Model-based Security Testing using UMLsec

Jan Jürjens

Computing Department
The Open University, GB

J.Jurjens@open.ac.uk

http://www.jurjens.de/jan

Crypto-Protocol Analysis

State of the affairs:

A *lot* of very successful work in formally verifying abstract models of crypto-protocol design.

- virtually every formal method has been applied
- seemingly more people working on verification than on designing protocols
- efficient tool-support usable by academics or specialists
- sometimes used at industrial size protocols (usually by tool developers themselves)

(Almost) solves the problem whether design is secure.

Problem

How do I know a crypto-protocol implementation is secure ?

Possible solution:

Verify design model, 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

Towards Verifying Legacy Implementations

Goal: Verify implementation created independently.

Options:

3) 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.

2) Create models and code manually and verify code against models.

- Advantages: Split heavy verification burden. Get some verification result already in design phase (for non-legacy implementations).

Testing Security-Critical Systems

Very challenging.

Given motivated adversary, would need full coverage (test every possible execution).

Usually infeasible (especially open systems).

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

Questions:

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

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

➔ How do you find these weaknesses using classical testing ?

(Yes, you don’t.)

Problem: Untrustworthy Programmer

- For security assurance, may not even trust the programmer of the code.
- May have intentionally built in back-door into code.
- May be impossible to find by random or black-box testing (e.g. hard-coded special password).
- Even worse when elusive weaknesses are used (previous slide).

➔ What are the precautions in practice ?

(Mostly none.)

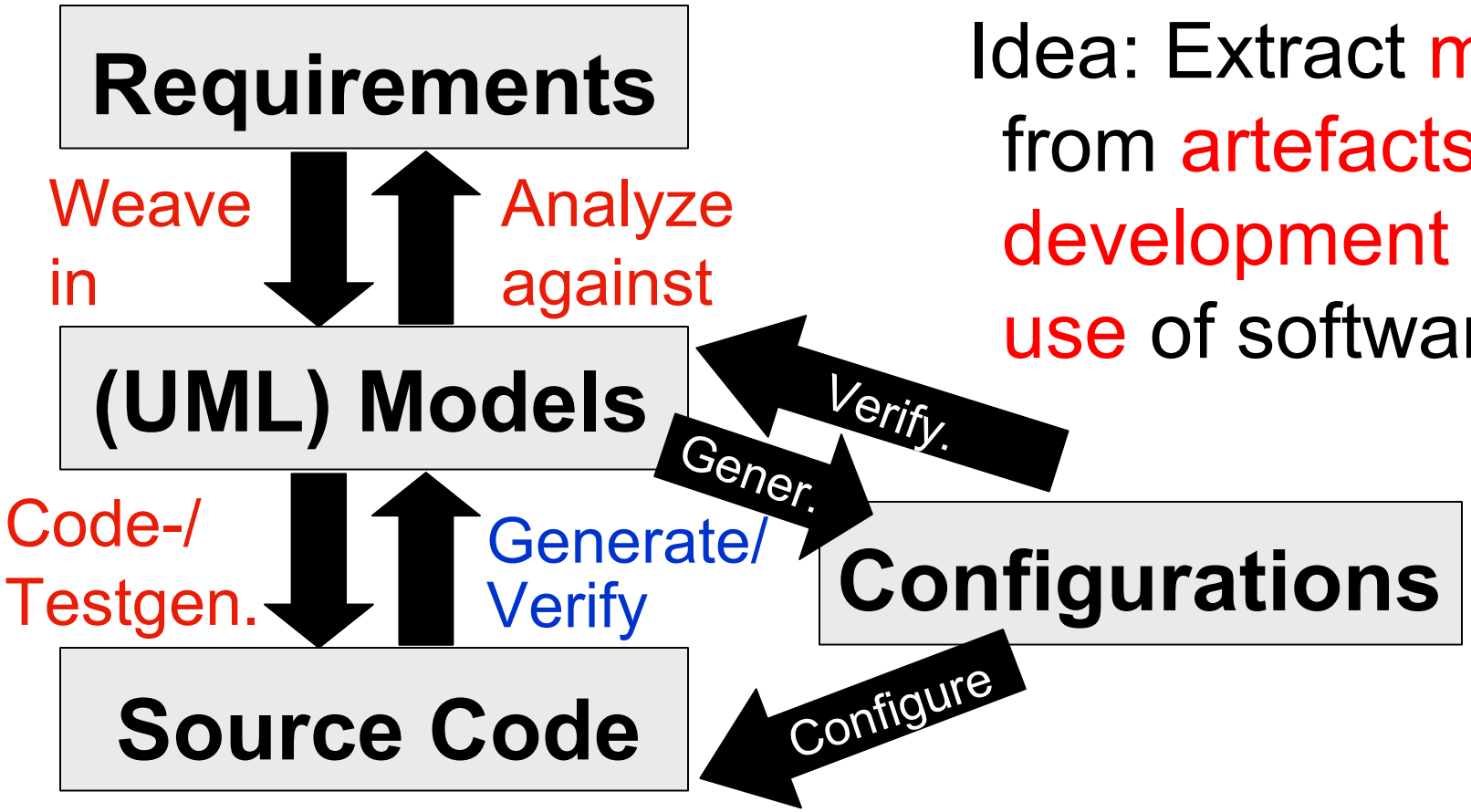
Special Problem: Crypto

- 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 ?

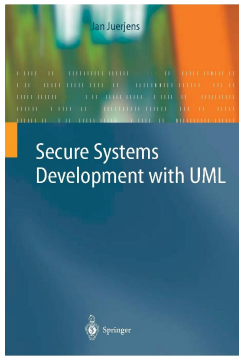
➔ What`s your answer ?

(Not using conventional testing.)

Background: Model-based Security Engineering

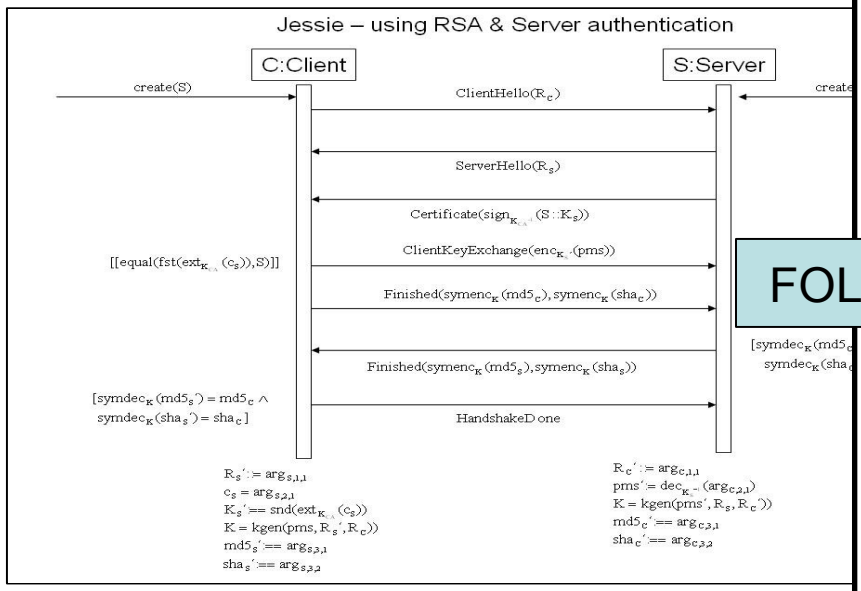


Idea: Extract models from artefacts in development and use of software.



➔ Long-term goal: Tool-supported, theoretically sound, efficient automated security design & analysis.

Model Verification



FOL

```
...
((
  knows(ArgC_3)
  & (equal(fst(ArgC_3), type_serverkeyexchange))
  & (equal(snd(ext(snd(snd(ArgC_3)), k_ca)), skey))
  & (equal(snd(ext(snd(ArgC_2), k_ca)), fst(snd(ArgC_3))))
)
=>(
  ((knows(ArgC_4_1)
    & equal(ArgC_4_1, type_serverhellodone))
    =>(
  ( ( true & equal(ClientKeyExchange, enc(premasterkey, skey)
)
...
%------------------------ Conjecture --

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

Check whether can derive *knows(s)*.

If yes, generate attack scenario.

If no, *s* secret (wrt our attacker).

ATP

```
analyzing results ...
model found/total failure
time limit information: 19 total / 18 strategy
(leaving wrapper).
task myUML_PID1491 on atbroy1 has status SUCCESS
(model found by strategy 300) consuming 1 seconds
deleting temporary files.
e-SETHEO done. exiting
```

Just an Exercise in Code Verification ?

State of the art in practical code verification: execution exploration by testing (possibly generated from models). 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.

General approaches to formal code verification exist (Isabelle et al), but limited use by (civilian) software engineers, and usually not for sophisticated properties like Dolev-Yao security.

➔ Develop specialized verification approach.

Model vs. Implementation

Sent / received data

Automated verification

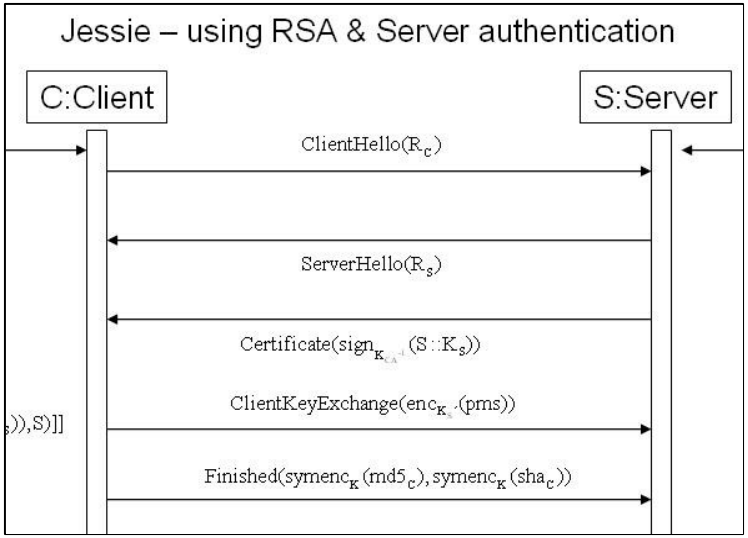
Test cases

Trace

Generate

Implemen-tation

Consistent?



in Model	Send: ClientHello	by Outputstream.write in
	type.getValue()	Handshake.write
	(bout.size() >>> 16 & 0xFF)	Handshake.write
	(bout.size() >>> 8 & 0xFF)	Handshake.write
	(bout.size() & 0xFF)	Handshake.write
Pver	major	ProtocolVersion.write
	minor	ProtocolVersion.write
	((gmtUnixTime >>> 24) & 0xFF)	Random.write
	((gmtUnixTime >>> 16) & 0xFF)	Random.write
	((gmtUnixTime >>> 8) & 0xFF)	Random.write
	(gmtUnixTime & 0xFF)	Random.write
R_C	randomBytes	ClientHello.write
	sessionId.length	ClientHello.write
Sid	sessionId	ClientHello.write
	((suites.size() << 1) >>> 8 & 0xFF)	ClientHello.write
	((suites.size() << 1) & 0xFF)	ClientHello.write
Ciph[]	id[]	CipherSuite.write
	comp.size()	ClientHello.write
Comp[]	comp[2]	ClientHello.write

Data vs. Symbols (SSL project Jessie)

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

Identify: randomBytes

(in message *ClientHello*)

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

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**,...);

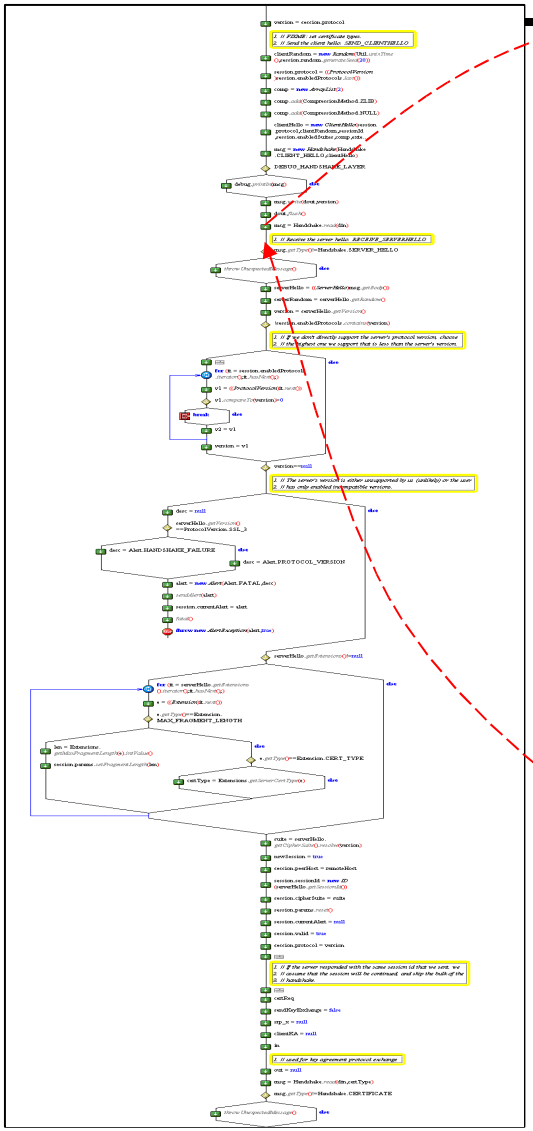
initialization of the used Random object

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

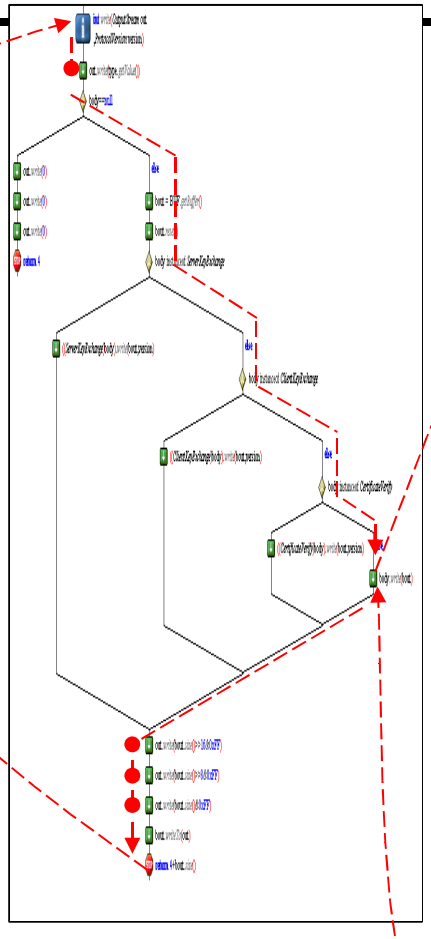
„meaning“

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

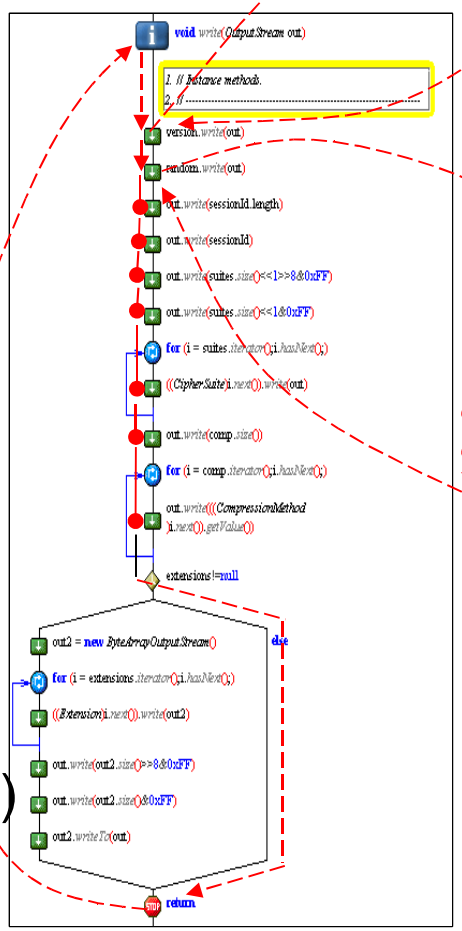
Sending Messages



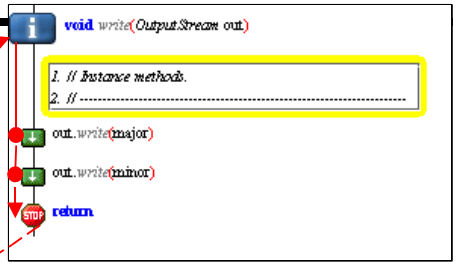
SSLSocket.doClientHandshake()



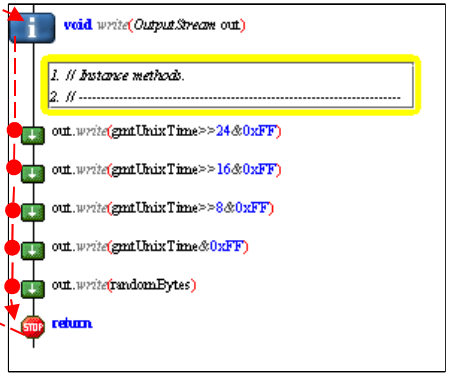
Handshake.write()



ClientHello.write()



ProtocolVersion.write()



Random.write()

traverse CFG

call of OutputStream.write()

```
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"); …   }  }
```

Model-based Security Testing: Strategies

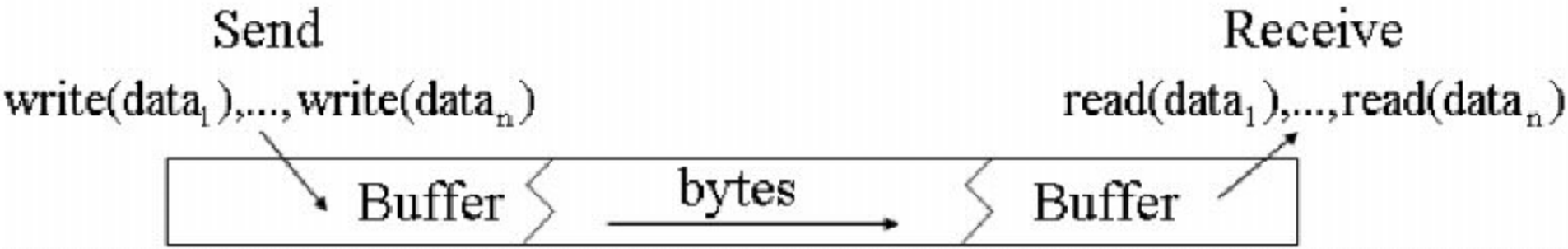
Internal: Ensure test-case selection from models does not miss critical cases: Select according to information on criticality / security.

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

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).



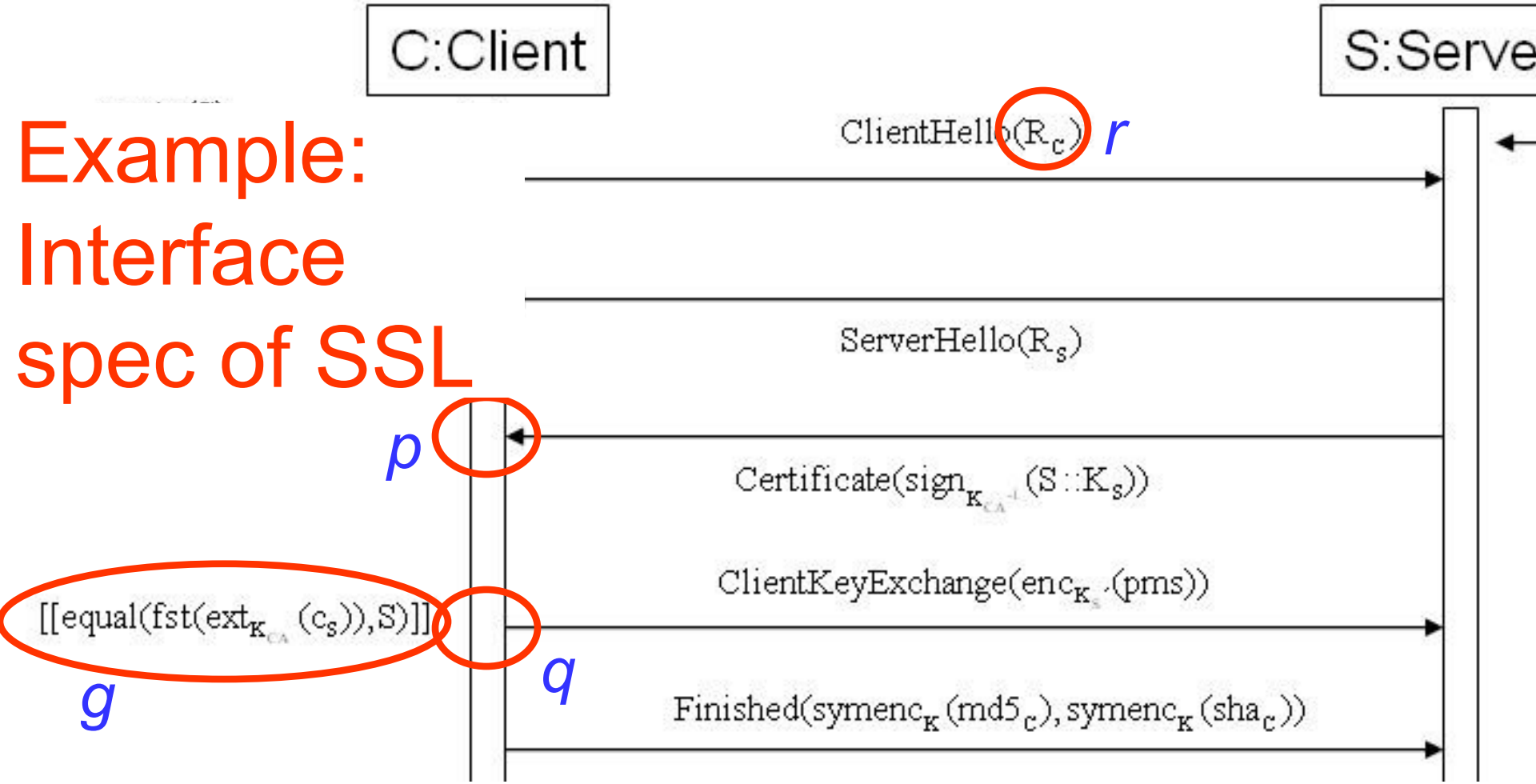
Example

Sending a protocol message (e.g. ClientHello):

- create the clientHello object with appropriate message parameters
- create the message object msg by giving the clientHello object as an argument
- call the write method at the msg object

```
ClientHello clientHello = new ClientHello(session.protocol,clientRandom,sessionId,
                                          session.enabledSuites,comp, extensions);
Handshake msg = new Handshake(Handshake.Type.CLIENT_HELLO, clientHello);
msg.write (dout, version);
```

Example: Interface spec of SSL



I) Identify program points:

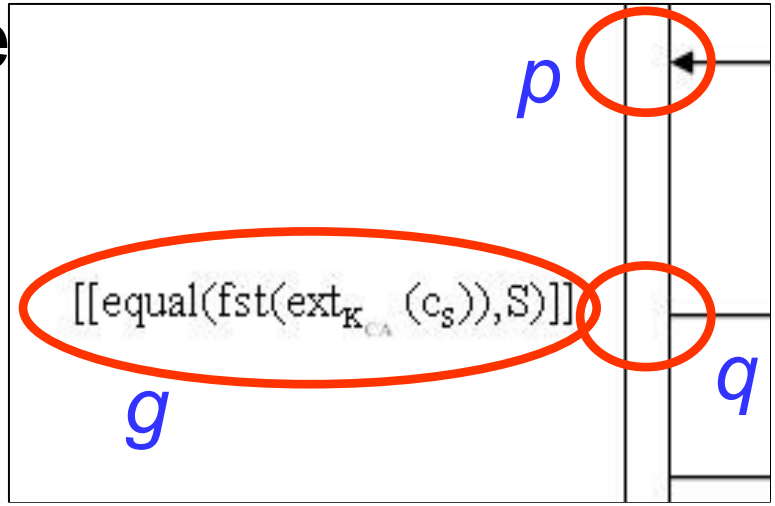
value (*r*), receive (*p*), guard (*g*), send (*q*)

II) Check guards enforced

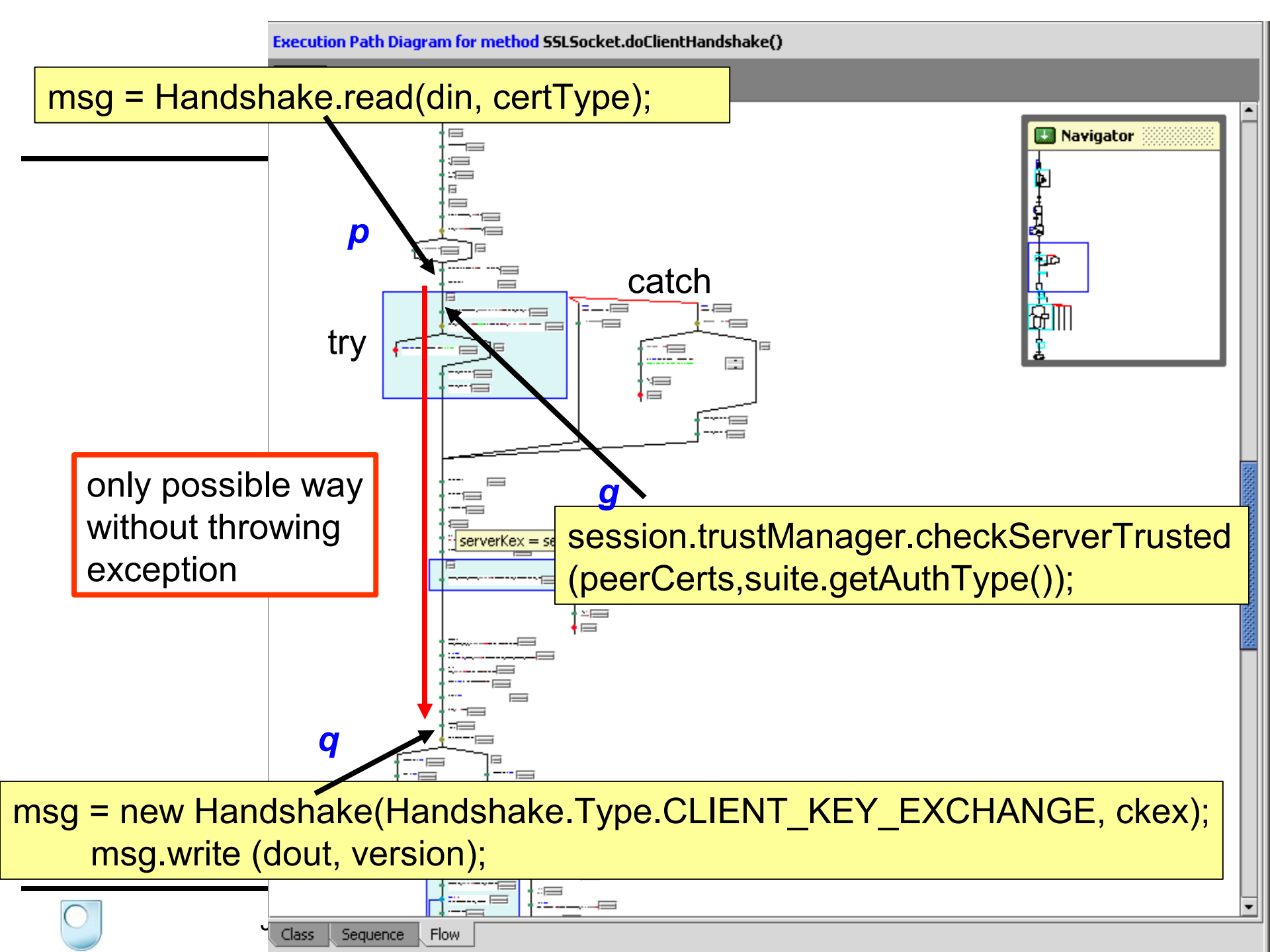
Checking Guards

Guard *g* enforced by code

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

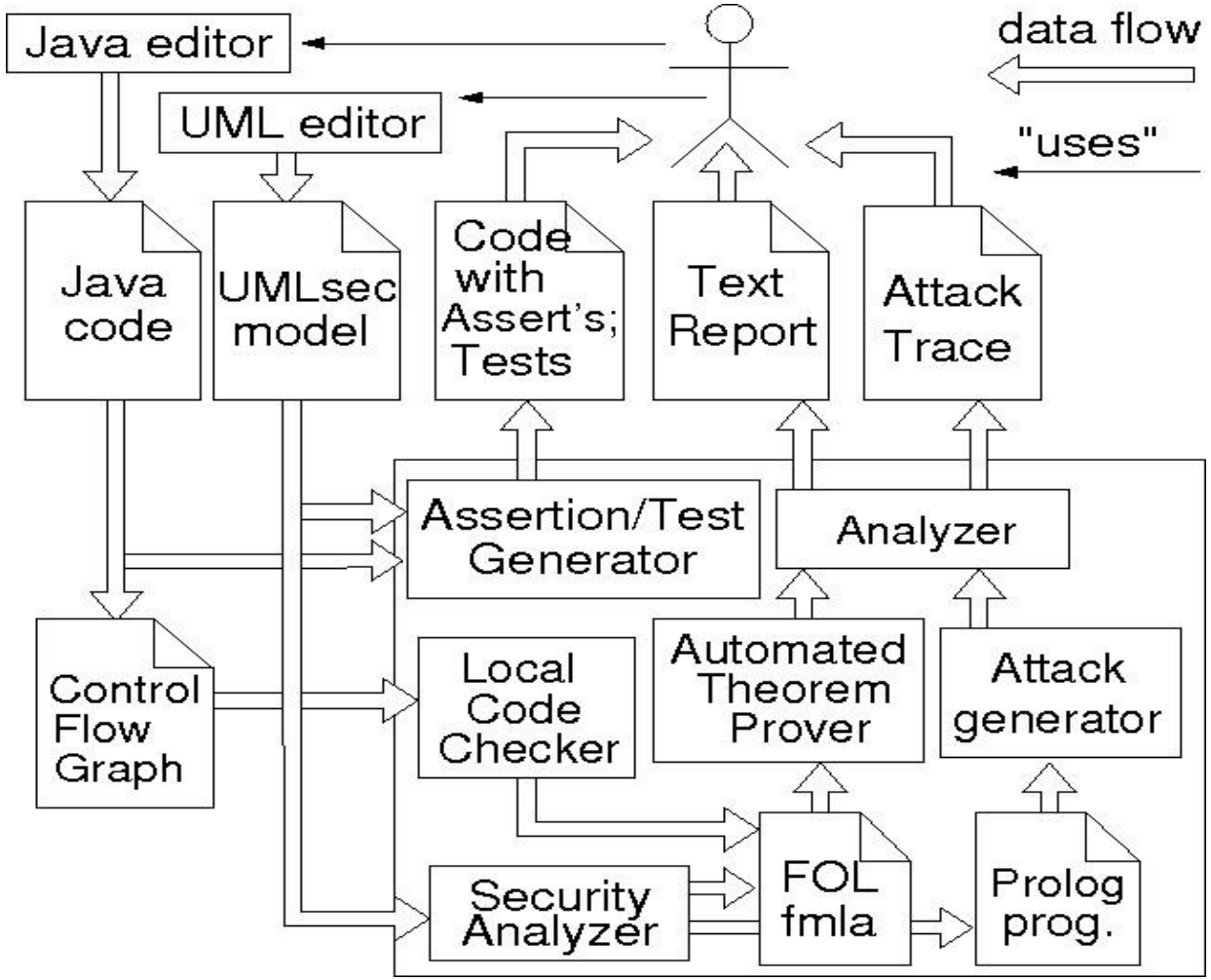


➔Testing against checks (symbolic abstractions for crypto).


Execution Path Diagram for method SSLSocket.doClientHandshake()
msg = Handshake.read(din, certType);
p
try
catch
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

Tool Support

[FASE05,ICSE06,ASE07, STTT07,ICSE08]



Also:

- configuration analysis: (user permissions, firewall rules/ policies) [FASE08]
- code traceability (with Yijun Yu)

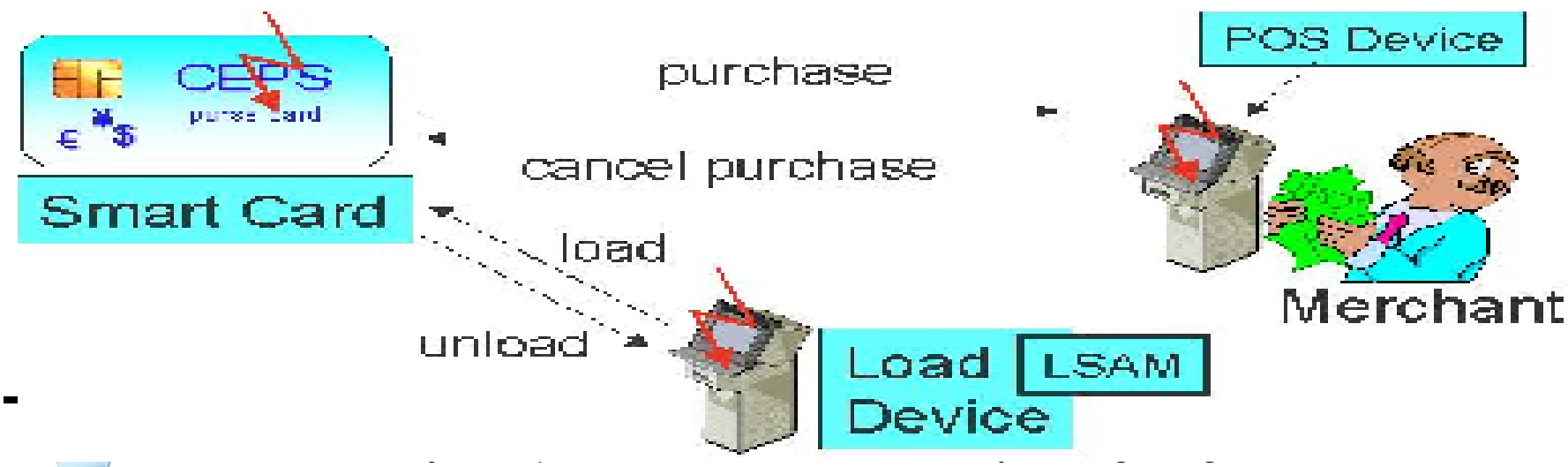
Open-source

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.



CEPS 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

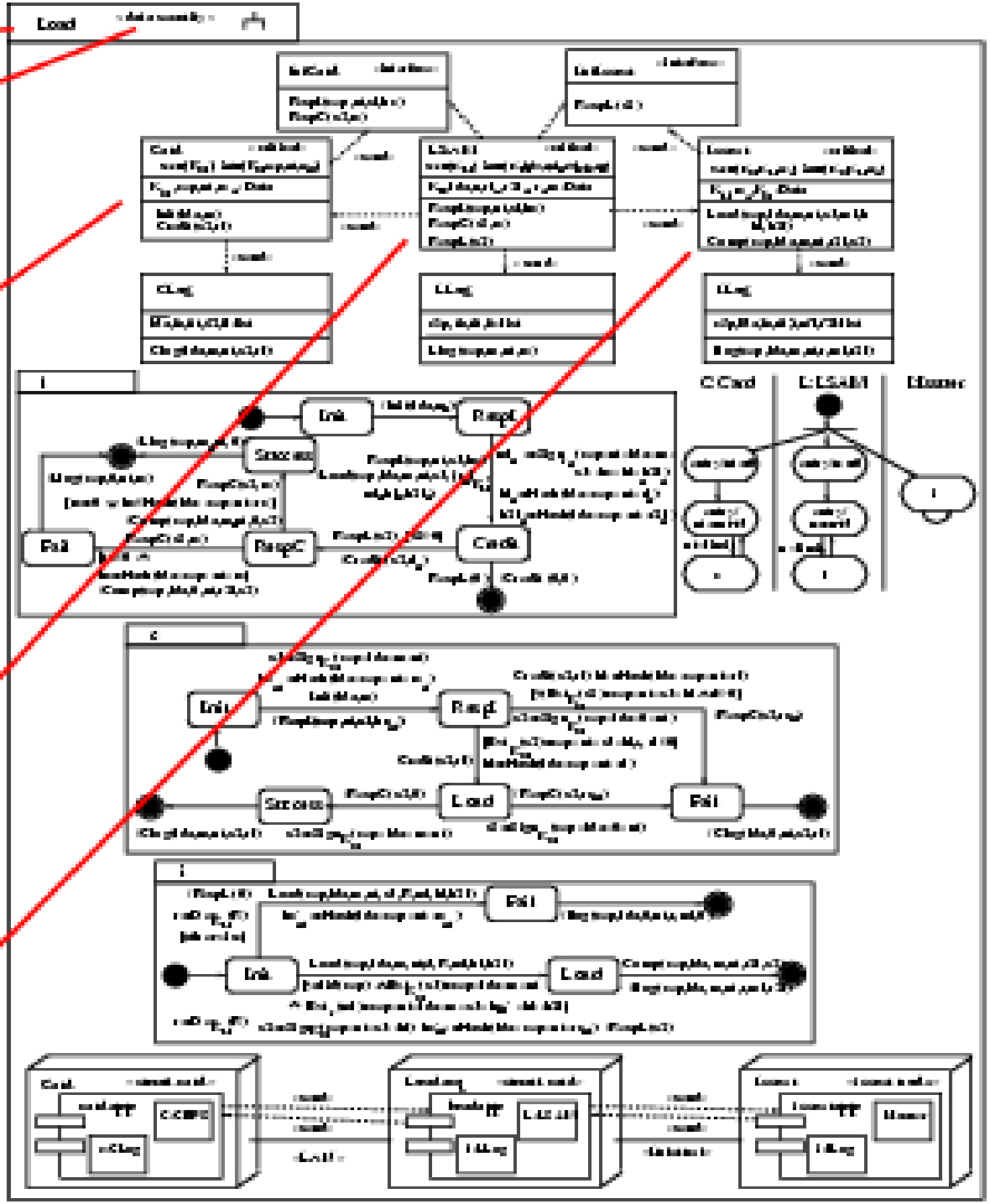
RespI(cep,nt,sl,hc)
RespC(s3,rc)
RespL(s2)

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.

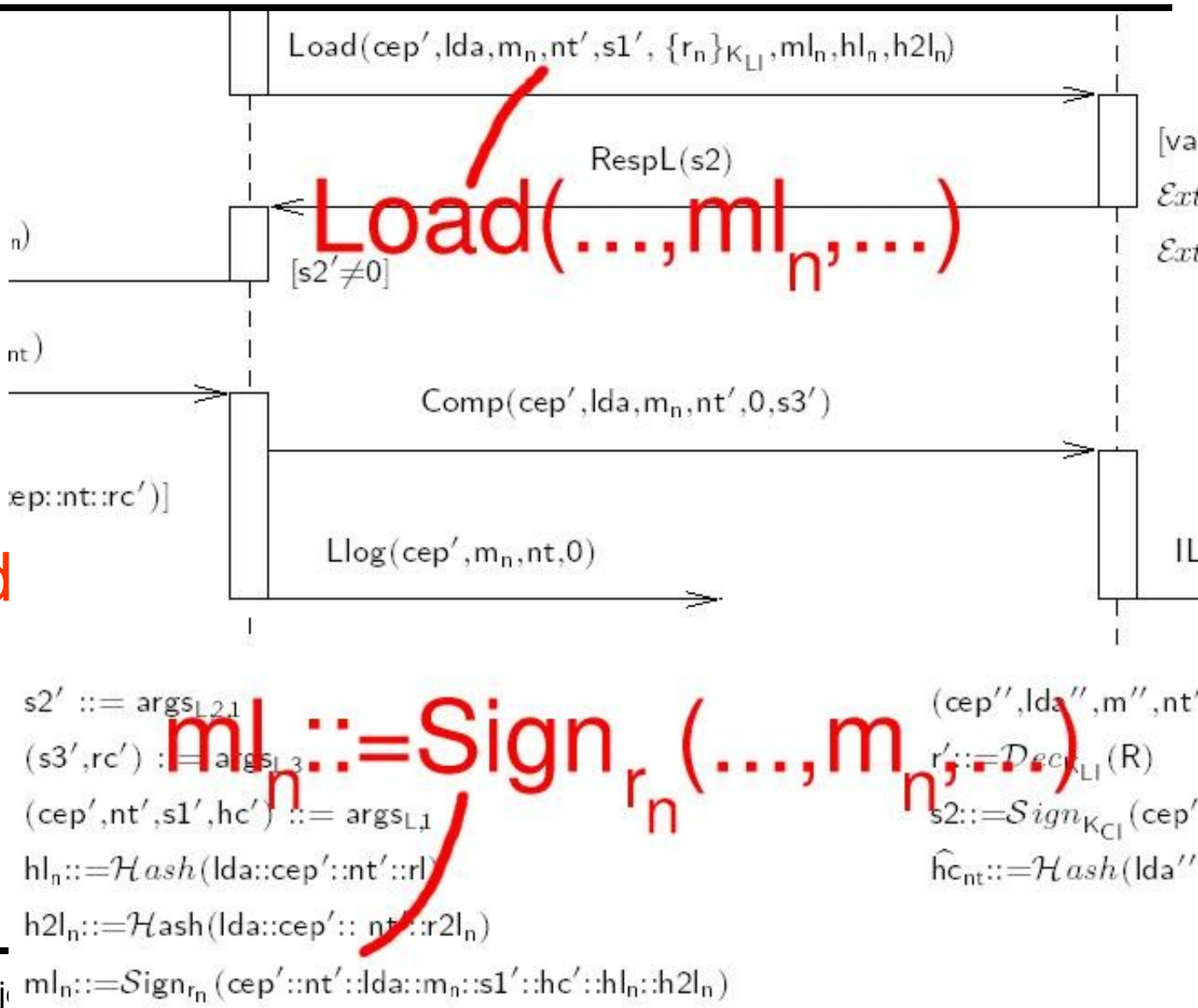
Use post-transaction settlement scheme.

Relies on secure auditing.

Verify this here (only executions completed without exception).

Flaw I

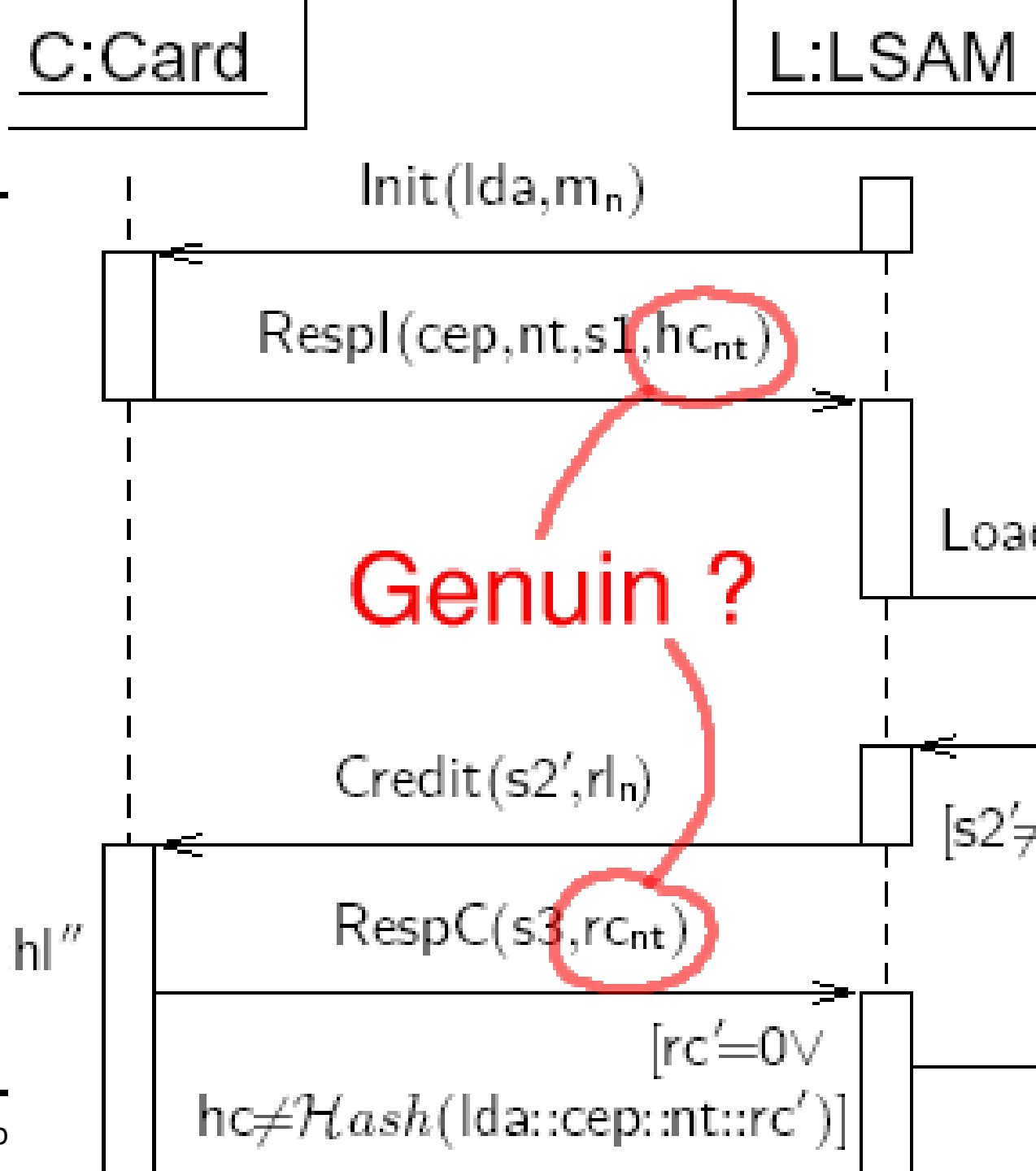
ml_n: „Proof“ for bank that load machine received money.
But: r_n shared between bank and load machine.



Flaw II

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

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



Conclusion

Seemingly first attempt at model based security testing for crypto-based Java legacy implementations.

Experiences so far encouraging.

Here only initial steps; more work needed (collaboration welcome !).

PS: Don`t miss presentation at FASE on Tuesday ☺

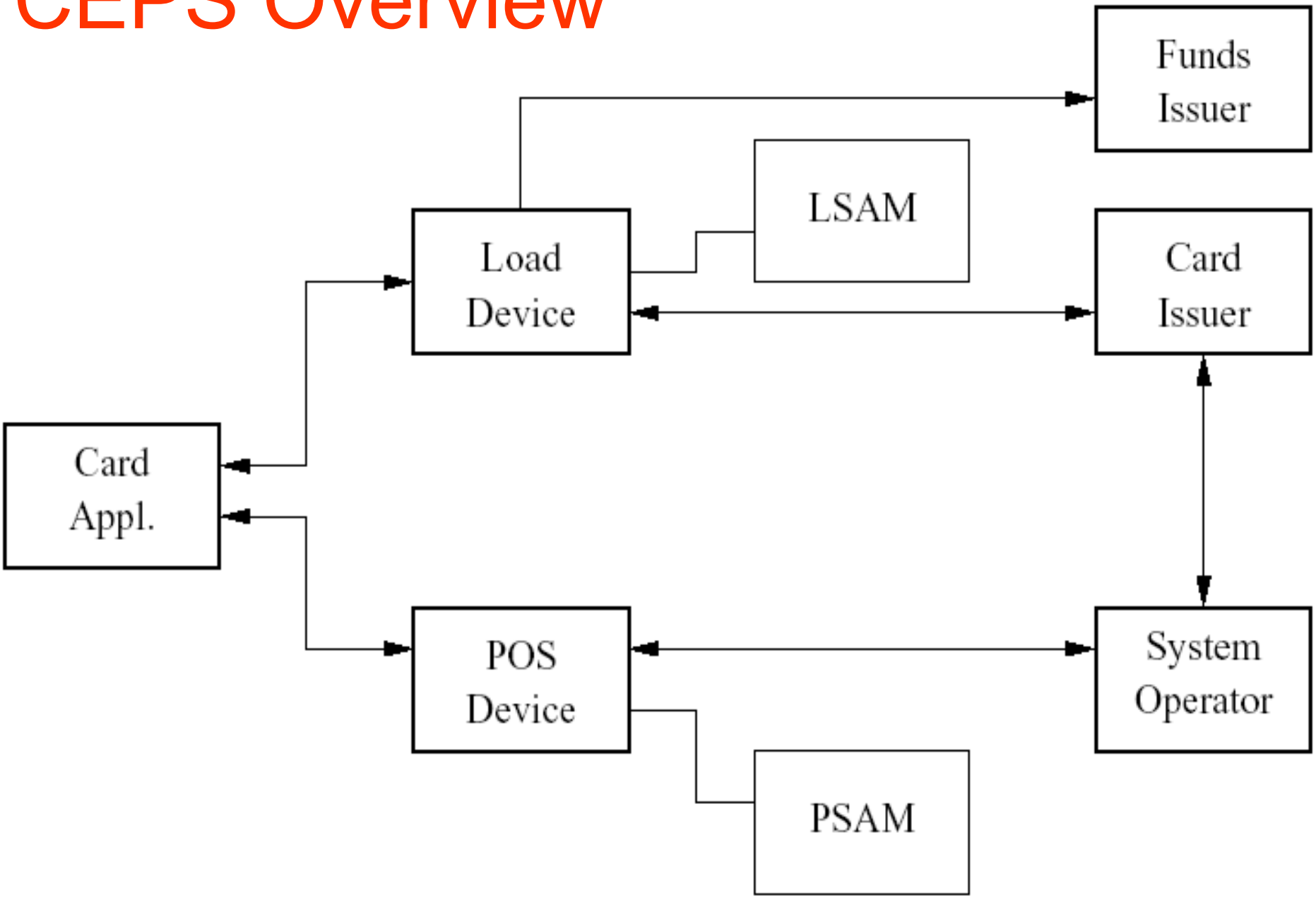
Questions ?

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

http://www.jurjens.de/jan

J.Jurjens@open.ac.uk

CEPS Overview



Purchase Protocol

Offline transaction to pay for goods with money previously loaded on card.

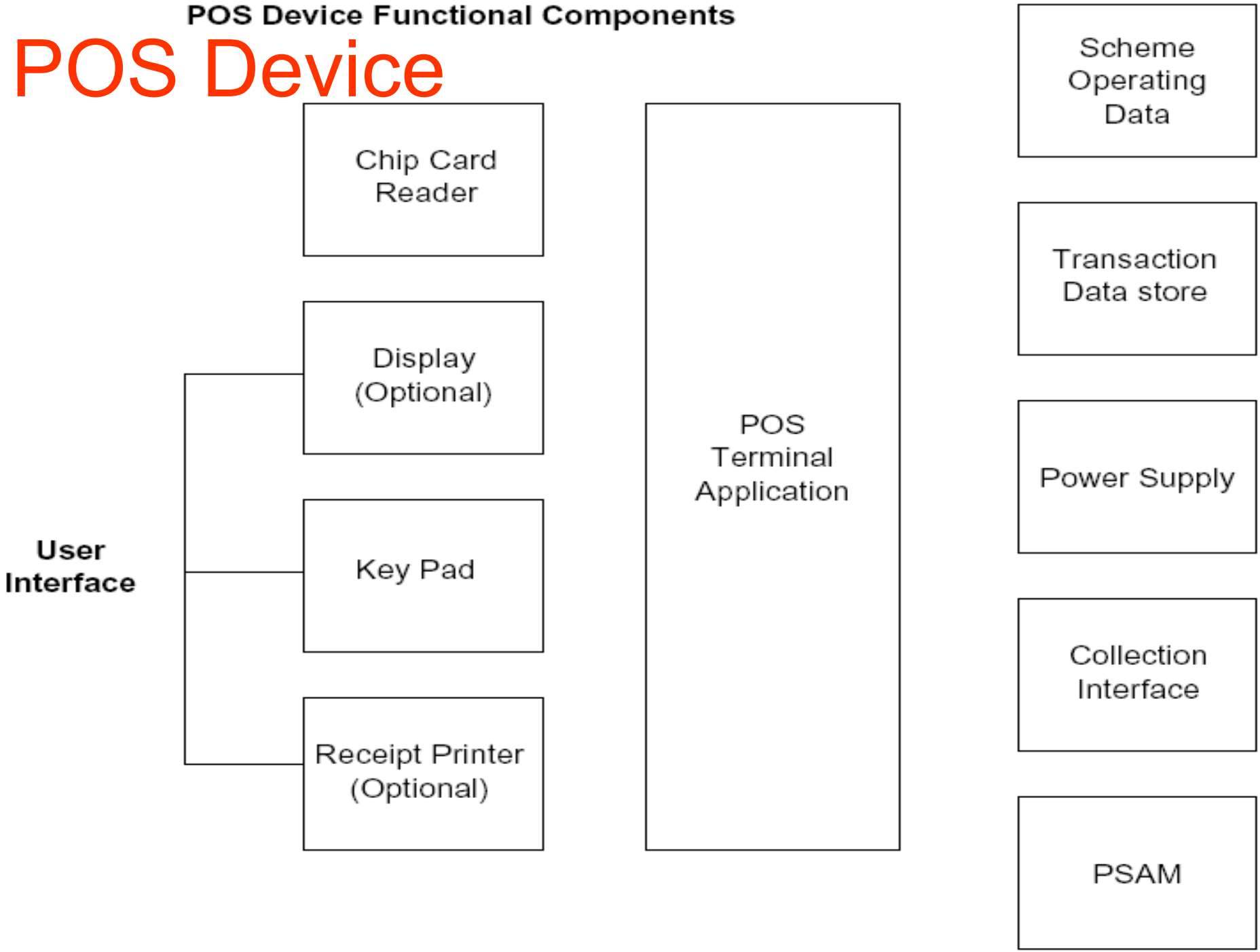
Protocol participants: customer's card, merchant's POS device.

POS device contains Purchase Security Application Module (PSAM): all security-critical data processing and storage for POS device.

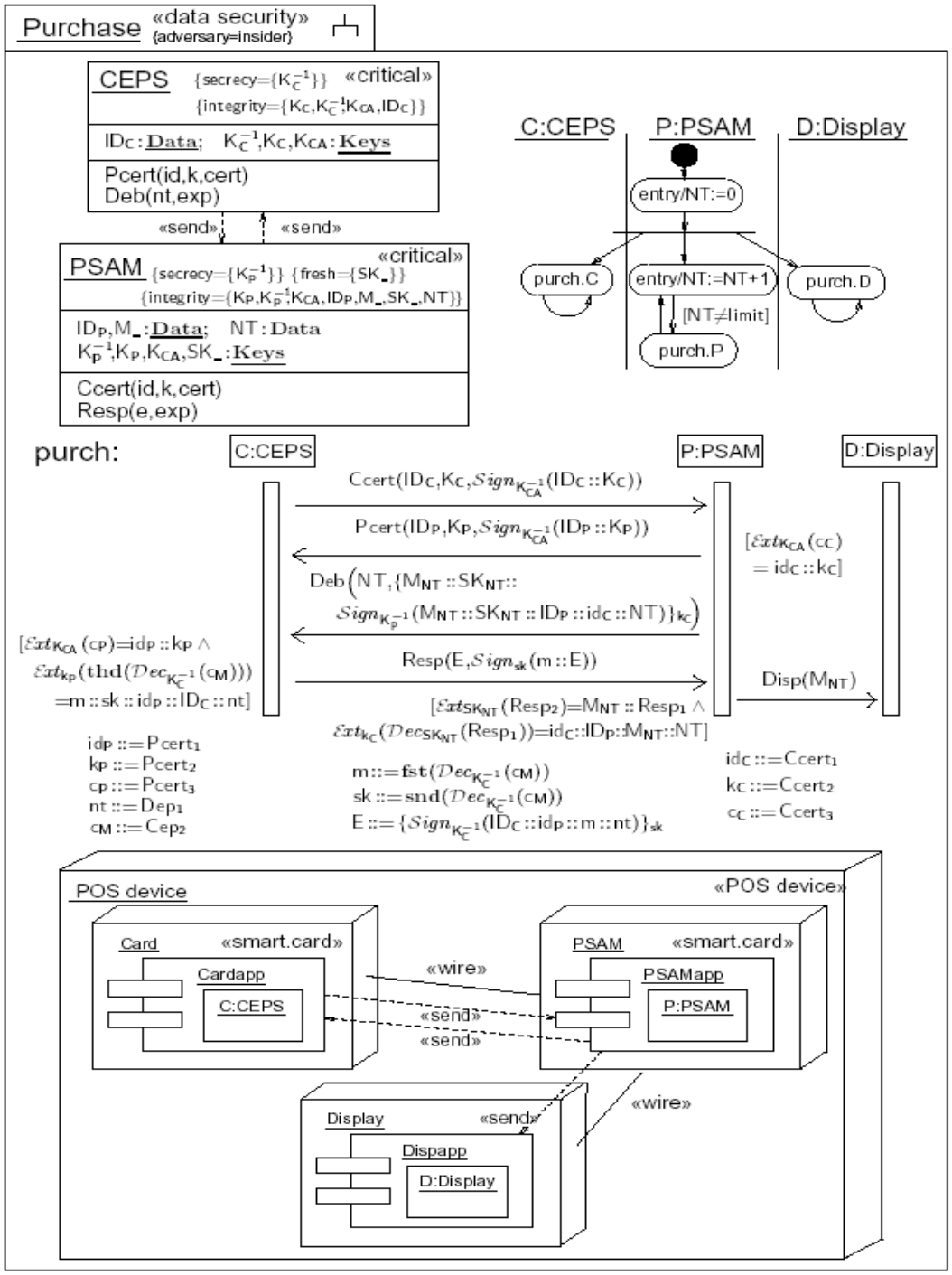
Card account balance adjusted; transaction data logged and later sent to issuer for financial settlement.

Use at public terminals; Internet use envisaged.

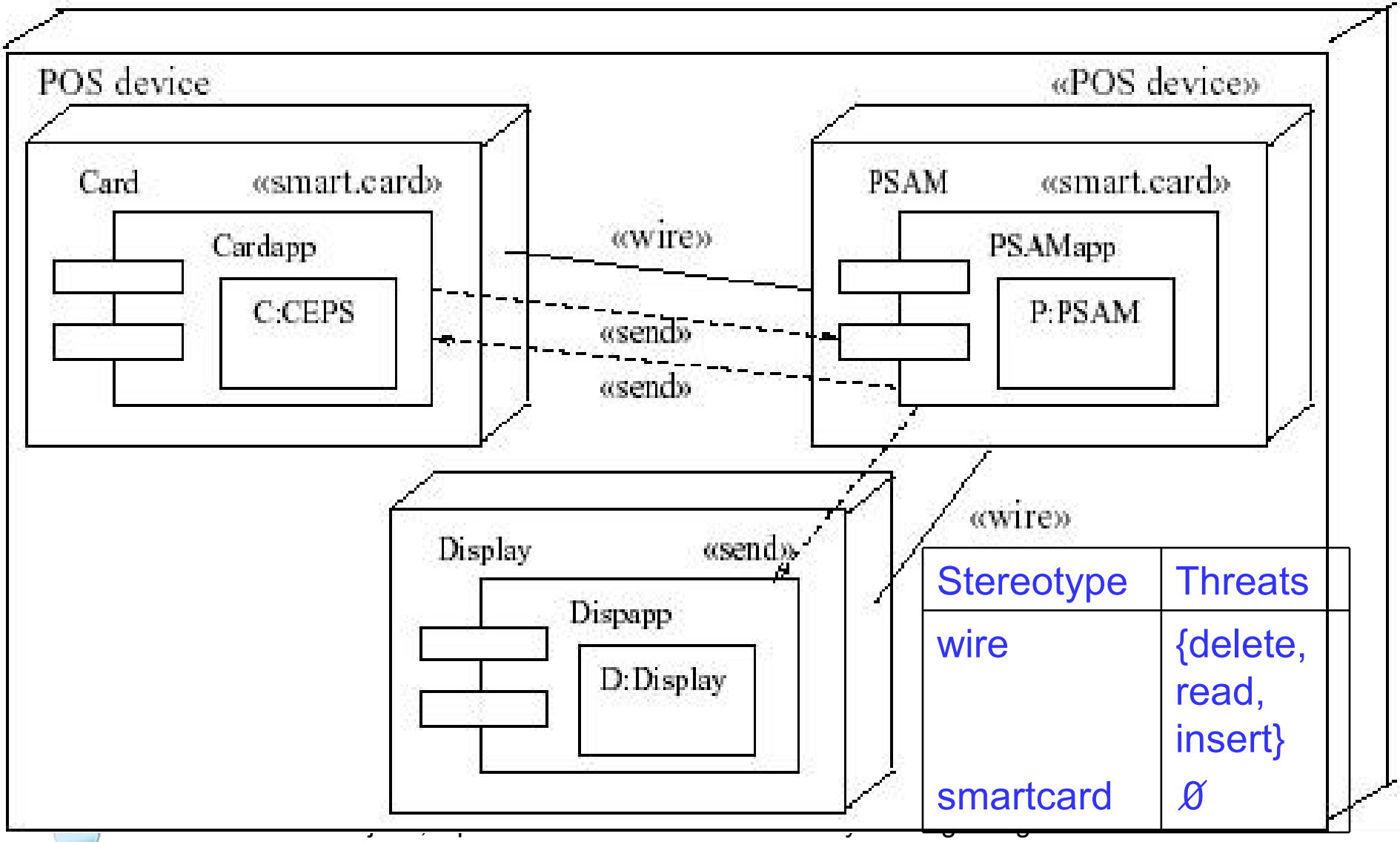
POS Device


POS Device Functional Components
Chip Card Reader
Display (Optional)
Key Pad
Receipt Printer (Optional)
User Interface
POS Terminal Application
Scheme Operating Data
Transaction Data store
Power Supply
Collection Interface
PSAM

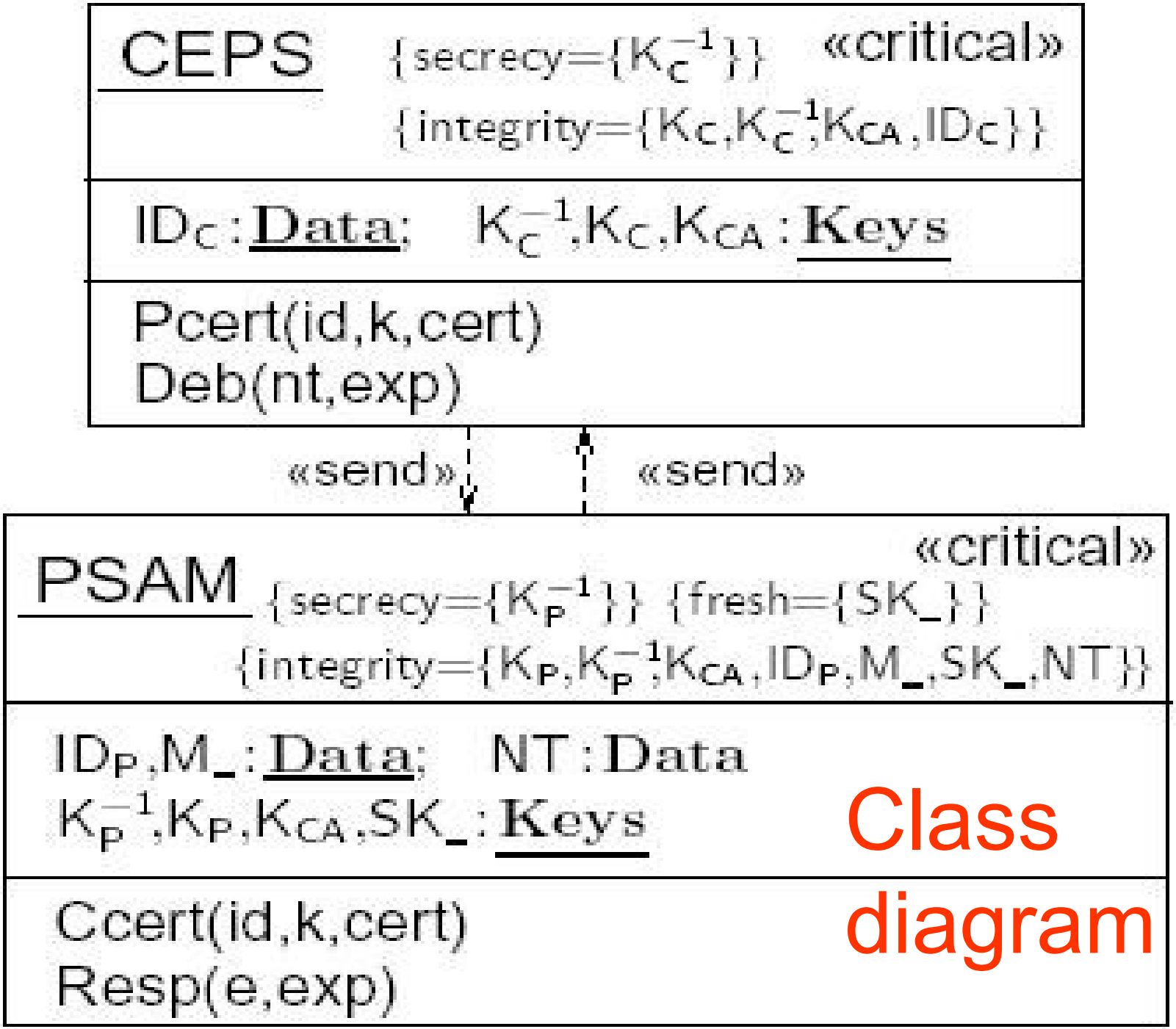
UMLsec Spec.

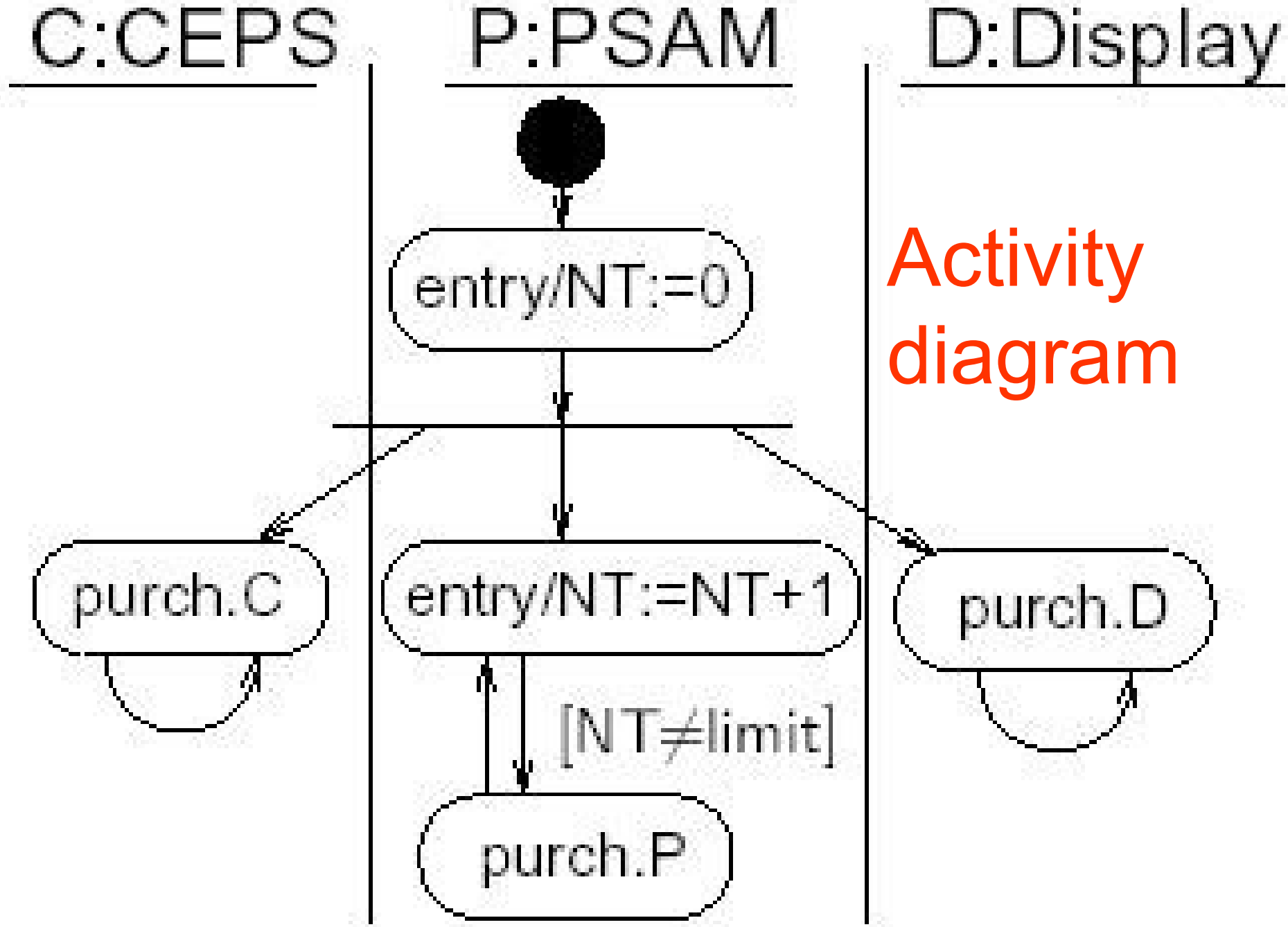

Purchase «data security» {adversary=insider}
CEPS {secrecy={K_C^-1}} «critical» {integrity={K_C,K_C^-1,K_CA,ID_C}}
ID_C:Data; K_C^-1,K_C,K_CA:Keys
Pcert(id,k,cert)
Deb(nt,exp)
«send» «send»
PSAM «critical» {secrecy={K_P^-1}} {fresh={SK_-}} {integrity={K_P,K_P^-1,K_CA,ID_P,M_-,SK_-,NT}}
ID_P,M_-:Data; NT:Data
K_P^-1,K_P,K_CA,SK_-:Keys
Ccert(id,k,cert)
Resp(e,exp)
C:CEPS
P:PSAM
D:Display
entry/NT:=0
purch.C
entry/NT:=NT+1
purch.D
[NT≠limit]
purch.P
purch:
C:CEPS
P:PSAM
D:Display
Ccert(ID_C,K_C,Sign_K_CA^-1(ID_C::K_C))
Pcert(ID_P,K_P,Sign_K_CA^-1(ID_P::K_P))
[Ext_K_CA(c_C) = id_C::k_C]
Deb(NT,{M_NT::SK_NT::Sign_K_P^-1(M_NT::SK_NT::ID_P::id_C::NT)}_k_C)
[Ext_K_CA(c_P)=id_P::k_P ∧ Ext_k_P(thd(Dec_K_C^-1(c_M))) =m::sk::id_P::ID_C::nt]
Resp(E,Sign_sk(m::E))
Disp(M_NT)
[Ext_SK_NT(Resp_2)=M_NT::Resp_1 ∧ Ext_k_C(Dec_SK_NT(Resp_1))=id_C::ID_P::M_NT::NT]
id_P::=Pcert_1
k_P::=Pcert_2
c_P::=Pcert_3
nt::=Dep_1
c_M::=Cep_2
m::=fst(Dec_K_C^-1(c_M))
sk::=snd(Dec_K_C^-1(c_M))
E::={Sign_K_C^-1(ID_C::id_P::m::nt)}_sk
id_C::=Ccert_1
k_C::=Ccert_2
c_C::=Ccert_3
POS device
«POS device»
Card
«smart.card»
Cardapp
C:CEPS
«wire»
PSAM
«smart.card»
PSAMapp
P:PSAM
«send»
«send»
Display
«send»
Dispapp
D:Display
«wire»

Purchase Protocol: Architecture

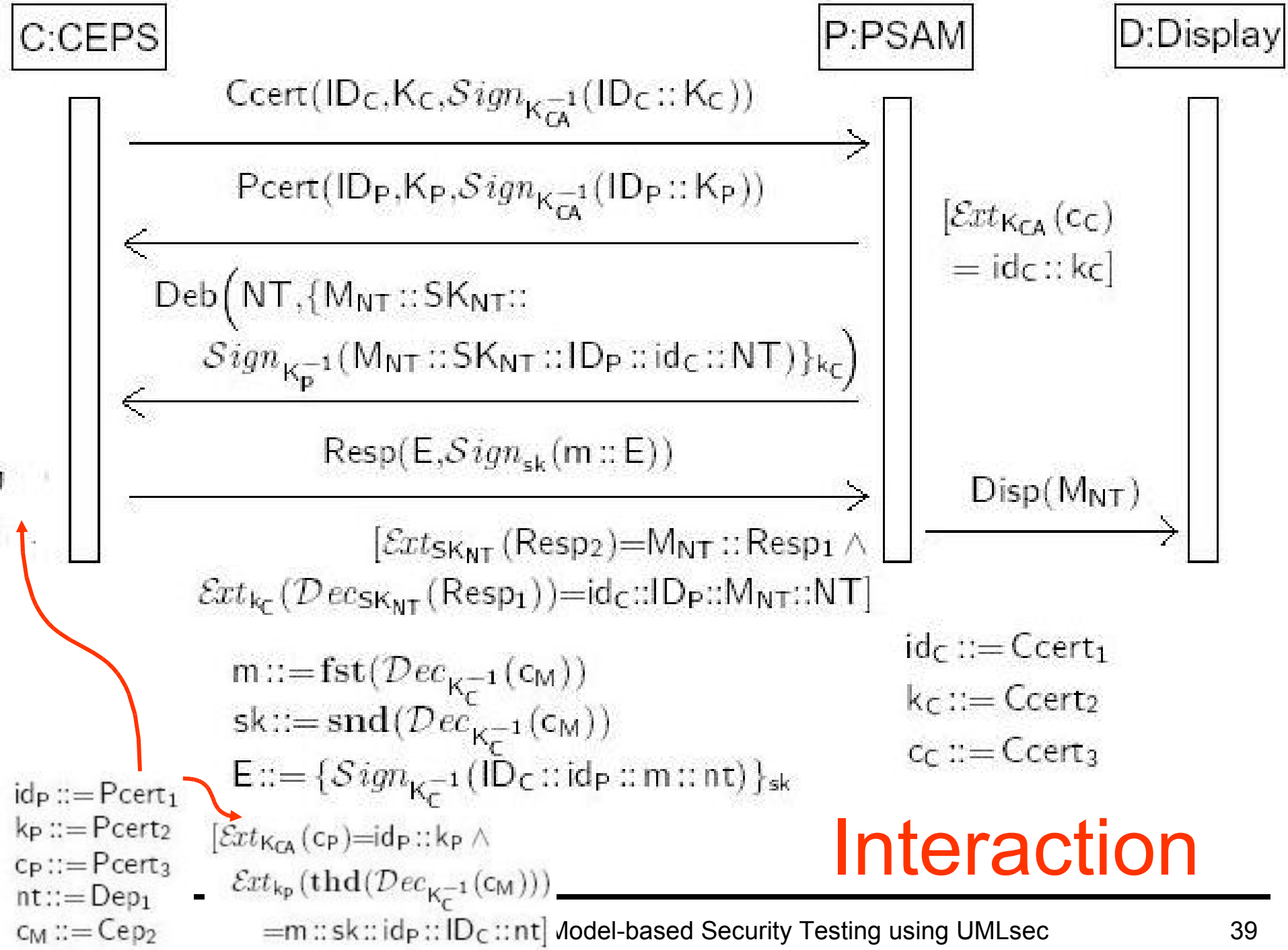


Stereotype	Threats
wire	{delete, read, insert}
smartcard	Ø


CEPS
{secrecy={K_C^-1}}
«critical»
{integrity={K_C,K_C^-1,K_CA,ID_C}}
ID_C : Data; K_C^-1,K_C,K_CA : Keys
Pcert(id,k,cert)
Deb(nt,exp)
«send»
«send»
PSAM
«critical»
{secrecy={K_P^-1}} {fresh={SK_}}
{integrity={K_P,K_P^-1,K_CA,ID_P,M_,SK_,NT}}
ID_P,M_ : Data; NT : Data
K_P^-1,K_P,K_CA,SK_ : Keys
Ccert(id,k,cert)
Resp(e,exp)
Class diagram



Activity diagram


C:CEPS
P:PSAM
D:Display
Ccert(ID_C,K_C,Sign_{K_CA^-1}(ID_C :: K_C))
Pcert(ID_P,K_P,Sign_{K_CA^-1}(ID_P :: K_P))
[Ext_{K_CA}(c_C) = id_C :: k_C]
Deb(NT,{M_NT :: SK_NT :: Sign_{K_P^-1}(M_NT :: SK_NT :: ID_P :: id_C :: NT)}_{k_C})
Resp(E,Sign_sk(m :: E))
Disp(M_NT)
[Ext_{SK_NT}(Resp_2)=M_NT :: Resp_1 ∧ Ext_{k_C}(Dec_{SK_NT}(Resp_1))=id_C::ID_P::M_NT::NT]
m ::= fst(Dec_{K_C^-1}(c_M))
sk ::= snd(Dec_{K_C^-1}(c_M))
E ::= {Sign_{K_C^-1}(ID_C :: id_P :: m :: nt)}_sk
id_C ::= Ccert_1
k_C ::= Ccert_2
c_C ::= Ccert_3
id_P ::= Pcert_1
k_P ::= Pcert_2
c_P ::= Pcert_3
nt ::= Dep_1
c_M ::= Cep_2
[Ext_{K_CA}(c_P)=id_P :: k_P ∧ Ext_{k_P}(thd(Dec_{K_C^-1}(c_M))) =m :: sk :: id_P :: ID_C :: nt]
Interaction

Security Threat Model

Supposed to provide mutual authentication between terminal and card.

Card, PSAM assumed tamper-resistant.

Intercept communication links, replace components.

Possible attack motivations:

- **(Non-)Cardholder**: purchase without pay.
- **Merchant employee**: buy digital content with customer's card.
- **Card issuer employee**: credit transactions to own (cover-up) business.

May coincide or collude.

Security conditions (informal)

Cardholder security. Merchant can only claim amount registered on card after transaction.

Merchant security. Merchant receives proof of transaction in exchange for sold good.

Card issuer security. Sum of balances of valid cards and PSAMs unchanged by transaction.

Merchant security

Each time display D receives value M_{NT}, P is in possession of $Sign_{KCA-1}(ID_C::K_C)$ and $Sign_{KC}^{1}(ID_C::ID_P::M_{NT}::NT)$ for some ID_C, K_C^{-1} and new value NT.

Not satisfied. Attack automatically computed.

Attack exploits the fact that POS device is not tamper-proof.

Redirect messages between card and PSAM to another PSAM (e.g. to buy digital content, on the cost of the cardholder).

$$\mathsf{C} \xrightarrow{\mathsf{Ccert}(\mathsf{ID_C},\mathsf{K_C},\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}(\mathsf{ID_C}::\mathsf{K_C}))} A \xrightarrow{\mathsf{Ccert}(\mathsf{ID_C},\mathsf{K_C},\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}(\mathsf{ID_C}::\mathsf{K_C}))} \mathsf{P}'$$

$$\mathsf{C} \xleftarrow{\mathsf{Pcert}(\mathsf{ID_{P'}},\mathsf{K_{P'}},\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}(\mathsf{ID_{P'}}::\mathsf{K_{P'}}))} A \xleftarrow{\mathsf{Pcert}(\mathsf{ID_{P'}},\mathsf{K_{P'}},\mathcal{S}ign_{\mathsf{K_{CA}^{-1}}}(\mathsf{ID_{P'}}::\mathsf{K_{P'}}))} \mathsf{P}'$$

$$A \xleftarrow{\mathsf{Deb}(\mathsf{NT},\{\mathsf{M_{NT}}::\mathsf{SK_{NT}}::\mathcal{S}ign_{\mathsf{K_{P'}^{-1}}}(\mathsf{M_{NT}}::\mathsf{SK_{NT}}::\mathsf{ID_{P'}}::\mathsf{id_C}::\mathsf{NT})\}_{\mathsf{k_C}})} \mathsf{P}'$$

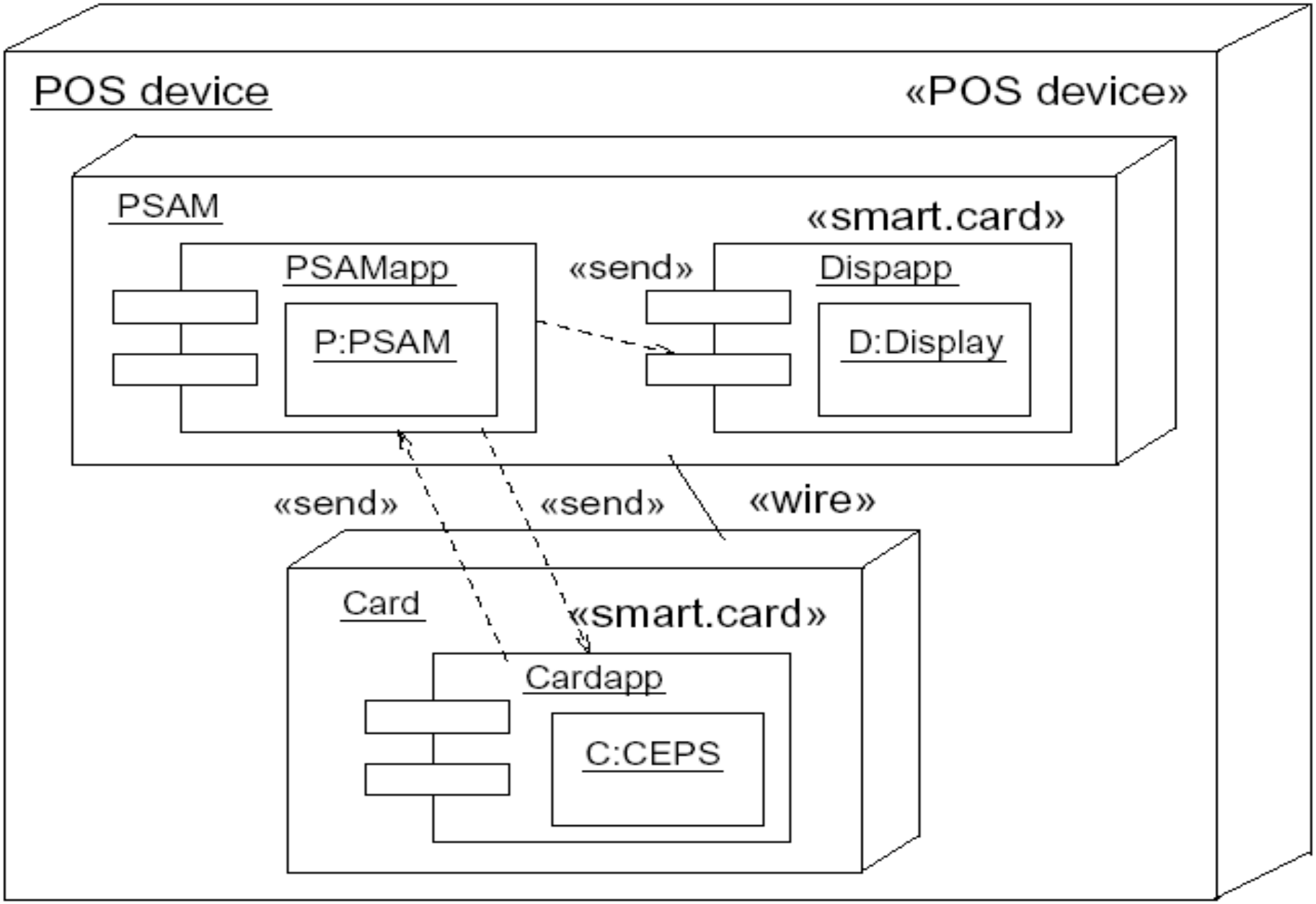
$$\mathsf{C} \xleftarrow{\mathsf{Deb}(\mathsf{NT},\{\mathsf{M_{NT}}::\mathsf{SK_{NT}}::\mathcal{S}ign_{\mathsf{K_{P'}^{-1}}}(\mathsf{M_{NT}}::\mathsf{SK_{NT}}::\mathsf{ID_{P'}}::\mathsf{id_C}::\mathsf{NT})\}_{\mathsf{k_C}})} A$$

$$\mathsf{C} \xrightarrow{\mathsf{Resp}(\mathsf{E},\mathcal{S}ign_{\mathsf{sk}}(\mathsf{m}::\mathsf{E}))} A \xrightarrow{\mathsf{Resp}(\mathsf{E},\mathcal{S}ign_{\mathsf{sk}}(\mathsf{m}::\mathsf{E}))} \mathsf{P}'$$

Attack

$$A \xrightarrow{\mathsf{Disp}(\mathsf{M_{NT}})} \mathsf{D}$$

Fix: Protect PSAM-device link



Load Protocol

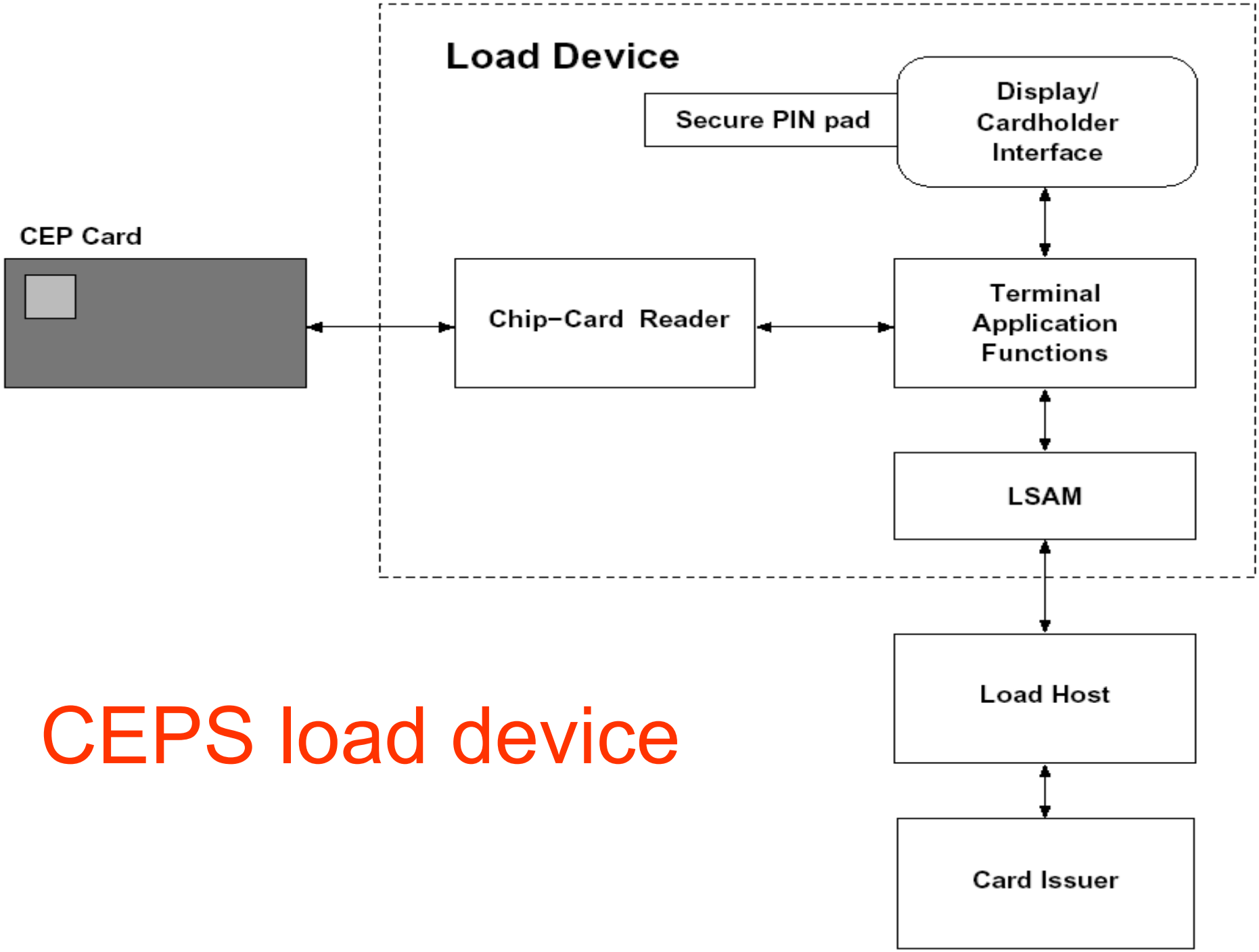
Unlinked, cash-based load transaction (on-line).

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.



CEPS load device

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}}

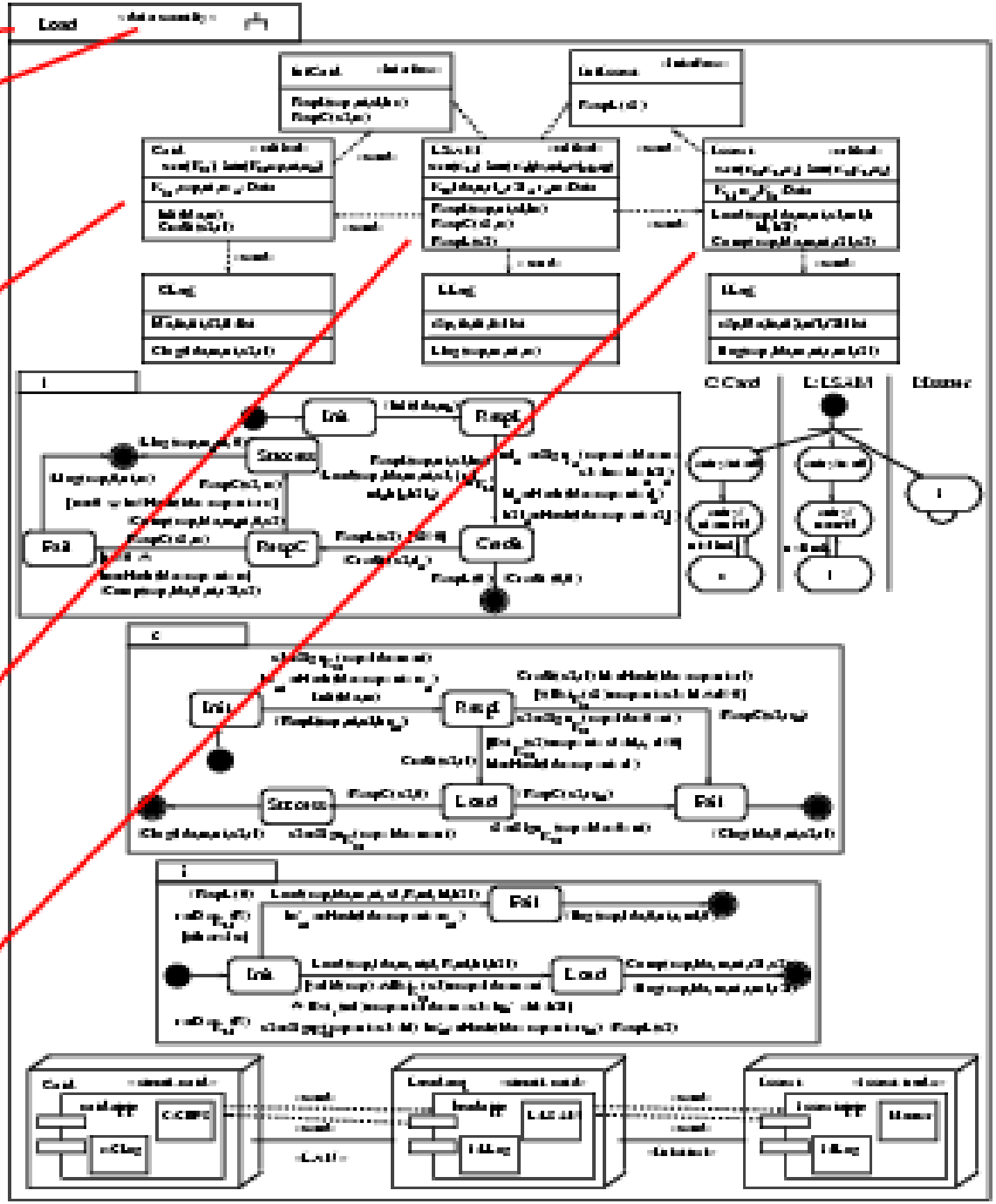
lda, n, rl_n, $r2l_n$, m_n : Data
K_{LI}, r_n : Keys

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

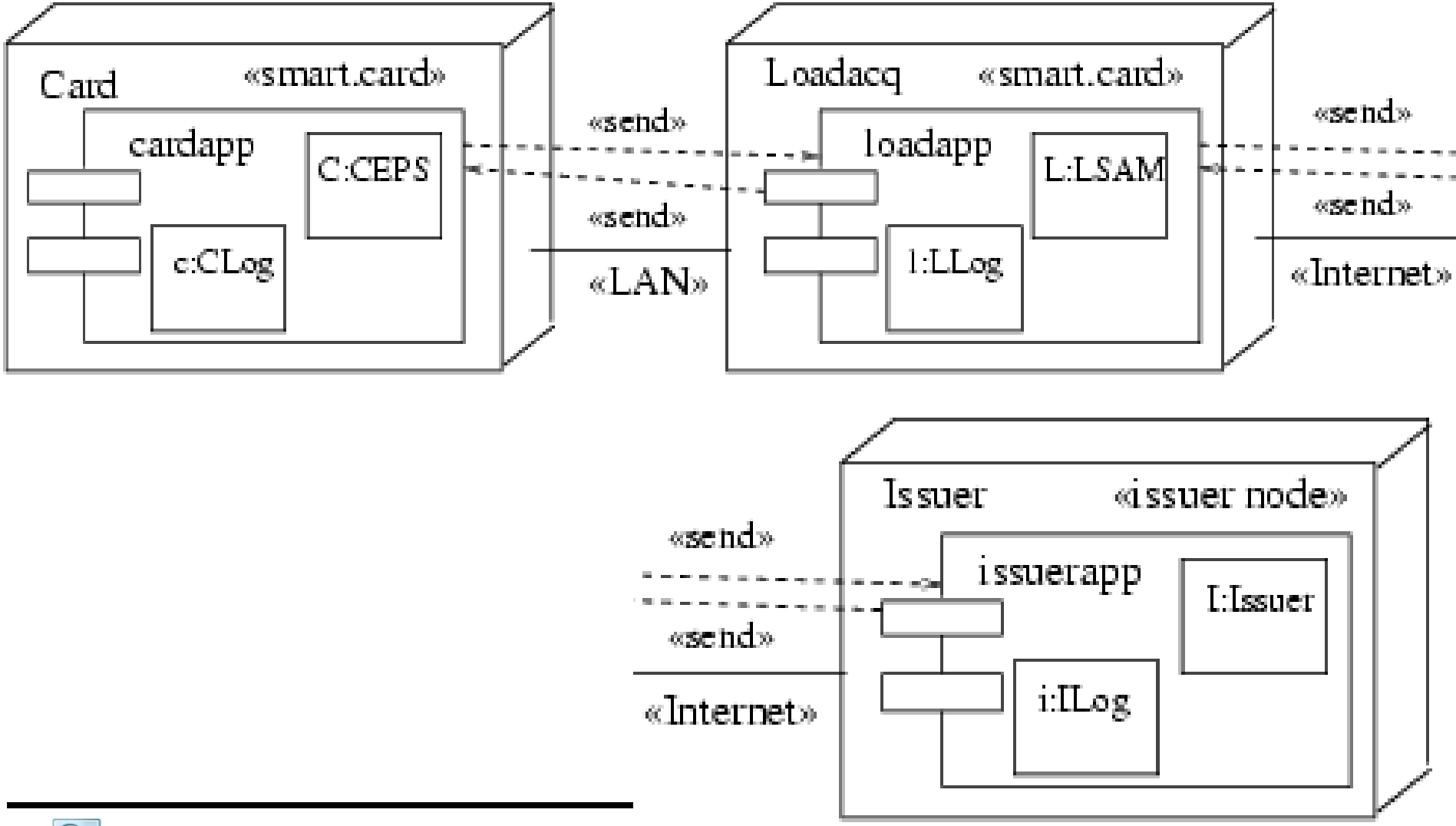
Issuer «critical»
{secrecy={K_{CI},K_{LI},rc_{nt}}}
{integrity={K_{CI},K_{LI},rc_{nt}}}

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

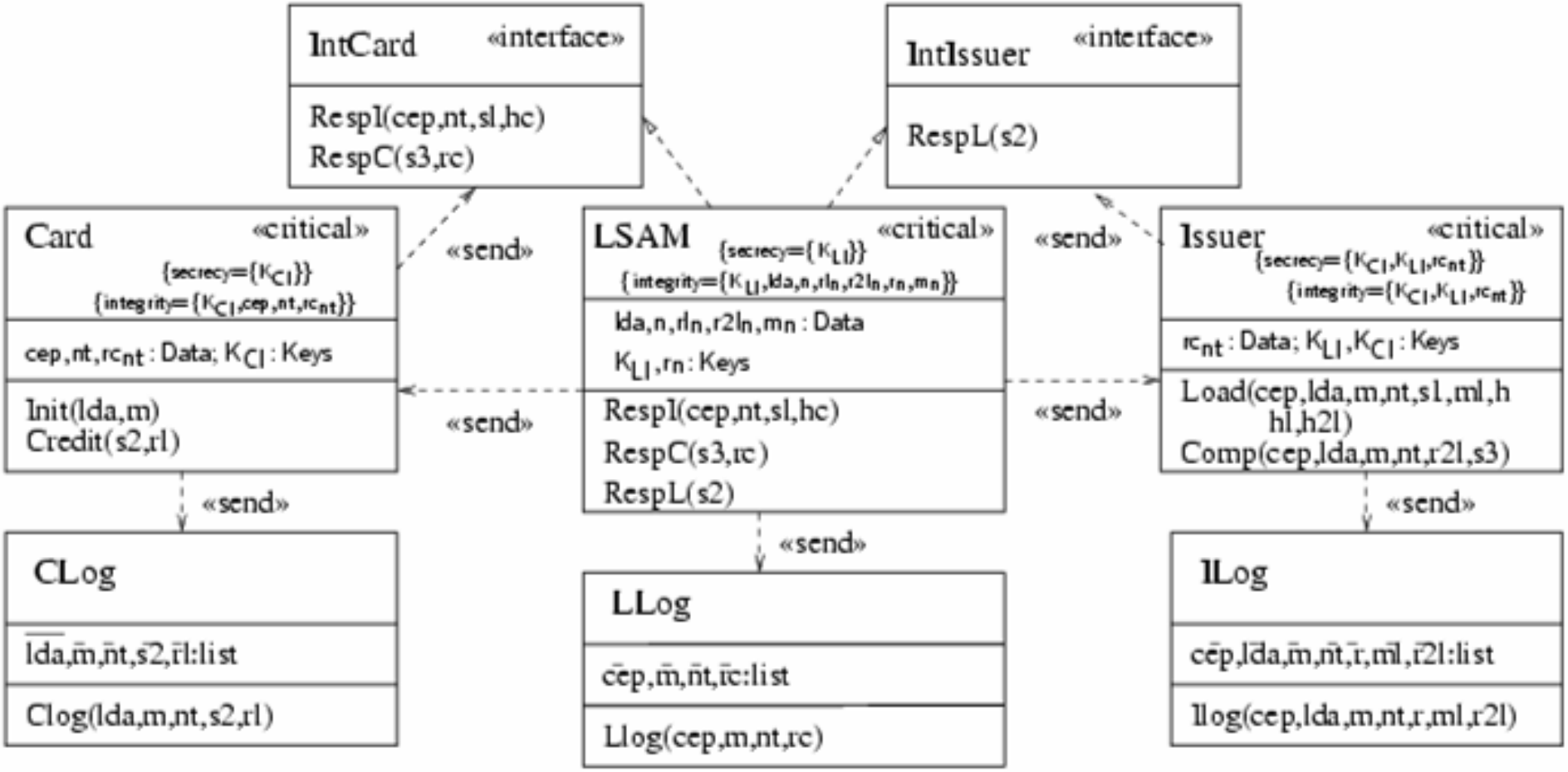
Load(cep,lda,m,nt,s1,ml,h
hl,h2l)
Comp(cep,lda,m,nt,r2l,s3)



Load Protocol: Physical View

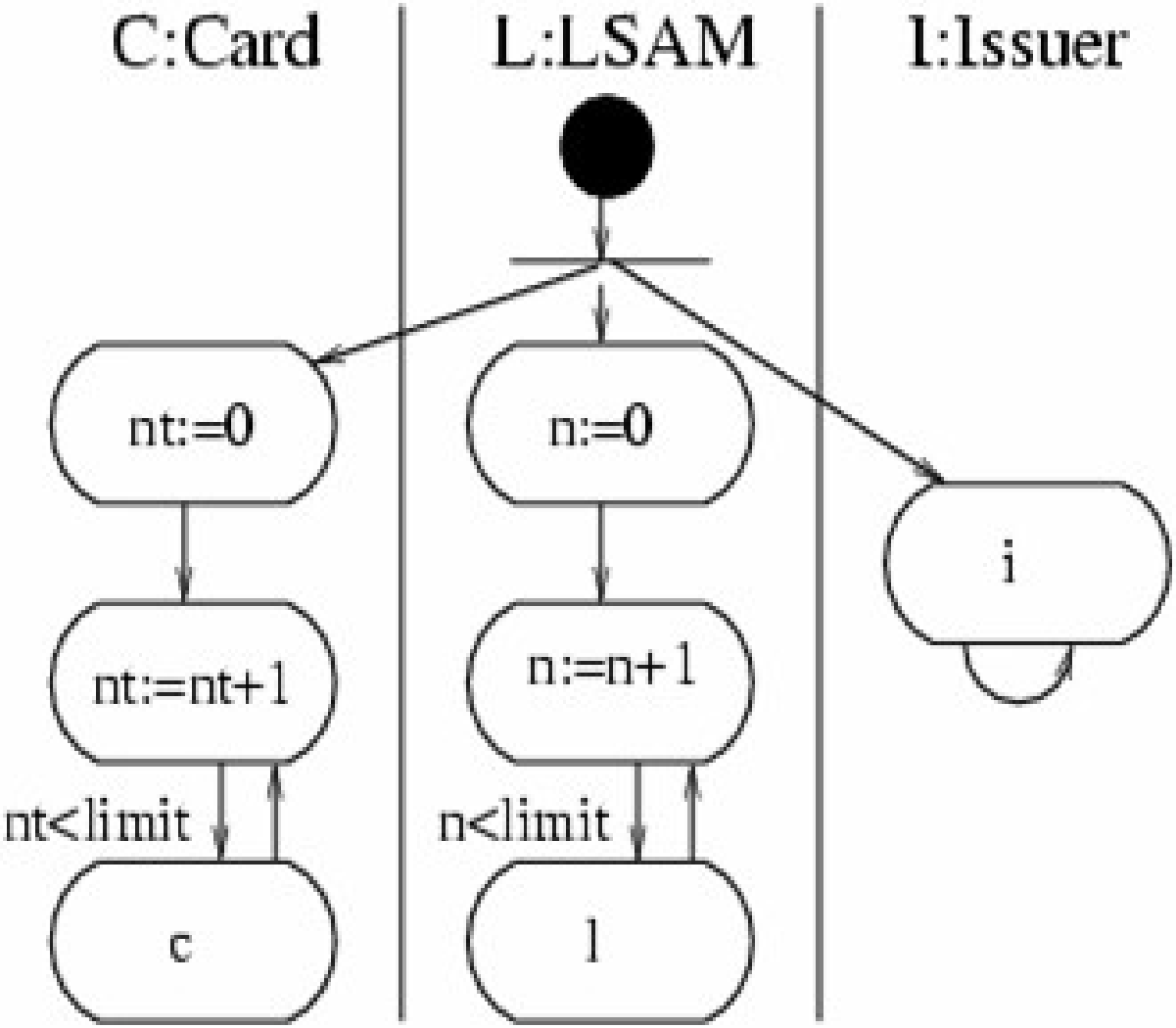


Load Protocol: Structural View

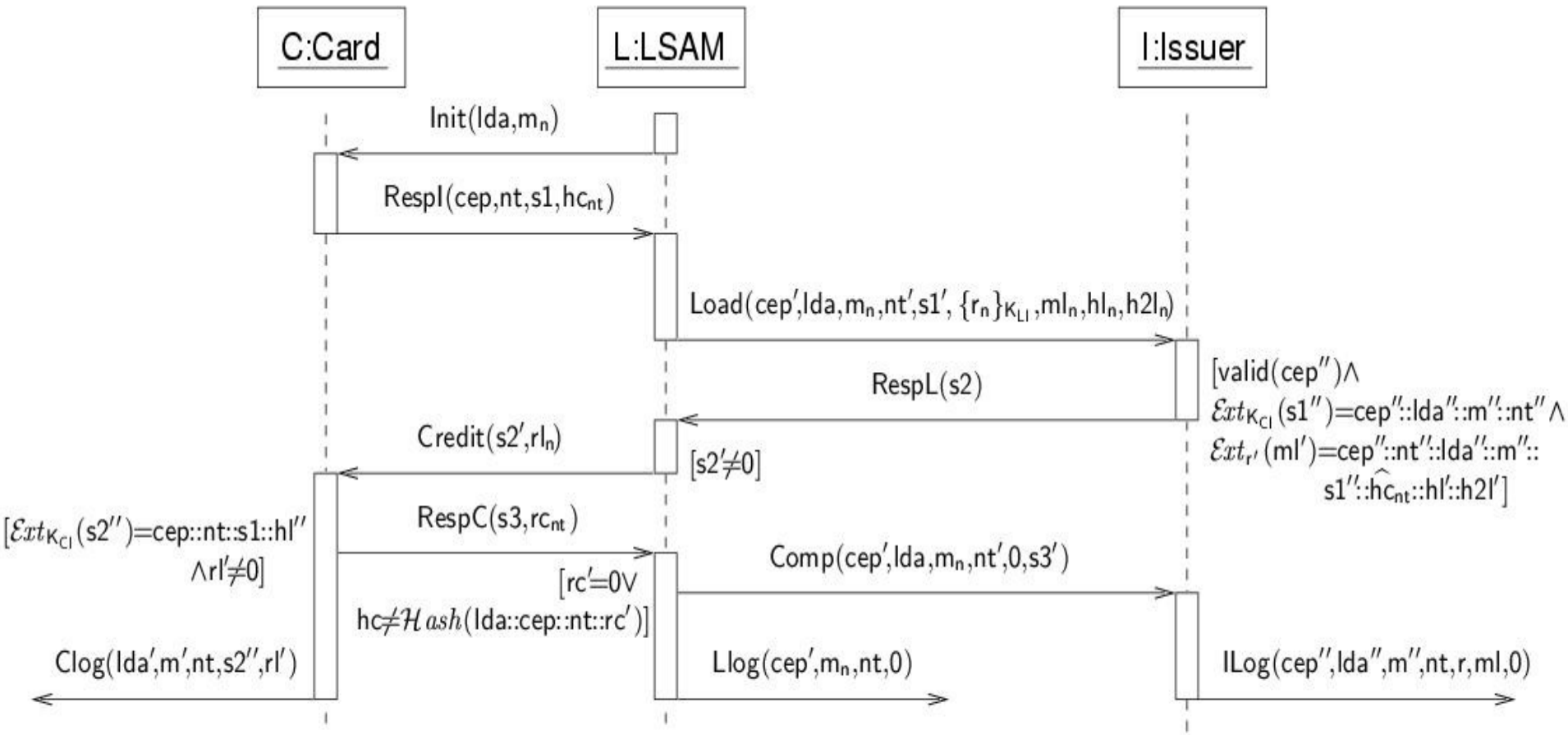


Variable	Explanation
C	card
L	LSAM
I	card issuer
$\mathsf{rc_{nt}}$	secret random values shared between card and issuer
$\mathsf{rl_n}$, $\mathsf{r2l_n}$	random numbers of LSAM
$\mathsf{r_n}$	symmetric keys of LSAM
$\mathsf{m_n}$	transaction amounts
m, rl, hl	$\mathsf{m_n}$, $\mathsf{rl_n}$, $\mathsf{hl_n}$ as received at card issuer
nt	card transaction number
n	acquirer-generated identification number
lda	load device identifier
cep	card identifier
$\mathsf{s1}$	card signature: $\mathcal{S}ign_{\mathsf{K_{CI}}}(\mathsf{cep} :: \mathsf{lda} :: \mathsf{m} :: \mathsf{nt})$
$\mathsf{hc_{nt}}$	card hash value: $\mathcal{H}ash(\mathsf{lda} :: \mathsf{cep} :: \mathsf{nt} :: \mathsf{rc_{nt}})$
$\widehat{\mathsf{hc}}_{\mathsf{nt}}$	$\mathsf{hc_{nt}}$ as created at issuer
rc, hc	$\mathsf{rc_{nt}}$, $\mathsf{hc_{nt}}$ as received at load acquirer
$\mathsf{K_{CI}}$	key shared between card and issuer
$\mathsf{K_{LI}}$	key shared between LSAM and issuer
$\mathsf{ml_n}$	$\mathcal{S}ign_{\mathsf{r_n}}(\mathsf{cep} :: \mathsf{nt} :: \mathsf{lda} :: \mathsf{m_n} :: \mathsf{s1} :: \mathsf{hc} :: \mathsf{hl_n} :: \mathsf{h2l_n})$ (signed by LSAM)
$\mathsf{hl_n}$	hash of transaction data: $\mathcal{H}ash(\mathsf{lda} :: \mathsf{cep} :: \mathsf{nt} :: \mathsf{rl})$
$\mathsf{h2l_n}$	hash of transaction data: $\mathcal{H}ash(\mathsf{lda} :: \mathsf{cep} :: \mathsf{nt} :: \mathsf{r2l})$
$\mathsf{s2}$	issuer signature: $\mathcal{S}ign_{\mathsf{K_{CI}}}(\mathsf{cep} :: \mathsf{nt} :: \mathsf{s1} :: \mathsf{hl})$
$\mathsf{s3}$	card signature of the form $\mathcal{S}ign_{\mathsf{K_{CI}}}(\mathsf{cep} :: \mathsf{lda} :: \mathsf{m} :: \mathsf{nt})$

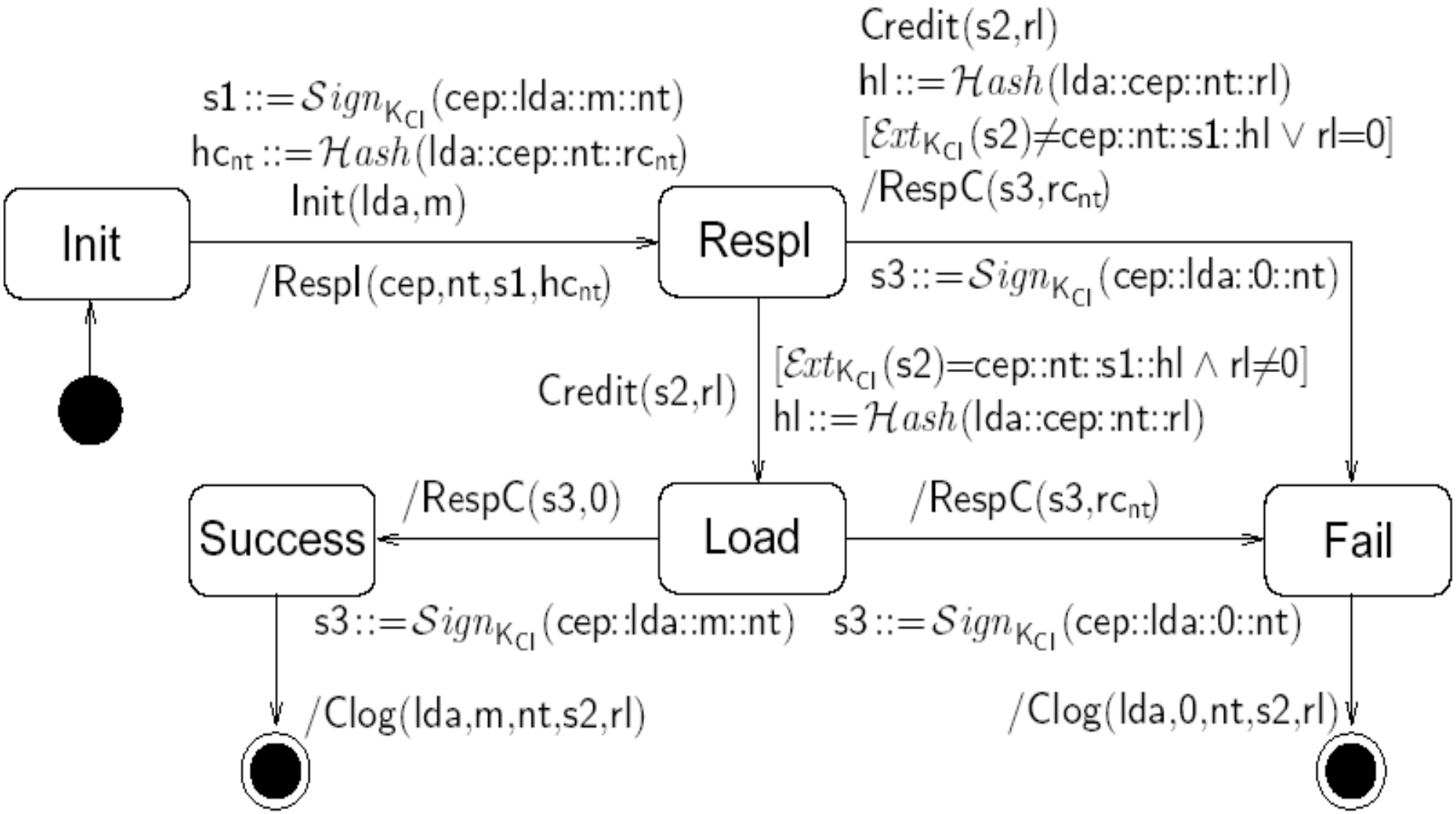
Load Protocol: Coordination View



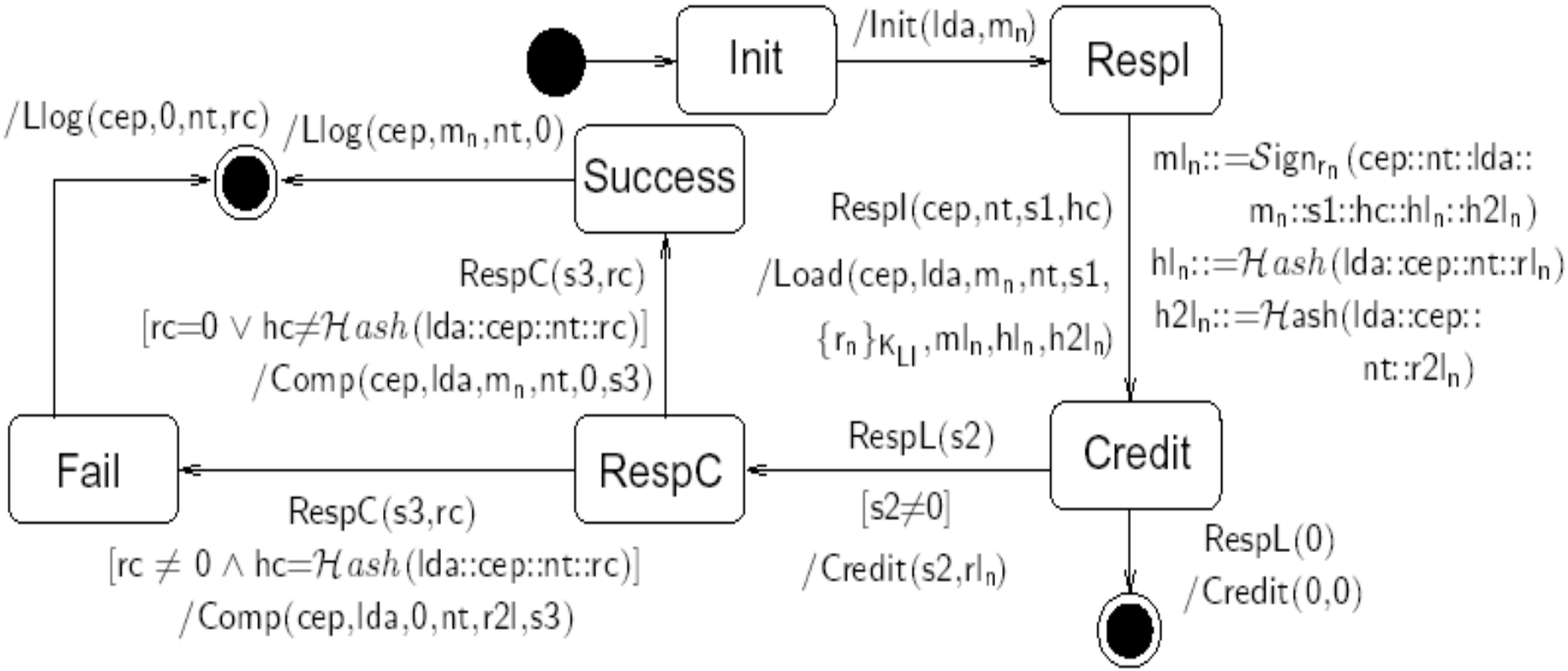
Load Protocol: Interaction View

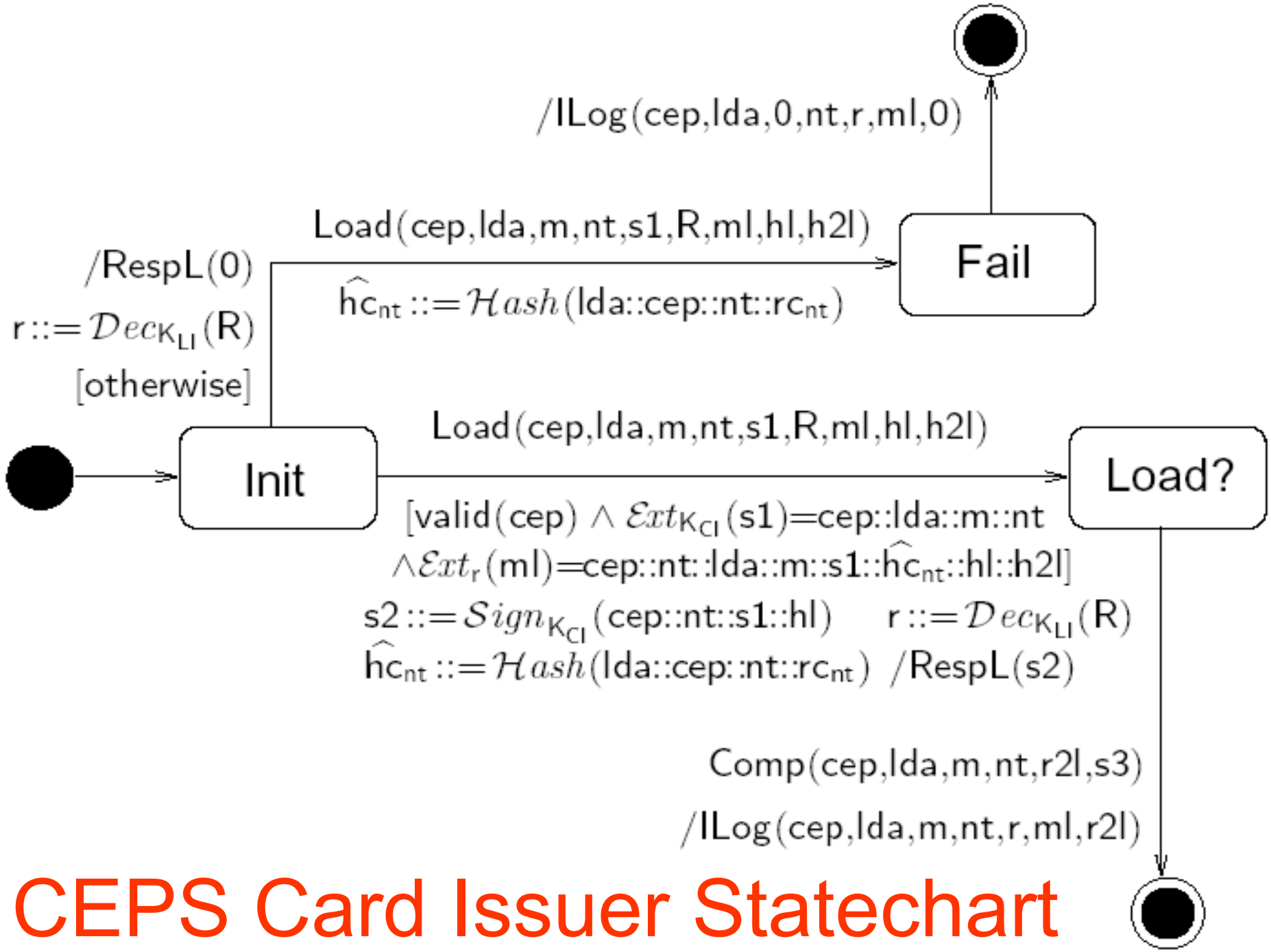


CEPS Card Statechart



CEPS LSAM Statechart





CEPS Card Issuer Statechart

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 I:LLog (so load acquirer L has received and retains m_n in cash) or
- Llog $(cep, lda, 0, nt)$ has been sent to I : 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)

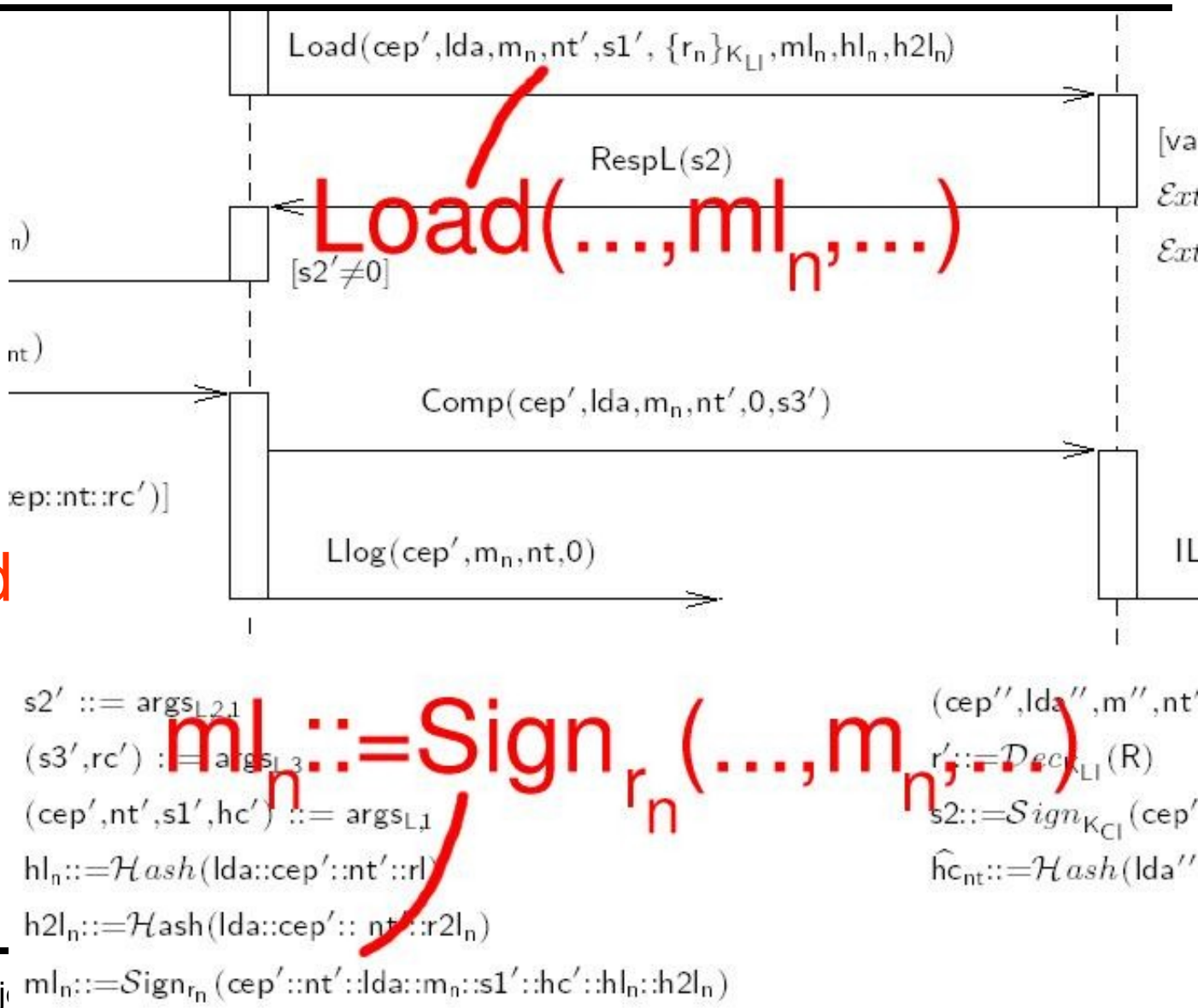
Flaw

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

Why ?

Flaw I

ml_n: „Proof“ for bank that load machine received money.
But: r_n shared between bank and load machine.



Correction

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

Verify this version wrt. above conditions.

Flaw II

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

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