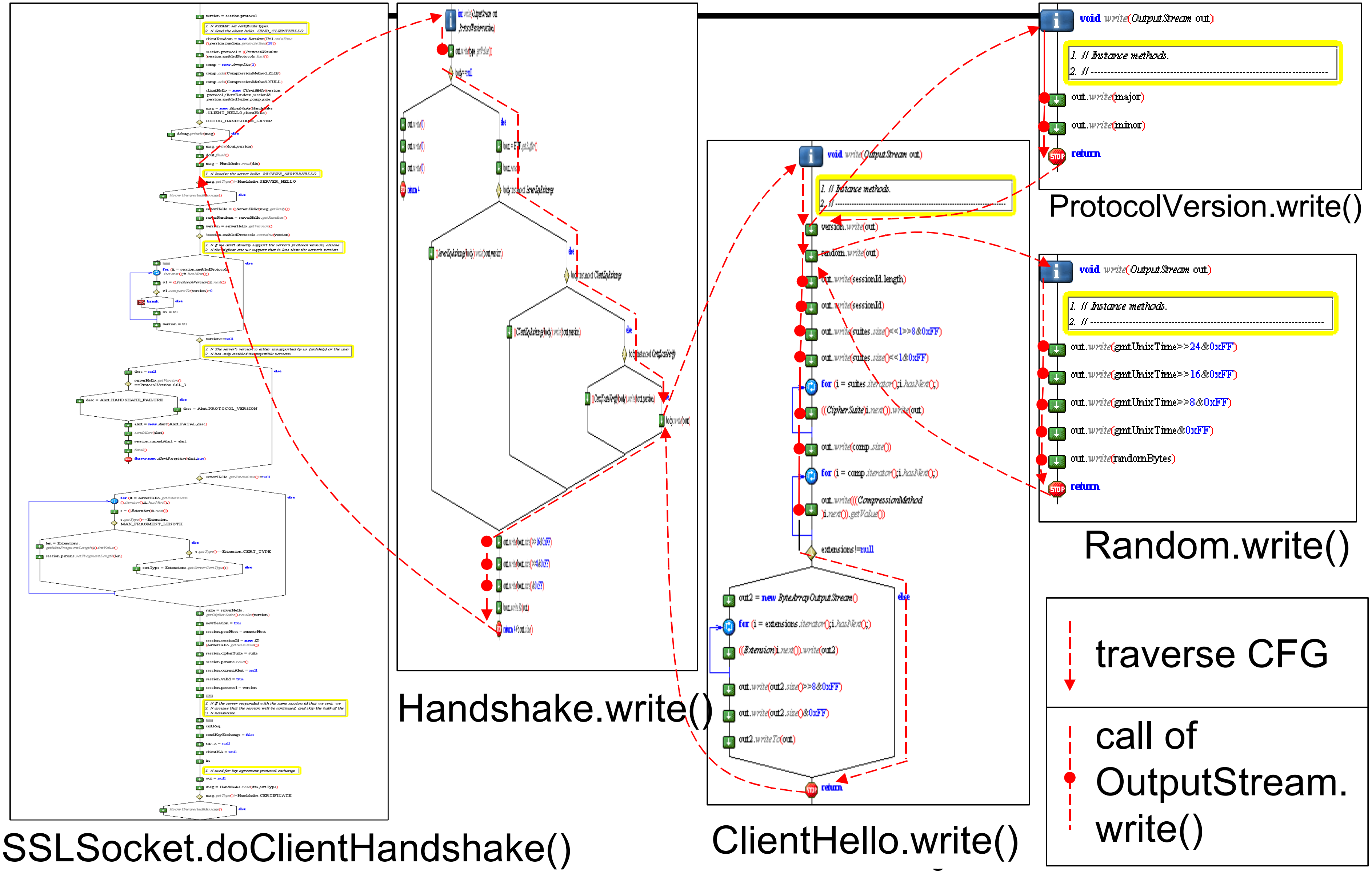
Random clientRandom =
new Random(...,**session.random.generateSeed(28)**);

„meaning“

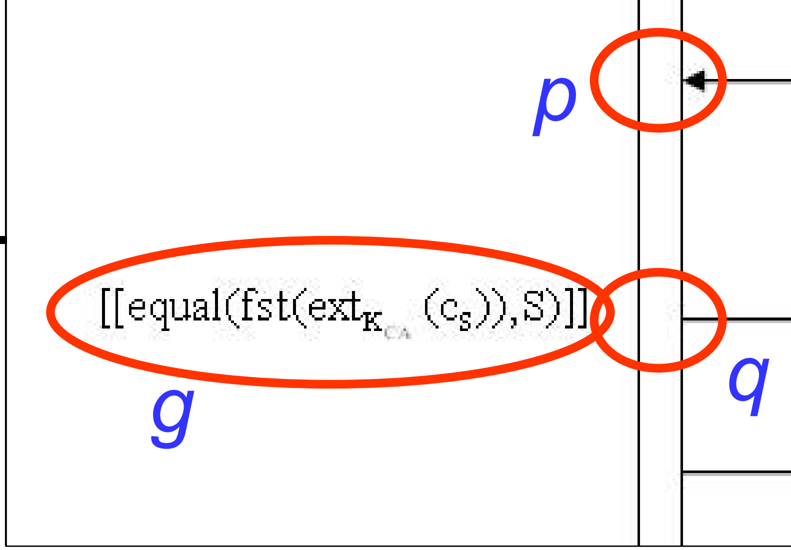
class SecureRandom (specified in: FIPS 140-2,RFC 1750) of package java.security
Function: generateSeed

Sending Messages

Automate this using patterns



Checking Guards



Guard *g* enforced by code?

b) Generate runtime check for *g* at *q* from diagram: simple + effective, but performance penalty.

c) Testing against checks (symbolic crypto for inequalities). [ICFEM02]

d) Automated formal local verification: conditionals between *p* and *q* logically imply *g* (using ATP for FOL). [ASE06]

```
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

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

calls doVerify()

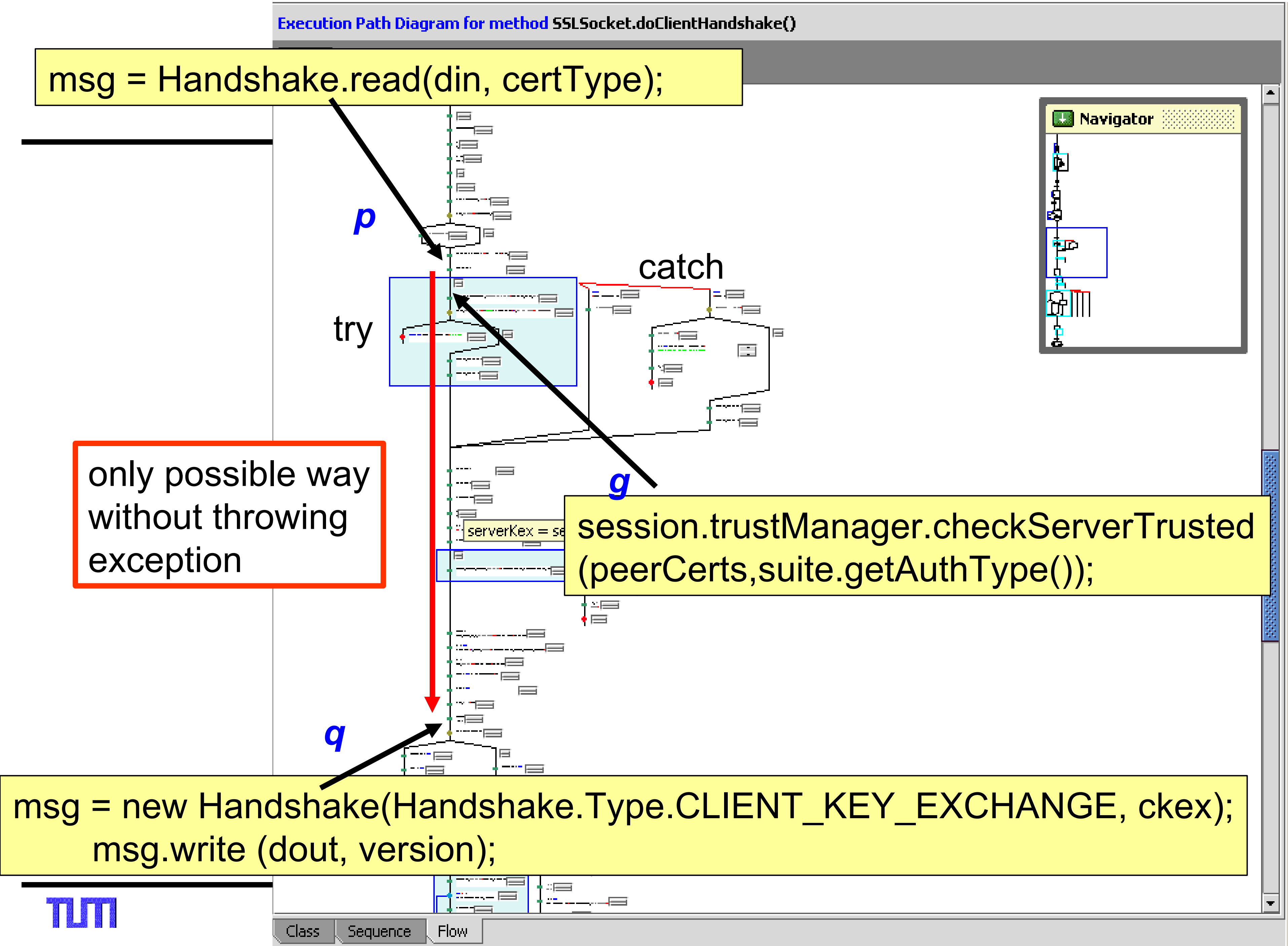
„meaning“

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"); ...   }  }
```


Execution Path Diagram for method SSLSocket.doClientHandshake()
msg = Handshake.read(din, certType);
p
catch
try
Navigator
only possible way without throwing exception
g
serverKex = se
session.trustManager.checkServerTrusted (peerCerts,suite.getAuthType());
q
msg = new Handshake(Handshake.Type.CLIENT_KEY_EXCHANGE, ckex);
msg.write (dout, version);
Class
Sequence
Flow

Some Applications

Analyzed designs / implementations / configurations for

- biometry, smart-card or RFID based identification
- authentication (crypto protocols)
- authorization (user permissions, e.g. SAP systems)

Analyzed security policies, e.g. for privacy regulations.

T··Systems·

Allianz

Deutsche Bank

HypoVereinsbank

CEPS™

BMW Group

msg systems

Münchener Rück
Munich Re Group

Bundesministerium
für Bildung
und Forschung

Bundesministerium
der Verteidigung

O2

Infineon

Bundesministerium
für Wirtschaft
und Technologie

Biometric Authentication System

In development by company in joint project.

Uses bio-reference template on smart-card.

Analyze given UML spec.

Discovered three major weaknesses in subsequently improved versions (misuse counter circumvented by dropping / replaying messages, smart-card insufficiently authenticated by mixing sessions). [ACSAC05]

Bank Application [SAFECOMP03]

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:

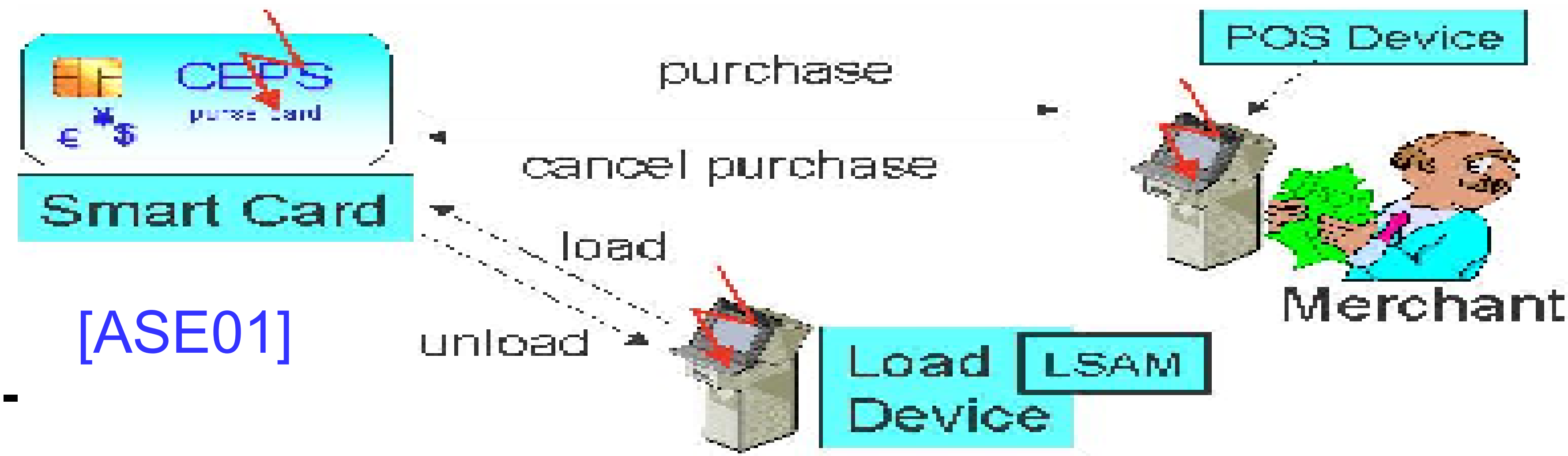
- confidentiality
- authenticity

Common Electronic Purse Specifications

Global elec. purse standard (Visa, 90% market).

Smart card contains account balance, performs crypto operations securing each transaction.

Formal analysis of load and purchase protocols: three significant weaknesses: purchase redirection, fraud bank vs. load device owner.



[ASE01]

Load Protocol

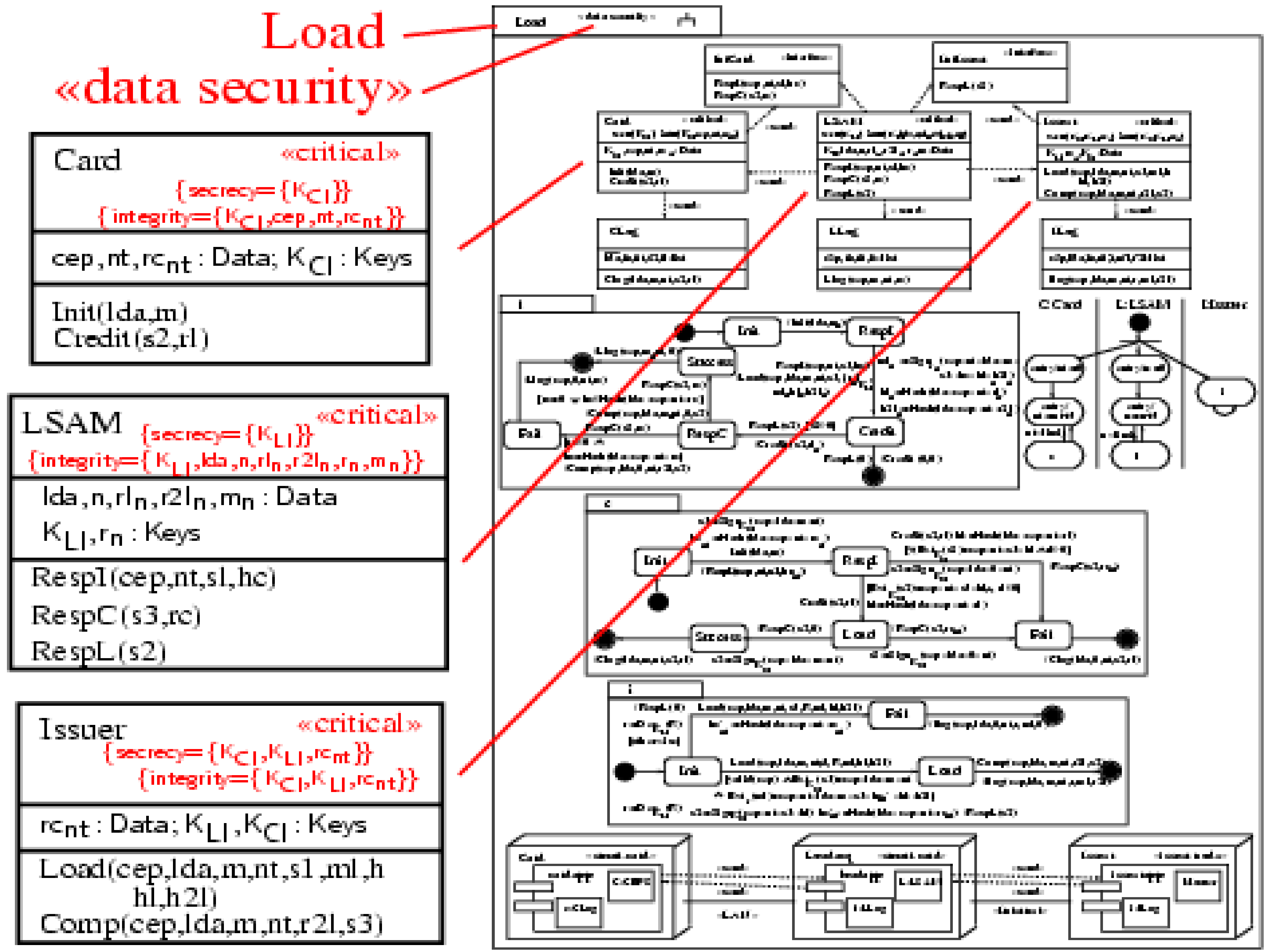
Unlinked, cash-based load transaction (on-line).

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)

Audit Security

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

Use post-transaction settlement scheme.

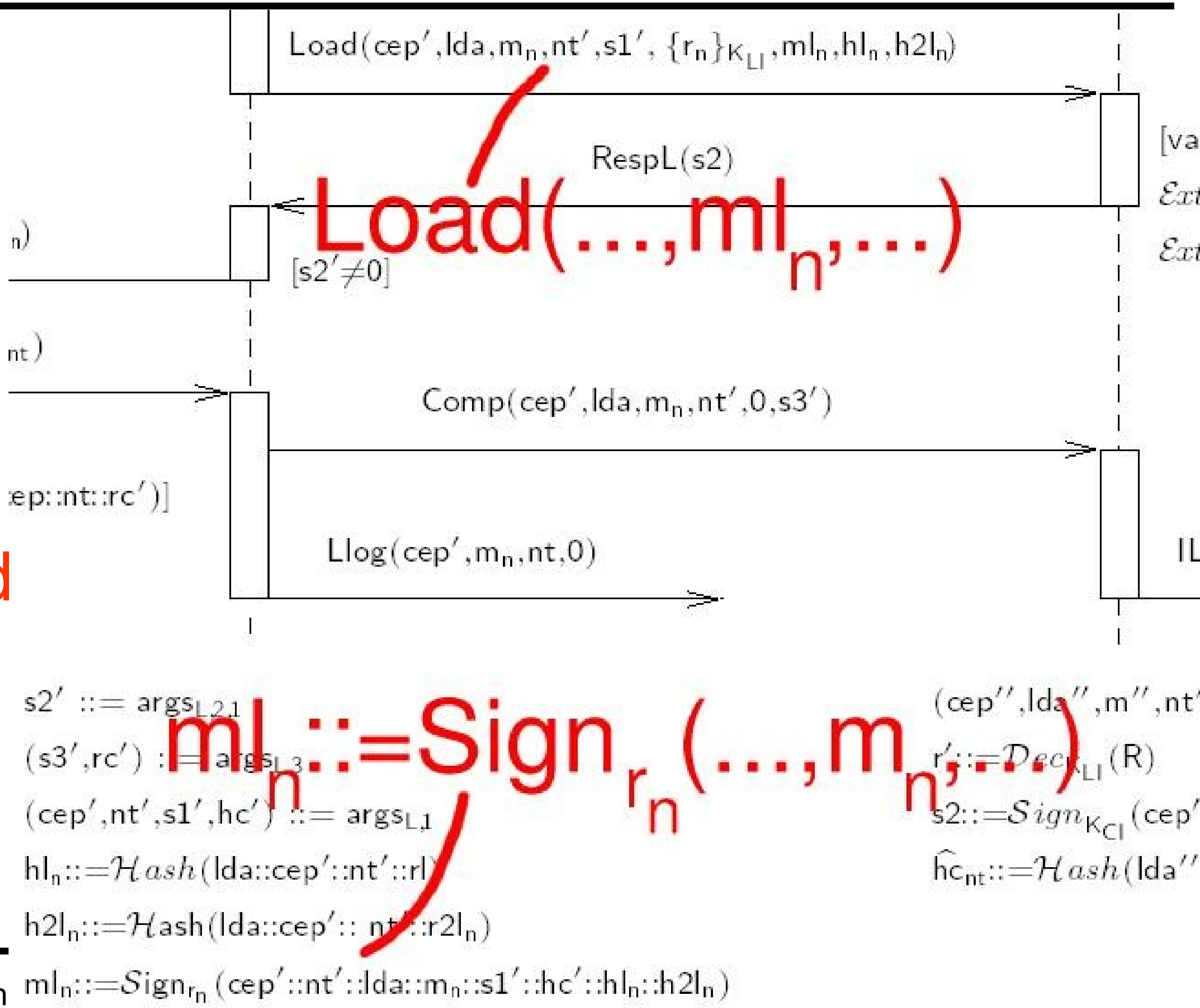
Relies on secure auditing.

Verify this here (only executions completed without exception).

Flaw I

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

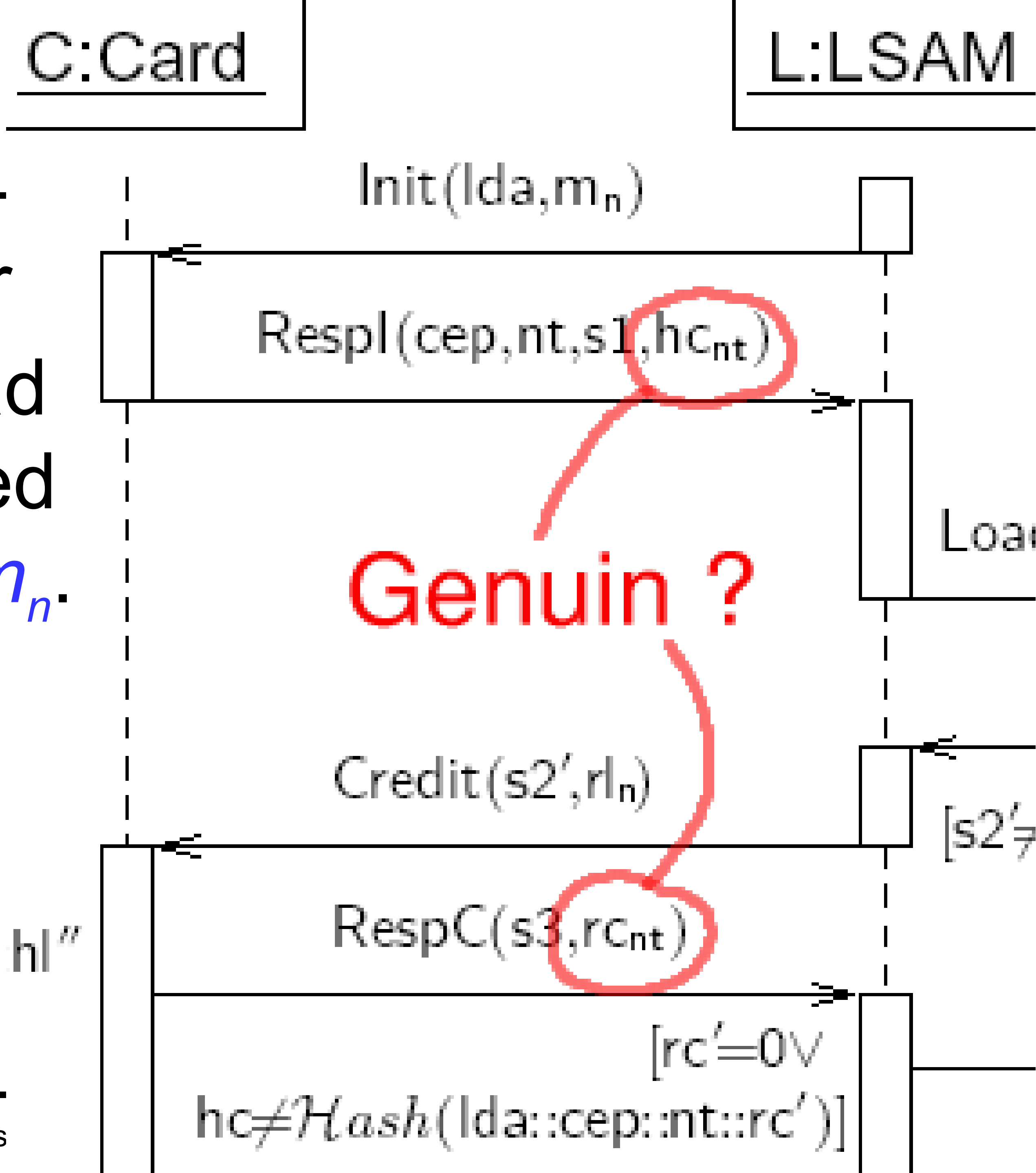
But: r_n shared between bank and load machine.



Flaw II

rc_{nt}: „Proof" for LSAM that load device received only amount m_n.

But: LSAM cannot prove validity of rc_{nt}.

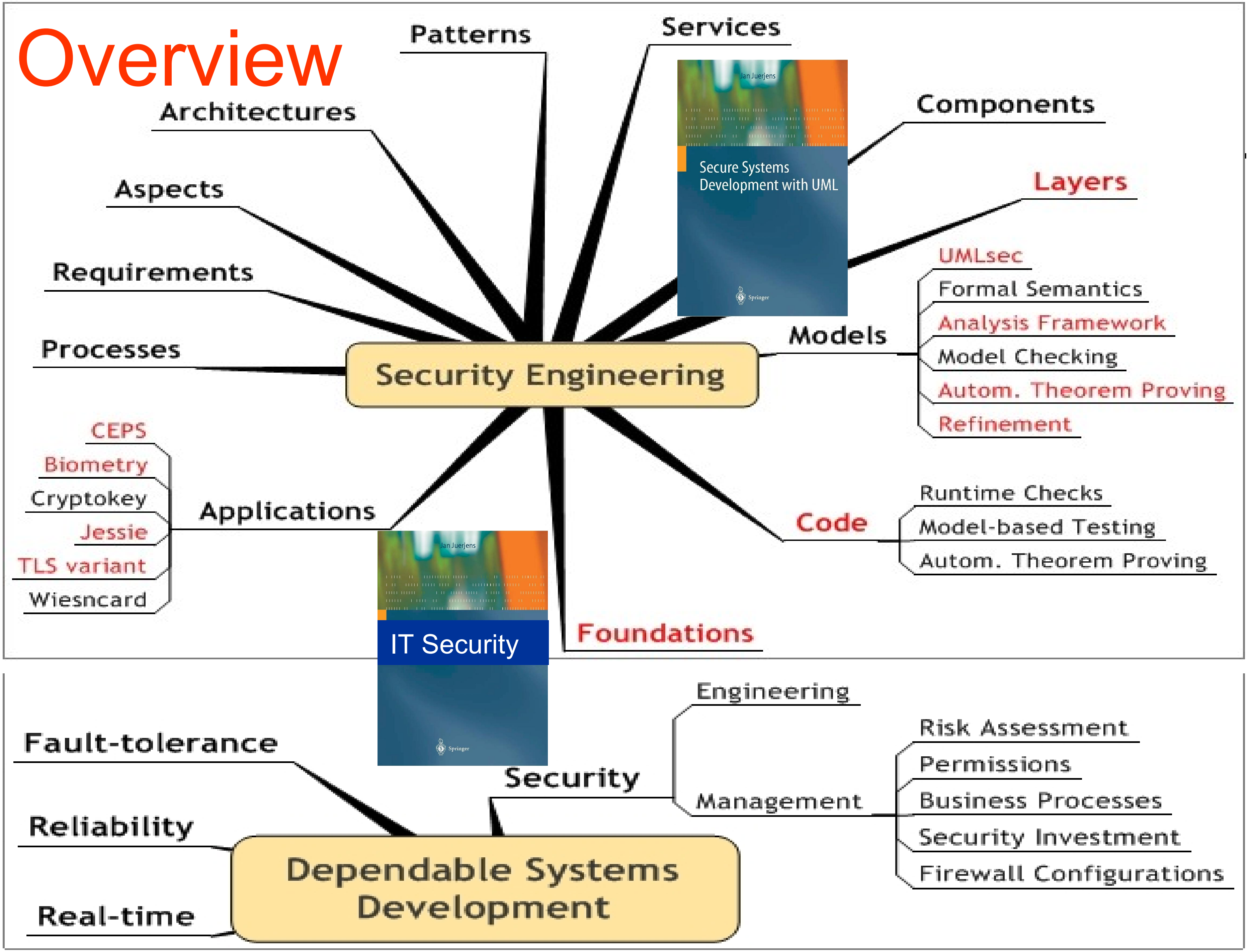


Conclusions

Model-based Testing using UMLsec:

- formally based approach
- automated tool support
- industrially used methods
- integrated approach (source-code, configuration data)

Overview



Patterns
Services
Architectures
Components
Aspects
Layers
Requirements
UMLsec
Formal Semantics
Analysis Framework
Processes
Models
Model Checking
Security Engineering
Autom. Theorem Proving
Refinement
CEPS
Biometry
Cryptokey
Applications
Jessie
TLS variant
Wiesncard
Runtime Checks
Code
Model-based Testing
Autom. Theorem Proving
Foundations
Jan Juerjens
Secure Systems Development with UML
Springer
Jan Juerjens
IT Security
Springer
Engineering
Fault-tolerance
Risk Assessment
Permissions
Security
Management
Business Processes
Reliability
Security Investment
Firewall Configurations
Dependable Systems Development
Real-time

Questions ?

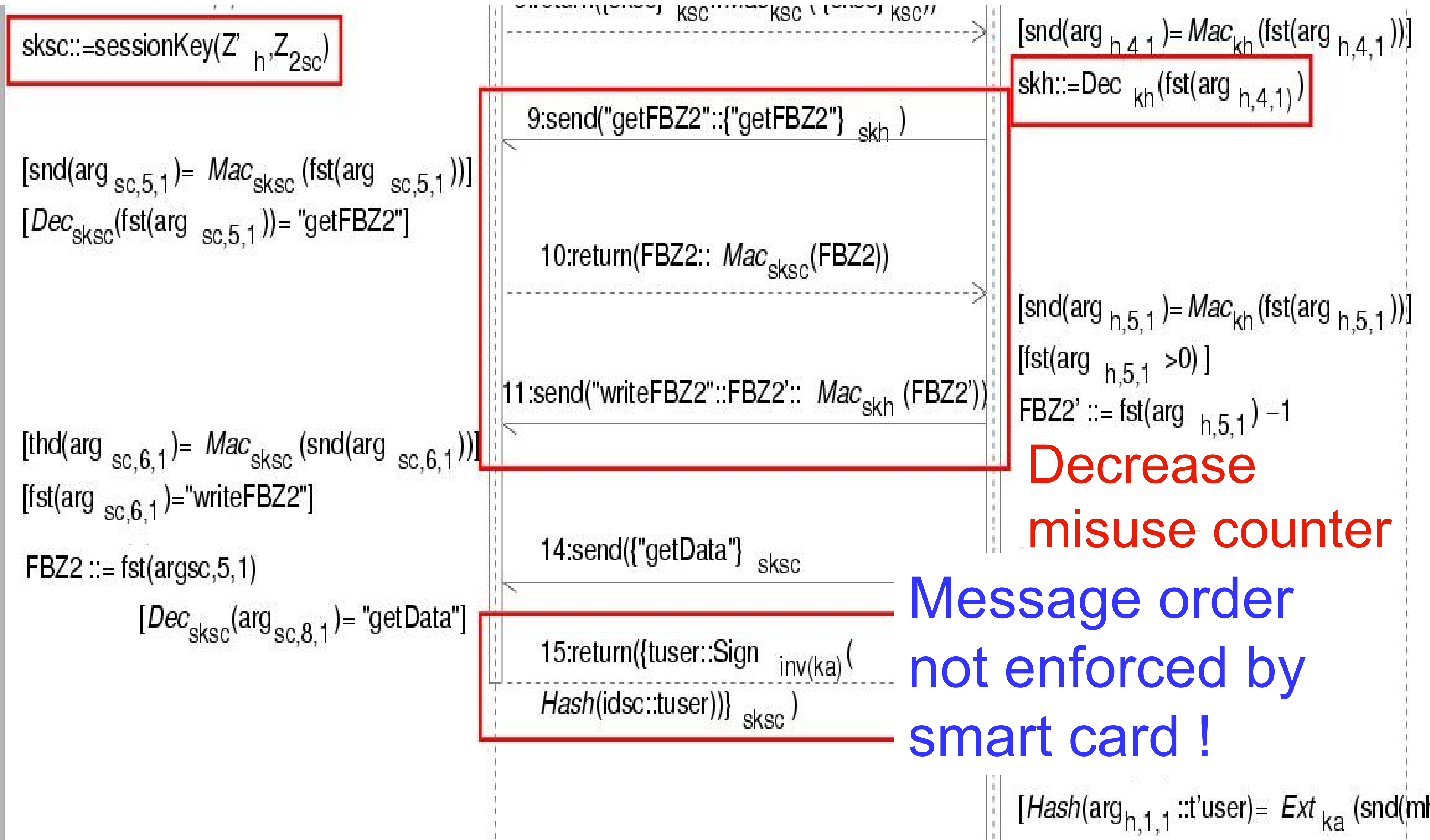
More information (papers, slides, tool etc.):

http://www.umlsec.org

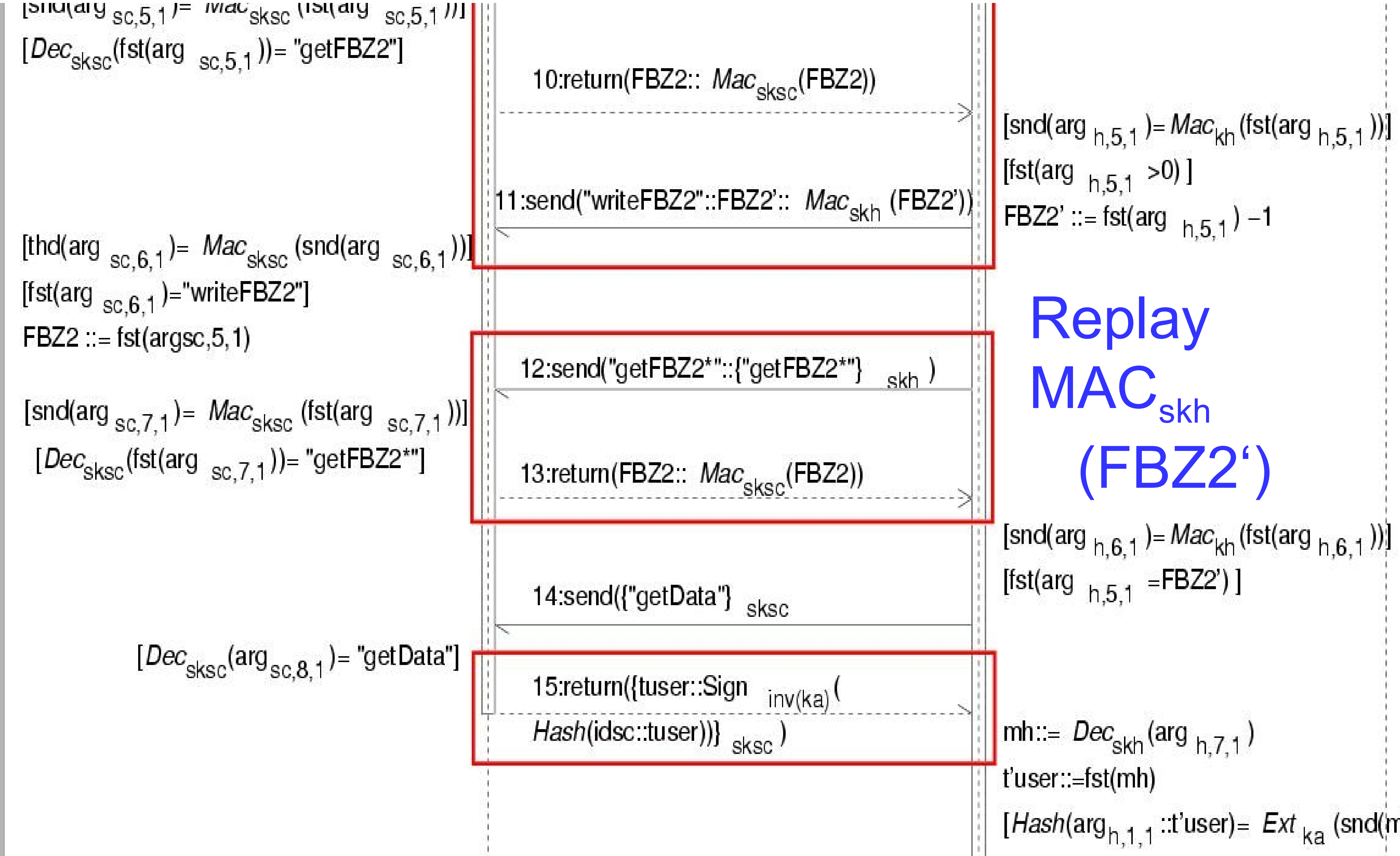
J.Jurjens@open.ac.uk

Backup

Authent. Protocol Pt. 2: Problem ?



Authent. Prot. Pt. 2: Problem



Replay MAC_{skh} (FBZ2‘)

Authentic. Protocol Part 1: Problem.