Understanding Security Goals Provided by Cryptoprotocol Implementations

Jan Jürjens

Software & Systems Engineering

Technical University of Munich

juerjens@in.tum.de

http://www4.in.tum.de/~juerjens

Software Engineering & Security

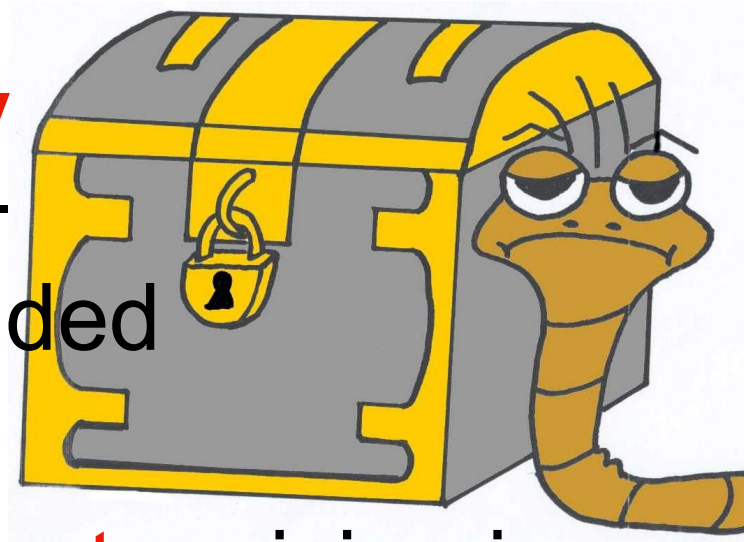
„Penetrate-and-patch“ (aka „banana strategy):

- insecure
- disruptive

Traditional formal methods: limited adoptation in industry.

- training people
- constructing formal specifications.

Model-based Security



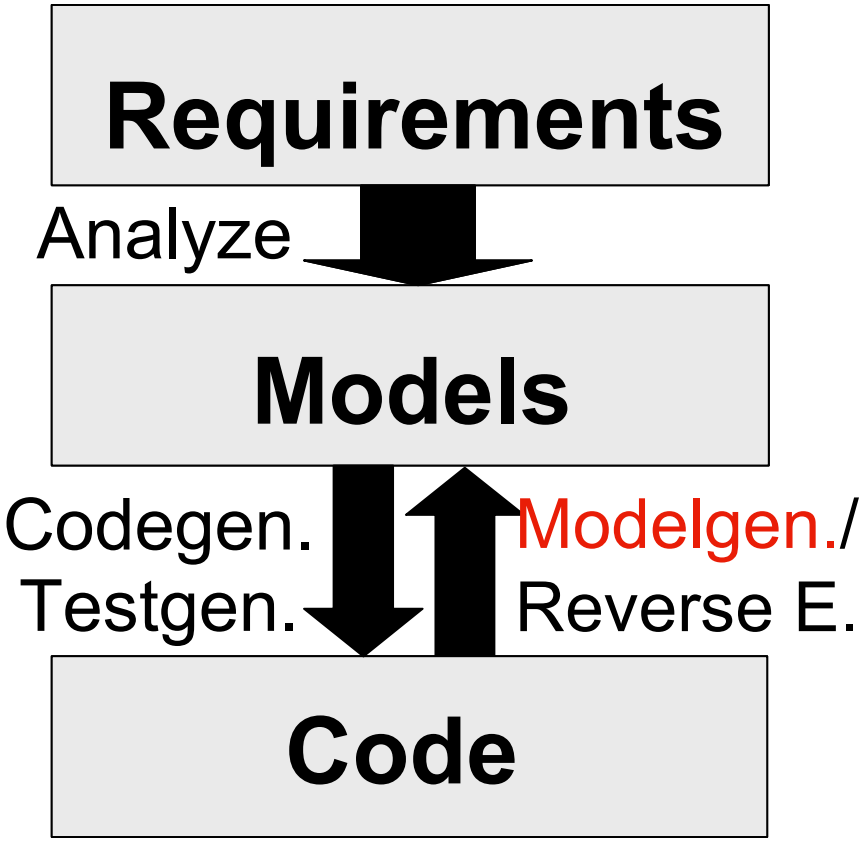
Increase security with bounded investment in time, costs:

- 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

- Analyze (UMLsec) models against security requirements.
- Generate code (or tests) from models.
- Generate models from evolving or legacy code.

Goal: model-based = source-code based.



Security Analysis of C-Programs

Goal: Logic-based security analysis of C programs which is as

- automatic and
- complete

as possible.

Note: can‘t be both perfectly automated and complete: Security in general undecidable.

Here: emphasize automation.

Abstraction

Enable efficient automated analysis by abstraction.

Use lookup tables for functions or code-blocks:

- symbolic representation of cryptographic or arithmetic routines
- technical infrastructure (packet_send, buffer_copy, …)
- data structures (e.g. a->b)

Factor out pointers usage.

Control Flow Graph

Use aicall (Absint, SB) to generate Mealy Machines from code:

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

where action can be outpattern or localvar:=value.

Loops: Abstract away or break up after maximal iteration number.

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,
- injects messages that it can produce in some links
- may access certain nodes.

Security Analysis in First-order Logic

Approximate set of possible data values flowing through system from above.

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

For any secret *s*, check whether can derive *knows(s)* using automated theorem prover.

First-order Logic: Basic Rules

Define *knows(E)* for any *E* initially known to the adversary.

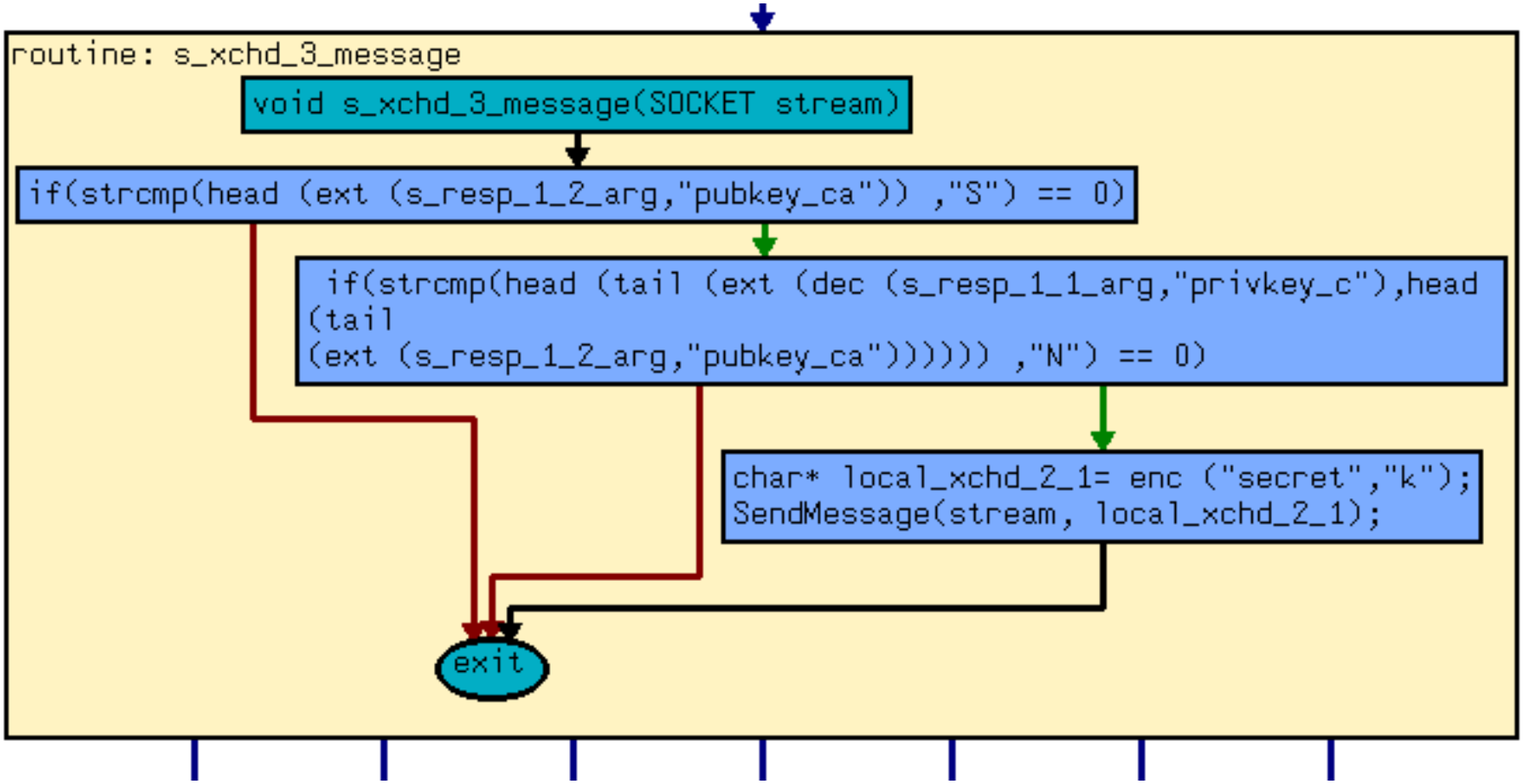
Define cryptosystem. E.g.: $Dec_{K^{-1}}(\{E\}_k) = E$

For evolving knowledge 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 Control Flow Diagram …



… Translate to 1st Order Logic

Graph transition

TR1=(in(msg_in),cond(msg_in),out(msg_out))

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

$\forall$ *msg_in. [knows(msg_in)Æ cond(msg_in)*

$\Rightarrow$*knows(msg_out)*

Æ PRED(TR2)]

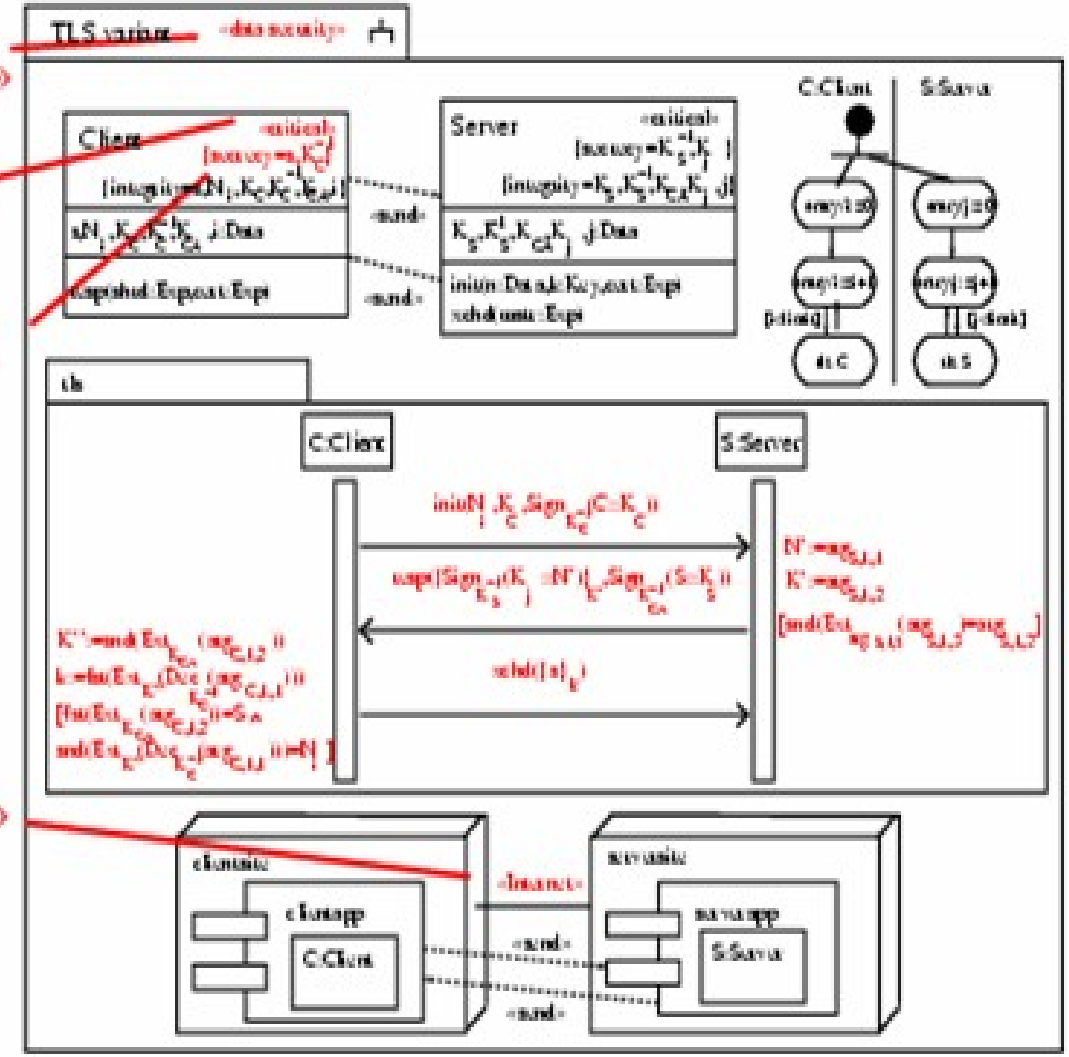
Abstraction (e.g. from senders, receivers): find all attacks, may have false positives.

Check whether can derive threat conjecture (e.g. *knows(s)* for a secret *s*) from axioms.

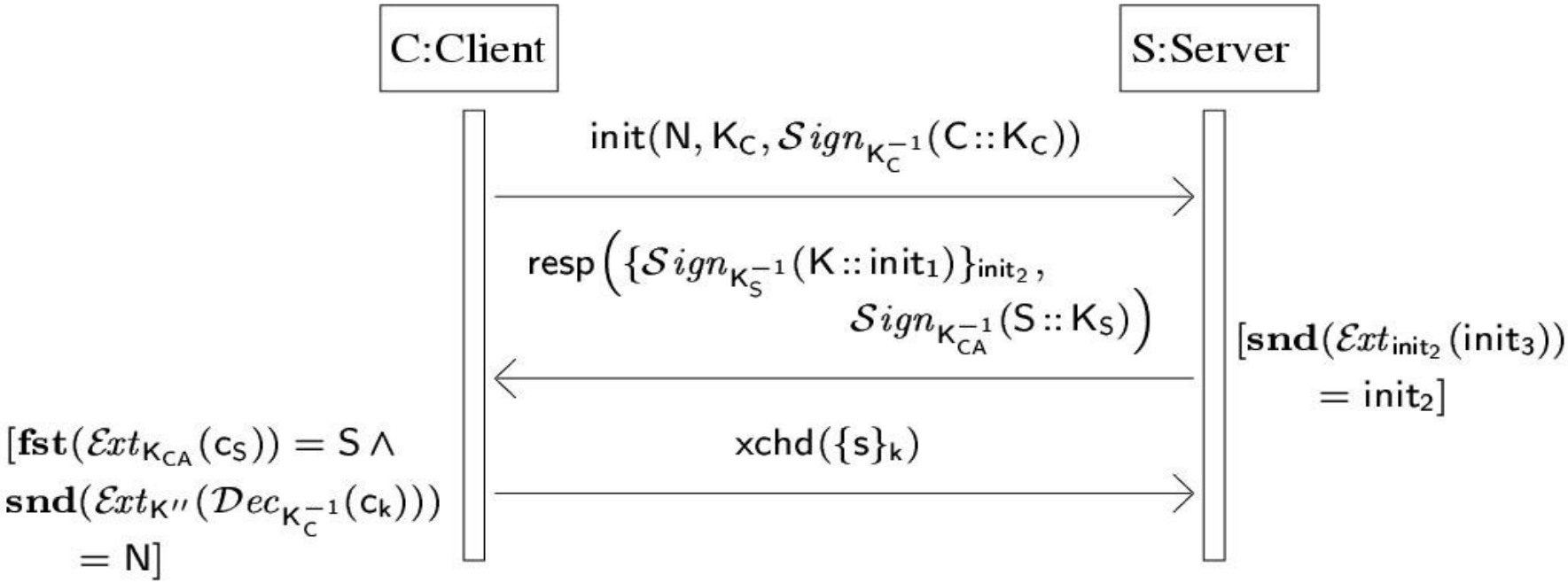
Example: Proposed Variant of TLS (SSL)

Presented at IEEE Infocom 1999.

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



Example: Translation to Logic



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

Surprise …

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

tlsvariant-freshkey-check.tptp
...
analyzing results ...
proof found
time limit information: 298 total / 297 strateg
...
e-SETHEO done. exiting
```

Attack

Can derive *knows(s)*.

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

➔ Completely insecure wrt stated goals.

But why ? Use prolog-based attack generator.

Man-in-the-Middle Attack

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

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

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

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

Biometric Authentication System

In development by large German 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).

Conclusions

Understanding Security Goals using First-Order-Logic:

- formally based approach
- automated, powerful tool support
- successful use in industrial projects

More information:
http://www4.in.tum.de/~juerjens

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

$\forall\forall E,K.Dec_K^{-1}(\{E\}_K)=E$

$\forall\forall E,K.Ext_K(Sign_K^{-1}(E))=E$

$\forall\forall E_1,E_2.head(E_1::E_2)=E_1$

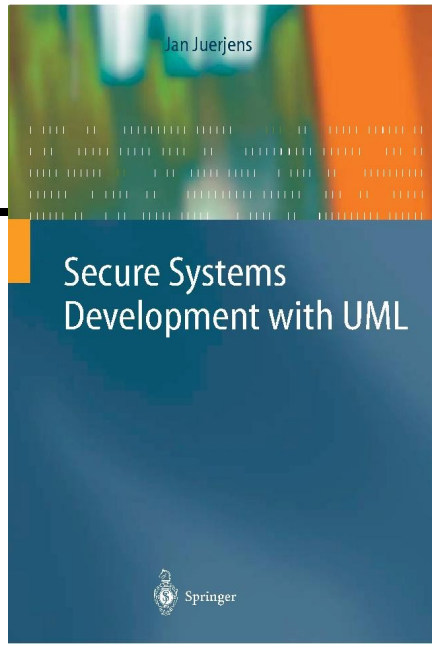
$\forall\forall E_1,E_2.tail(E_1::E_2)=E_2$

- 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, …).

Resources



TUM TB Dez. 2004 (mit T. Kuhn)

International Conference for Software Maintenance 2005

Memocode 2005

Models (UML) 2005 (mit S. Houmb)

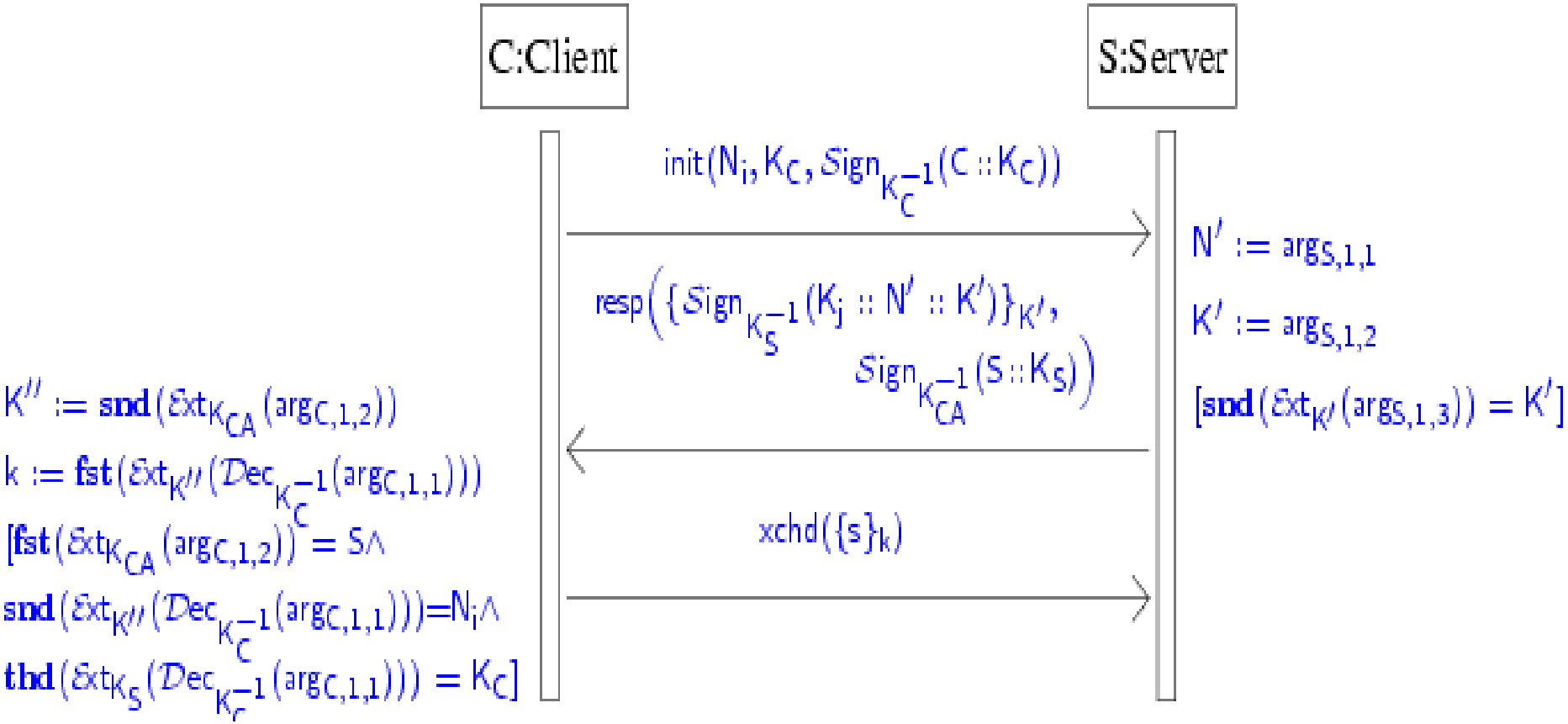
More information (papers, slides, tool, …): http://www.umlsec.org

Note: International Symposium on Secure Software Engineering (ISSSE 06 - IEEE)

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) ) ) )  )
```

The Fix



e-Setheo: *knows(s)* not derivable. Thus secure.

Industrial Application

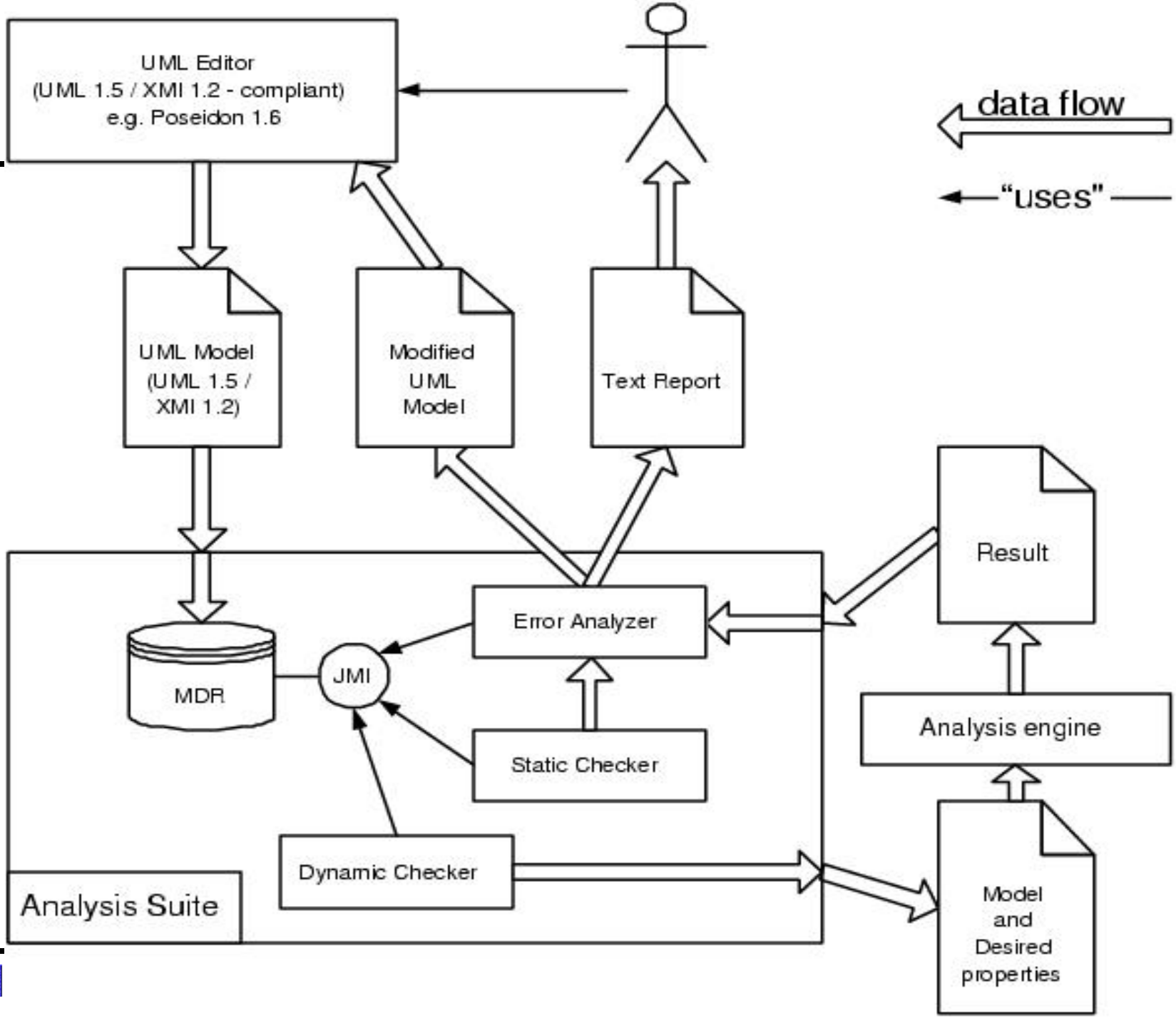
Verisoft project. Goal: Practical application of formal methods.

Planned for 8 years from 7/2003; 10 industrial + academic partners.

Integrated formal verification from application software down to operating system and prozessor on C-code level. Security-relevant:

- Biometric access control system
- Automotive emergeny application

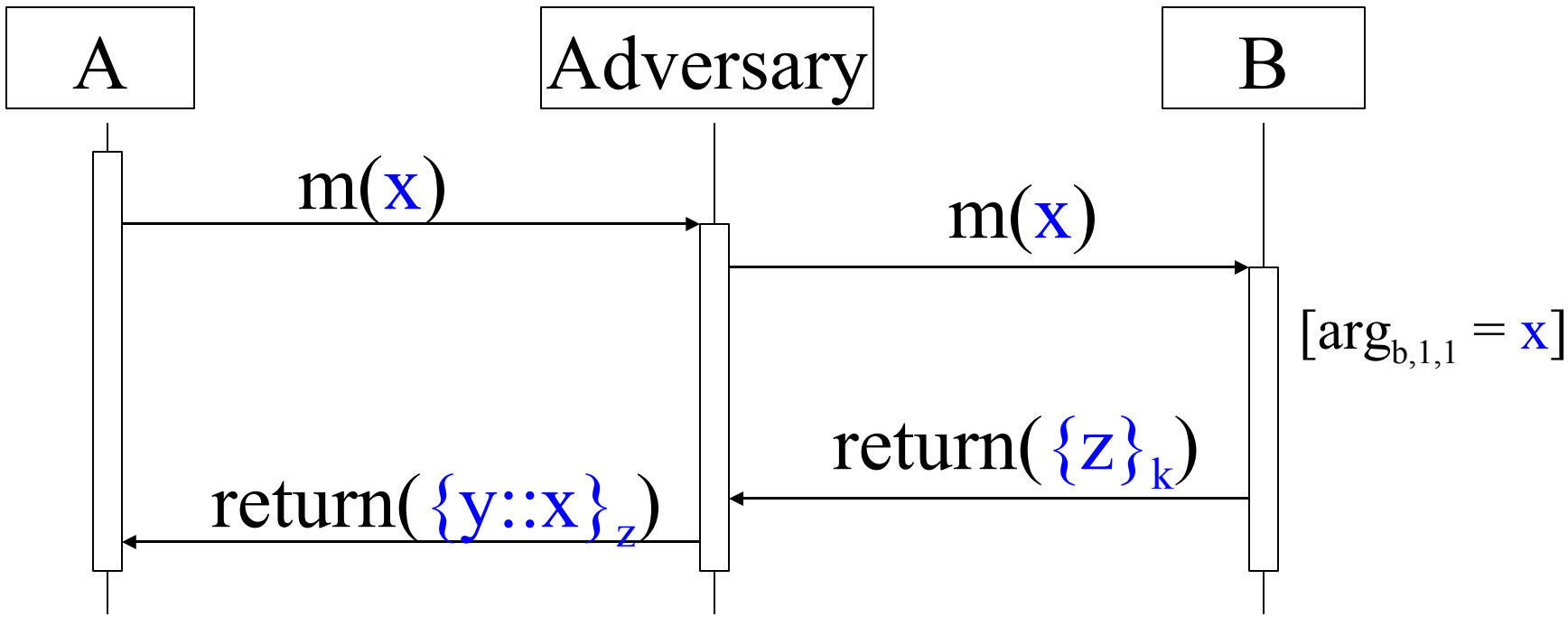
Apply UMLsec approach, tools.


UML Editor
(UML 1.5 / XMI 1.2 - compliant)
e.g. Poseidon 1.6
data flow
"uses"
UML Model
(UML 1.5 /
XMI 1.2)
Modified
UML
Model
Text Report
Result
Error Analyzer
MDR
JMI
Static Checker
Analysis engine
Dynamic Checker
Analysis Suite
Model
and
Desired
properties

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)) )  )
) )  )).
```

Adversary: Simulation



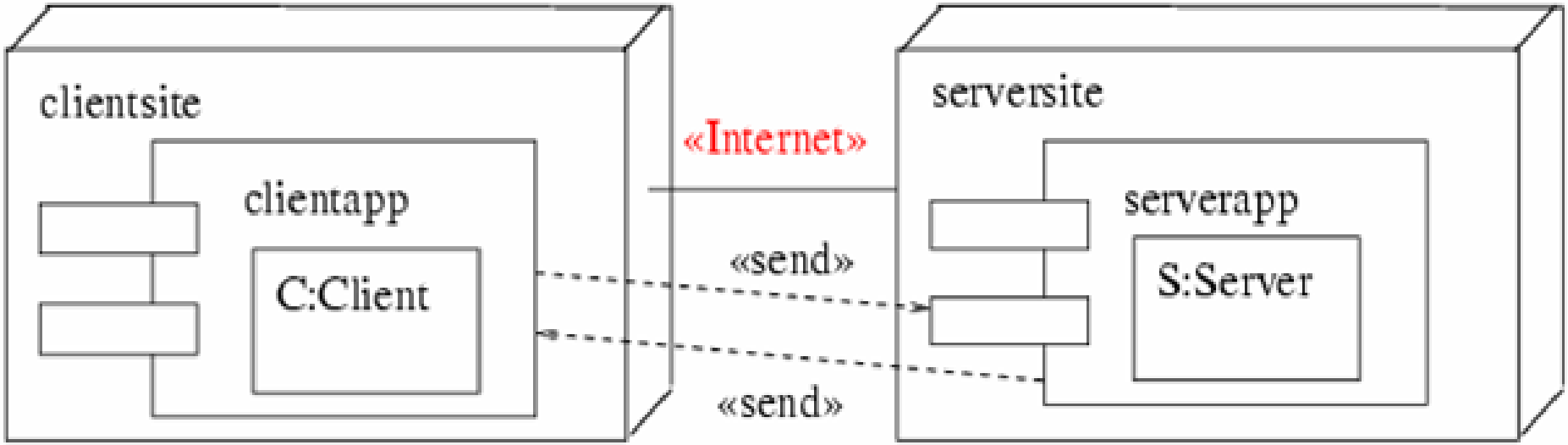
Adversary knowledge:

k^{-1}, y, x

$\{z\}_k$, z

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

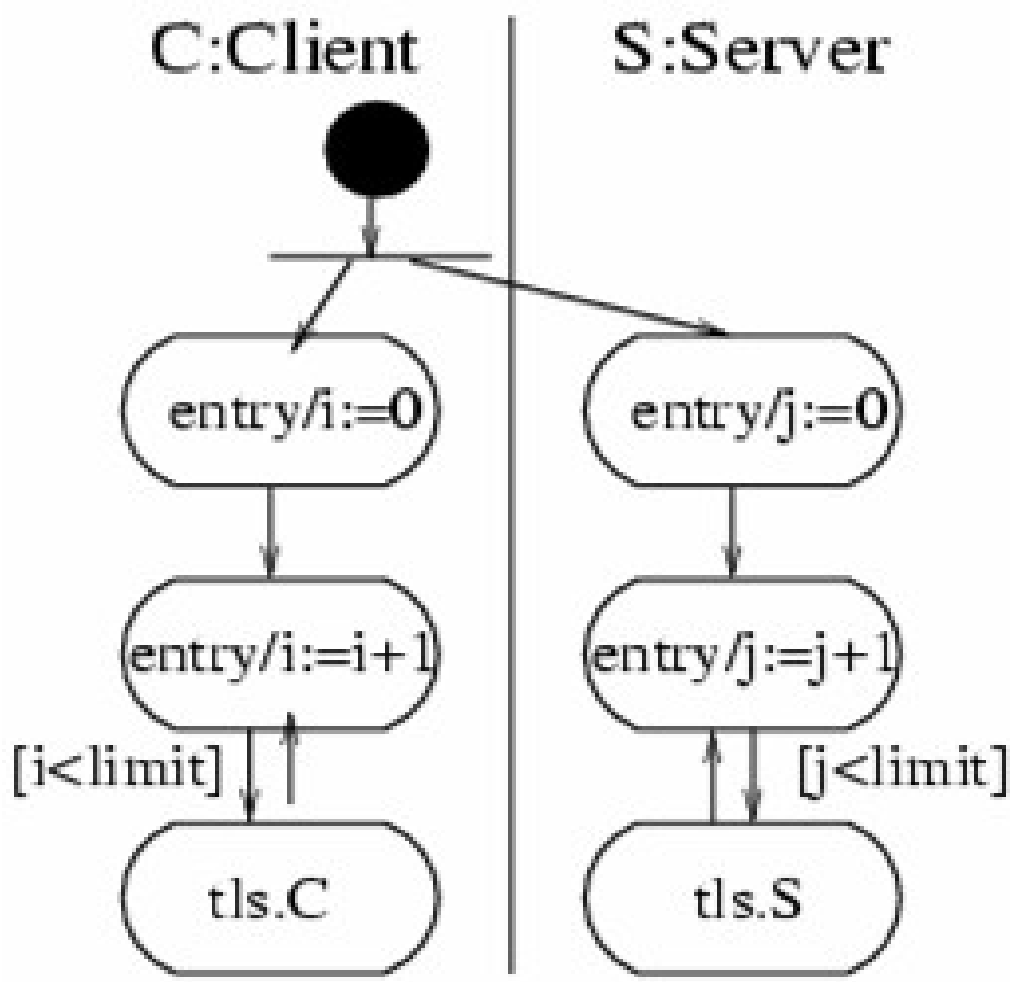
… and Physical Layer Model …



Deployment diagram.

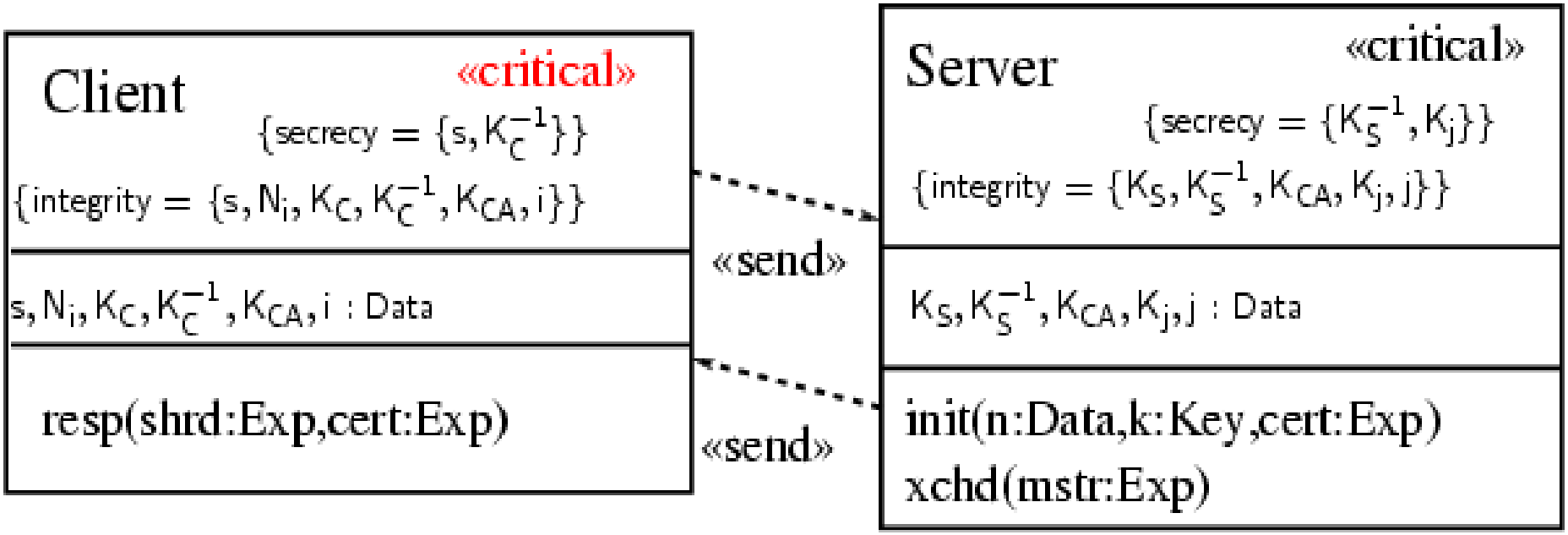
Derived adversary model: read, delete, insert data.

Execute in System Context



Activity diagram.

Formulate Data Security Requirements



Class diagram.

Gives conjecture: *knows(s)* derivable ?

Application: OpenSSH

Apply to OpenSSH handshake.

Can find security flaws from earlier version.

```
/* #ATP# state(1121,ssh_client_key_with_diffie_hellman) */
/* #ATP# set(1121,1122,g_c,generator(c),-) */
/* #ATP# set(1122,1123,p_c,getmodulus(c),-) */
/* #ATP# set(1123,1124,e_c,dh(g_c,x_c),-) */
/* #ATP# io(1124,1125,-,e_c,-) */
/* #ATP# io(1125,1126,ks,-,-) */
/* #ATP# set(1126,1127,k_c,key(f_c,x_c),-) */
/* #ATP# cond(1127,verify_host_key(k_c)==true,1128,0,-) */
/* #ATP# io(1128,1129,f,-,-) */
/* #ATP# io(1129,1130,sig,-,-) */
/* #ATP# set(1130,1131,k,dhcomputekey(f),-) */
/* #ATP# set(1131,1132,hh_c,hash(conc(verc_c,conc(vers_c,conc(
            kexinitc_c,conc(kexinits_c,conc(f_c,conc(e_c,k_c))))))),-) */
/* #ATP# cond(1132,verify(sig,inv(ks),hh_c)==true,1133,0,-)
```

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

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.

Formal semantics for UML: How

Diagrams in context (using subsystems).

Model actions and internal activities explicitly.

Message exchange between objects or components (incl. event dispatching).

For UMLsec: include adversary arising from threat scenario in deployment diagram.

Use Abstract State Machines (pseudo-code).

Adversaries

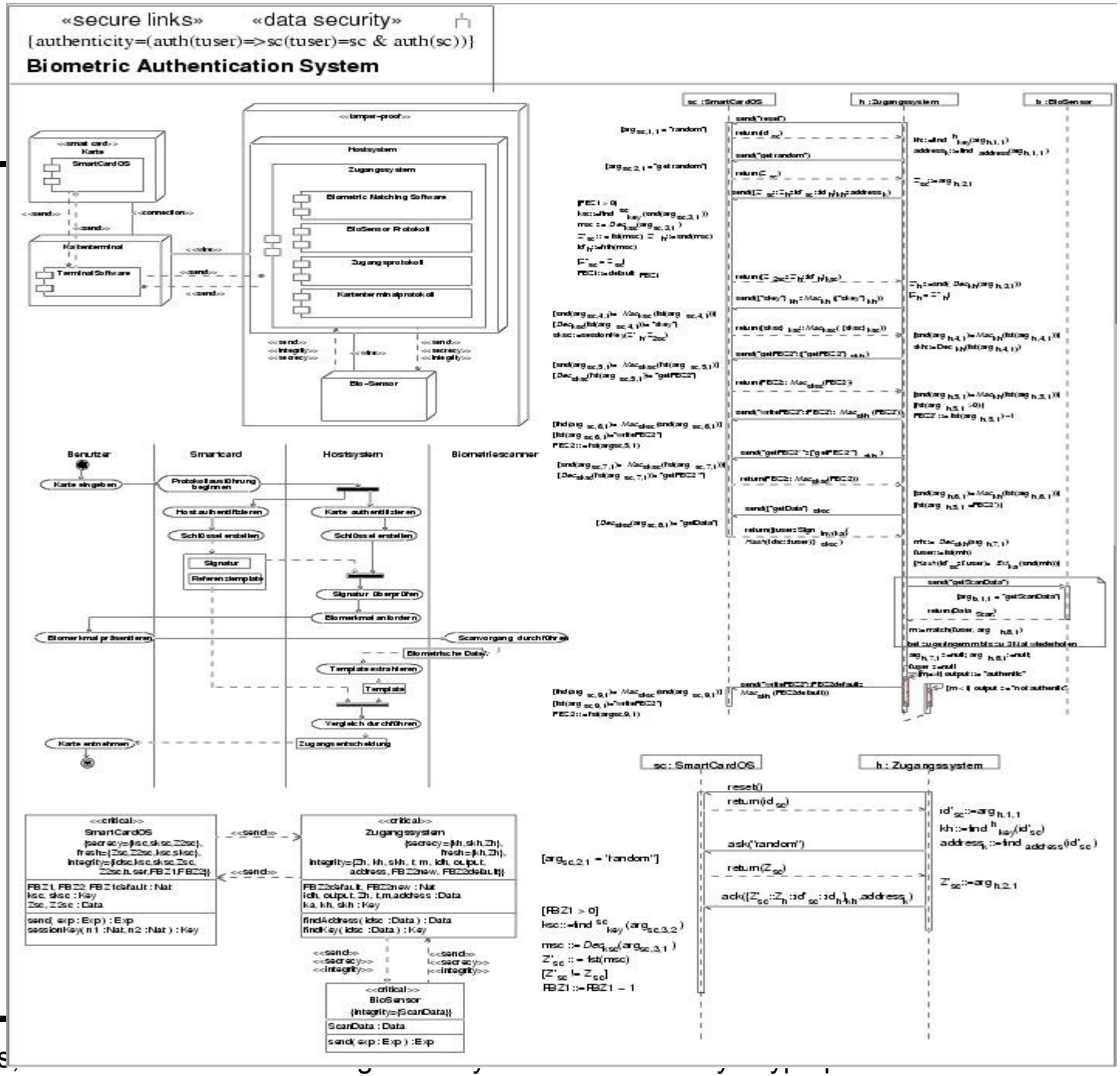
Model classes of adversaries.

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, verify against adversary model.

Big Picture


«secure links» «data security»
{authenticity=(auth(tuser)=>sc(tuser)=sc & auth(sc))}
Biometric Authentication System
Hostsystem
Zugangssystem
Biometric Matching Software
BioSensor Protokoll
Zugangsprotokoll
Kartenterminalprotokoll
Bio-Sensor
Kartenterminal
Terminal Software
SmartCardOS
Karte
Benutzer
Smartcard
Hostsystem
Biometriescanner
Karte eingeben
Protokollausführung beginnen
Host authentifizieren
Karte authentifizieren
Schlüssel erstellen
Signatur
Referenztemplate
Signatur überprüfen
Biomerkmal anfordern
Biomerkmal präsentieren
Scanvorgang durchführen
Biometrische Daten
Template extrahieren
Template
Vergleich durchführen
Zugangsentscheidung
Karte entnehmen
sc : SmartCardOS
h : Zugangssystem
b : BioSensor
sc : SmartCardOS
h : Zugangssystem
reset()
return(id_sc)
ask('random')
return(Z_sc)
ack([Z'_sc::Z_h::id'_sc::id_h,k_h,address_k])
[arg_sc,2,1 = 'random']
[RBZ1 > 0]
msc ::= Dec_k,sc(arg_sc,3,1)
Z'_sc ::= fst(msc)
[Z'_sc != Z_sc]
RBZ1 ::= RBZ1 - 1
id'_sc::=arg_h,1,1
kh::=find h_key(id'_sc)
address_k::=find address(id'_sc)
Z'_sc::=arg_h,2,1
«critical»
SmartCardOS
{secrecy={ksc,skscZsc},
fresh={Zsc,Zsc,ksc,skscl},
integrity={idsc,ksc,skscZsc,
Zsc,b,ser,RBZ1,RBZ2}}
RBZ1, RBZ2, RBZ1default : Nat
ksc, skscl : Key
Zsc, Zsc : Data
send(exp : Exp) : Exp
sessionKey(n1 :Nat, n2 :Nat) : Key
«critical»
Zugangssystem
{secrecy={kh,skh,Zh},
fresh={kh,Zh},
integrity={Zh, kh, skh, t, m, idh, output,
address, RBZ2new, RBZ2default}}
RBZ2default, RBZ2new : Nat
idh, output, t, Zh, t,m,address : Data
ka, kh, skh : Key
findAddress(idsc :Data) : Data
findKey(idsc :Data) : Key
«critical»
BioSensor
{integrity={ScanData}}
ScanData : Data
send(exp : Exp) : Exp
«send»
«secrecy»
«integrity»

Crypto Protocol Part 2: Problem ?

sksc::=sessionKey(Z' $_h$,Z_{2sc})

[snd(arg $_{sc,5,1}$)= Mac_{sksc} (fst(arg $_{sc,5,1}$))]

[Dec_{sksc}(fst(arg $_{sc,5,1}$))= "getFBZ2"]

[thd(arg $_{sc,6,1}$)= Mac_{sksc} (snd(arg $_{sc,6,1}$))]

[fst(arg $_{sc,6,1}$)="writeFBZ2"]

FBZ2 ::= fst(argsc,5,1)

[Dec_{sksc}(arg$_{sc,8,1}$)= "getData"]

Send reference template and signature to host system

ksc'' ksc ksc''

9:send("getFBZ2"::{"getFBZ2"} $_{skh}$)

10:return(FBZ2:: Mac_{sksc}(FBZ2))

11:send("writeFBZ2"::FBZ2':: Mac_{skh} (FBZ2'))

14:send({"getData"} $_{sksc}$

15:return({tuser::Sign $_{inv(ka)}$(

Hash(idsc::tuser))} $_{sksc}$)

[snd(arg $_{h,4,1}$)= Mac_{kh}(fst(arg $_{h,4,1}$))]

skh::=Dec $_{kh}$(fst(arg $_{h,4,1)}$)

[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

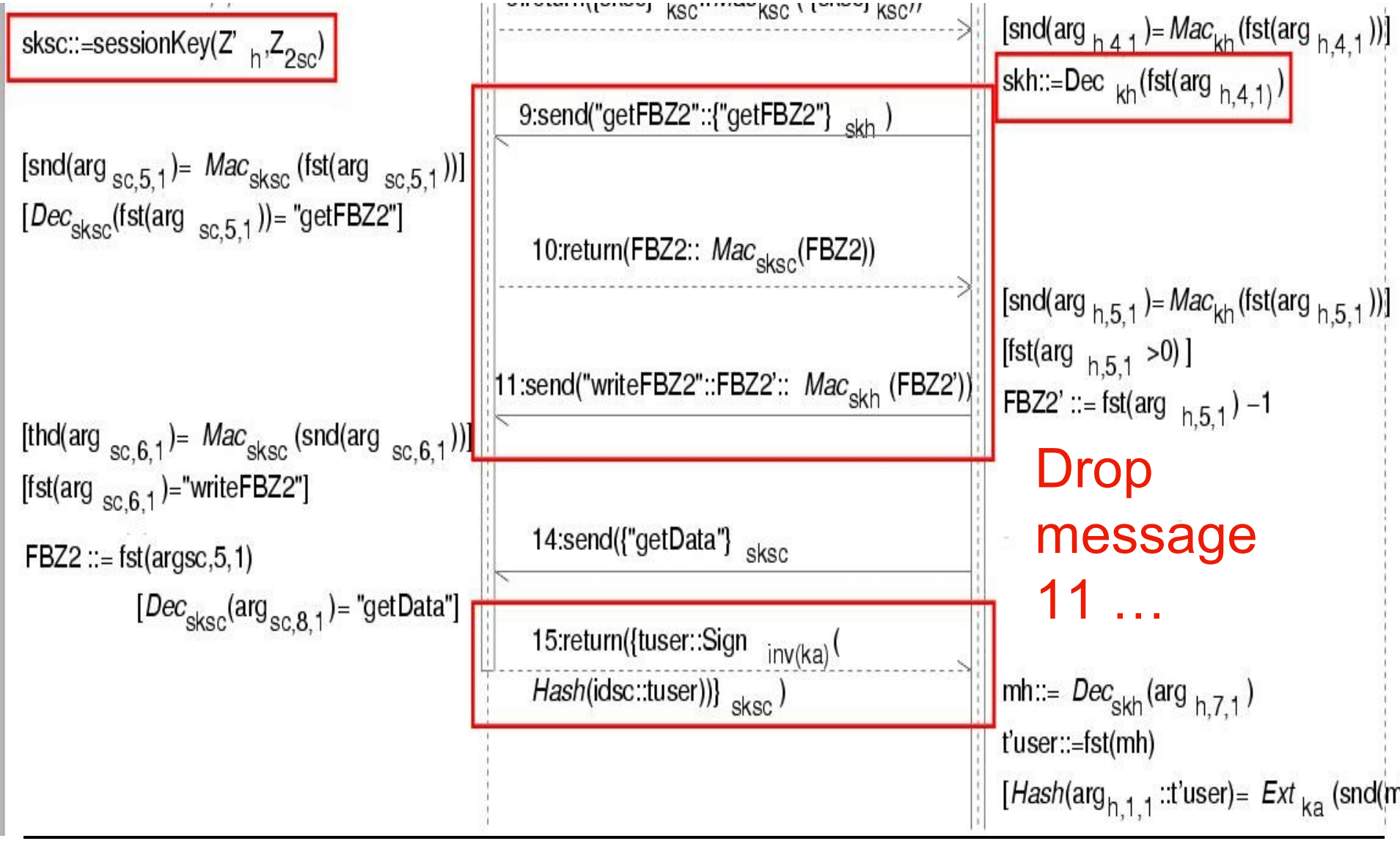
Decrease misuse counter

mh::= Dec_{skh}(arg $_{h,7,1}$)

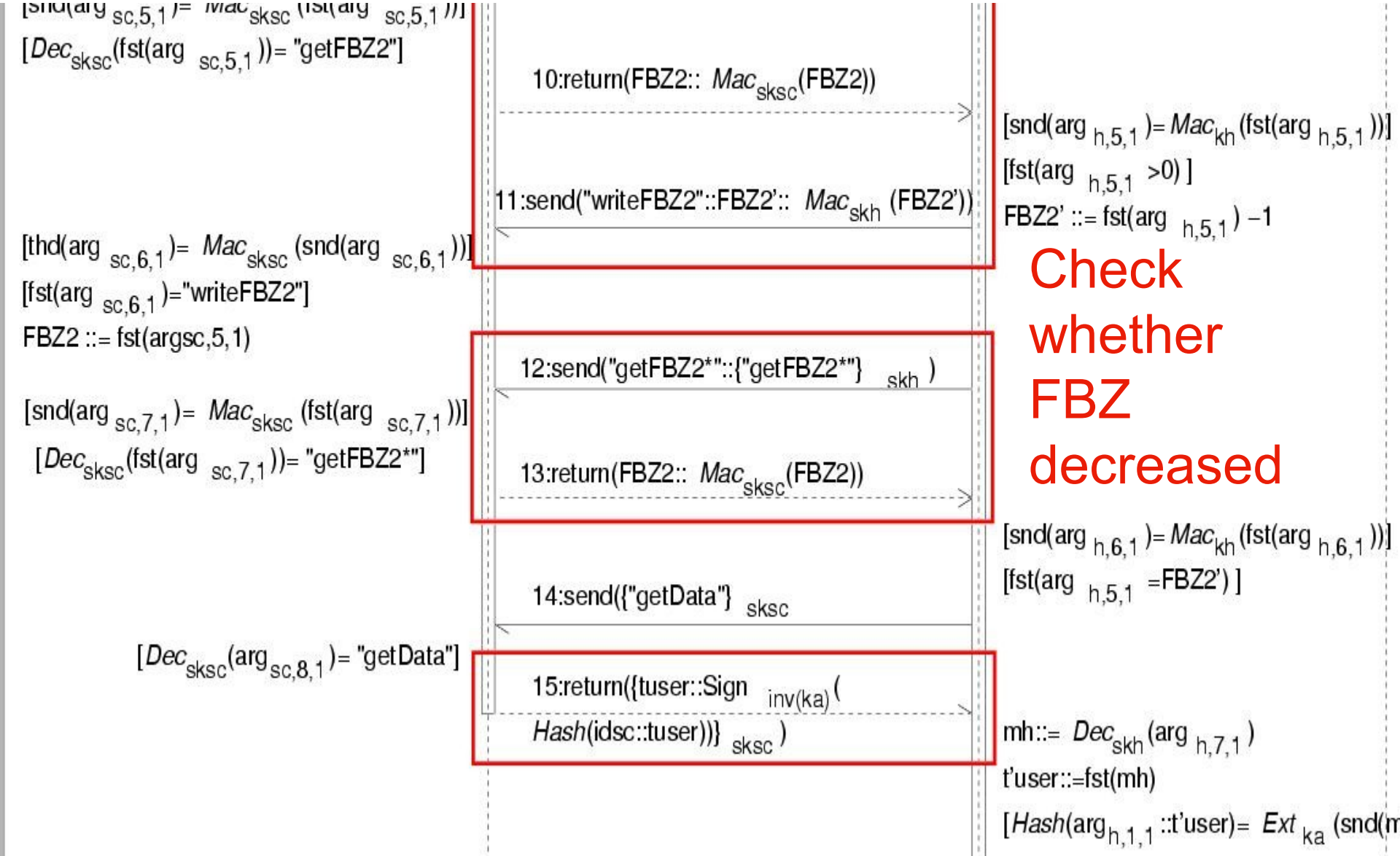
t'user::=fst(mh)

[*Hash*(arg$_{h,1,1}$ ::t'user)= Ext_{ka} (snd(mh

Crypto Protocol Part 2: Problem

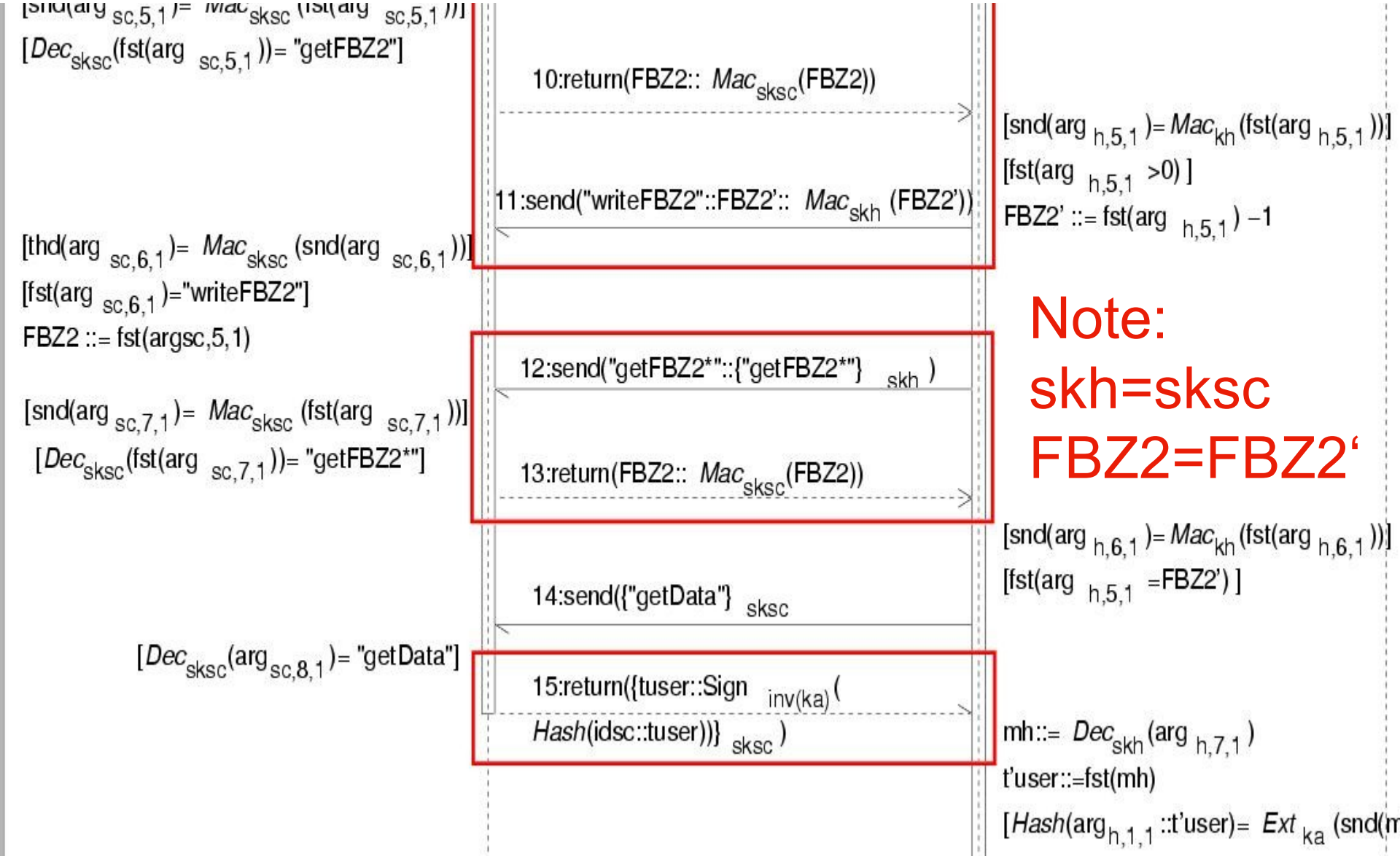


Crypto Protocol Part 2: Improvement



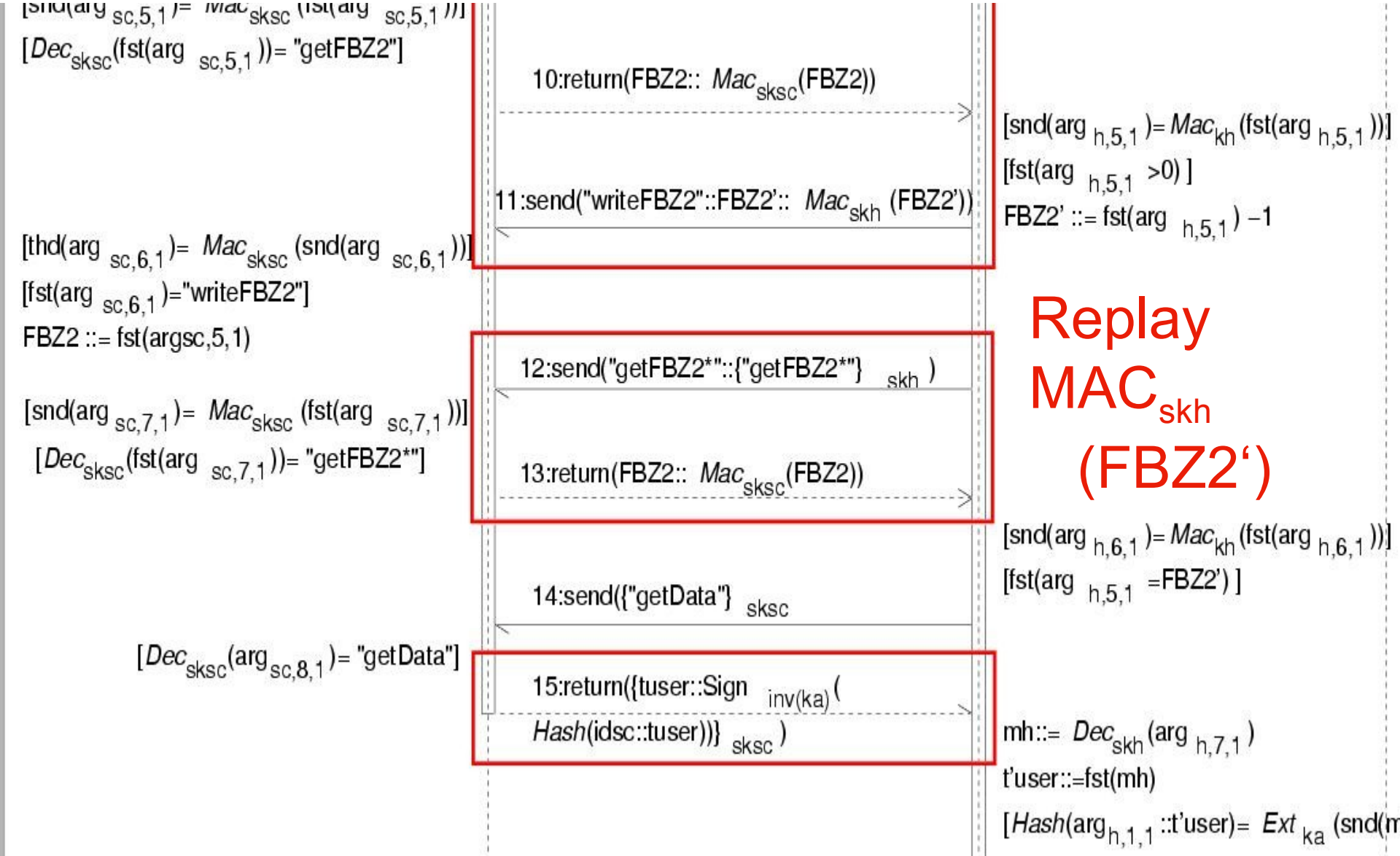
Check whether FBZ decreased

Crypto Protocol Part 2: Improvement ?



Note:
skh=sksc
FBZ2=FBZ2‘

Crypto Protocol Part 2: Problem



CEPS: Load Protocol

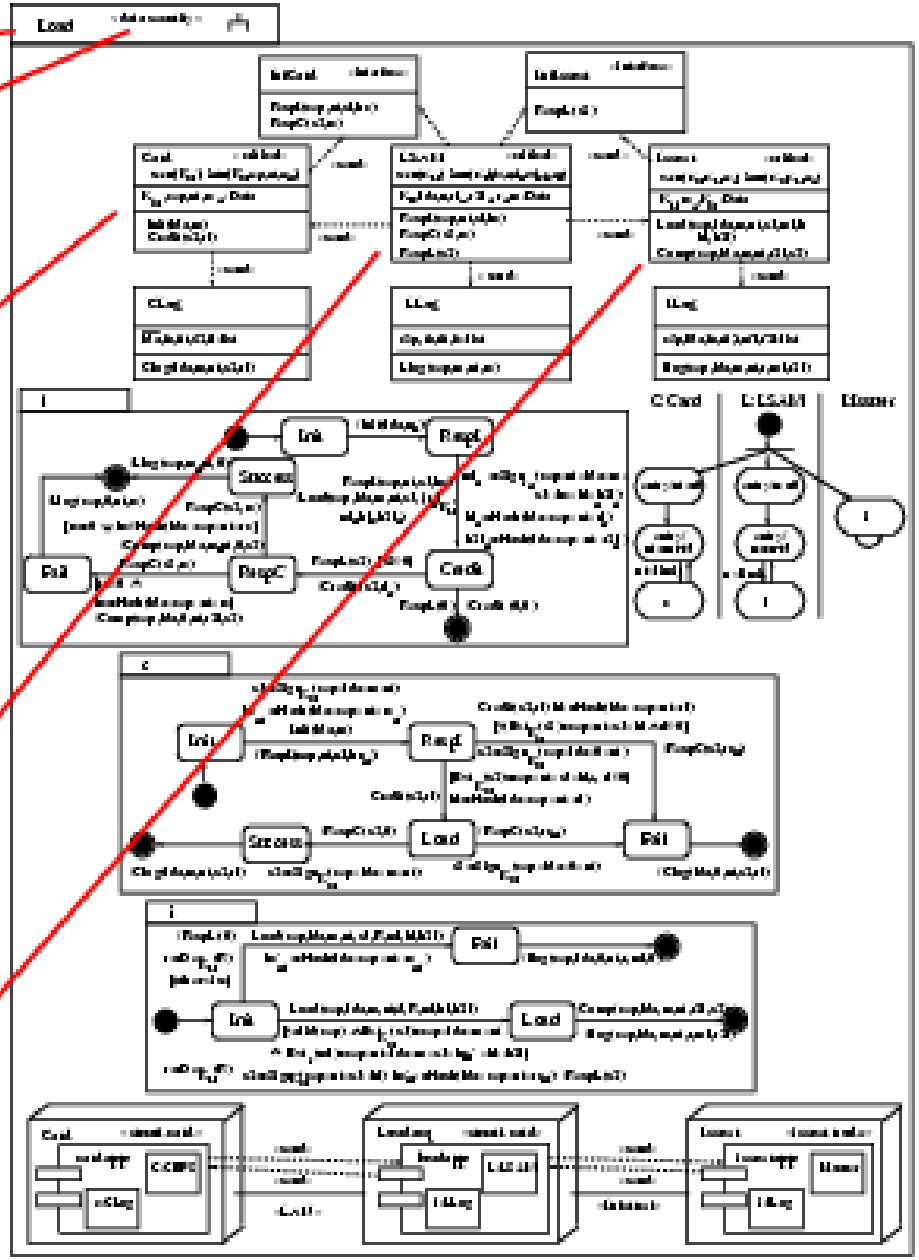
Load

«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,tl)

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

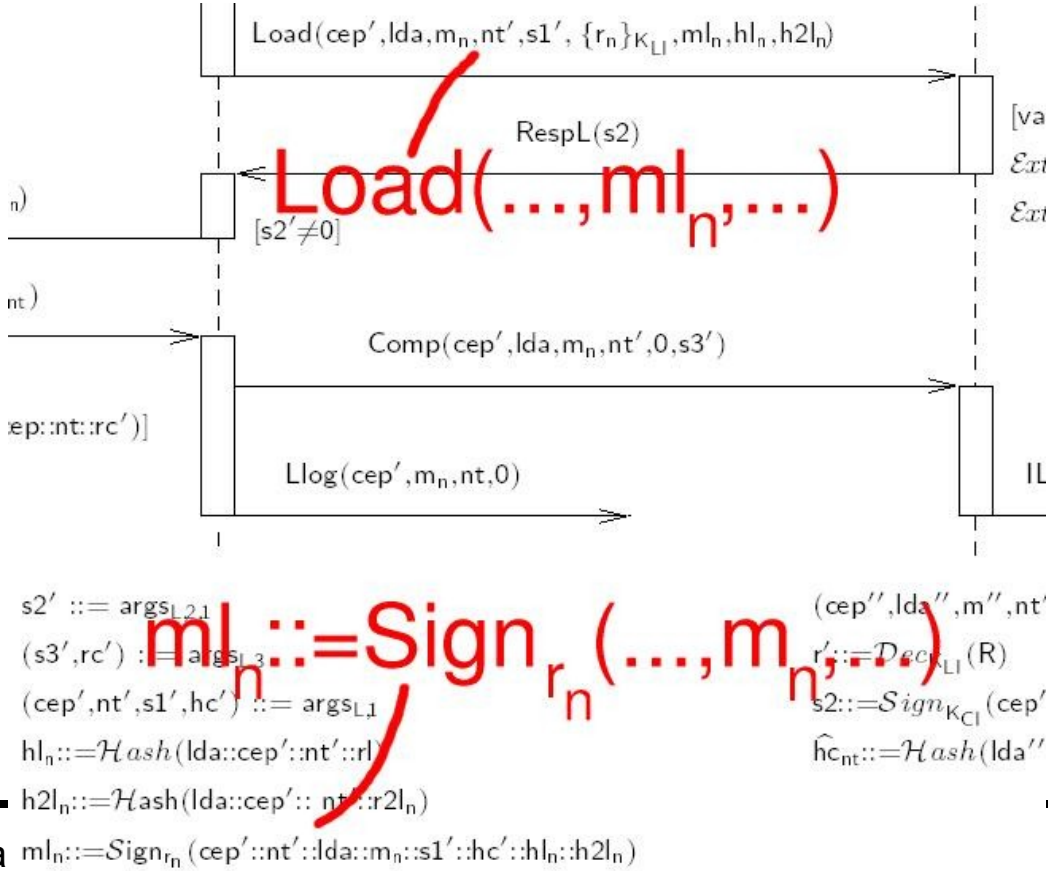
Issuer «critical» {secrecy={K_{CI},K_{LI},rc_{nt}}} {integrity={K_{CI},K_{LI},rc_{nt}}}
rc_{nt} : Data; K_{LI},K_{CI} : Keys
Load(cep,lda,m,nt,s1,ml,h hl,h2l) Comp(cep,lda,m,nt,r2l,s3)



CEPS

CEPS Load and Purchase Protocol (three weaknesses)

- ml_n: „Proof“ for bank that loading machine received money
- But: r_n shared between bank, loading machine
- Theorem: Corrected version secure.



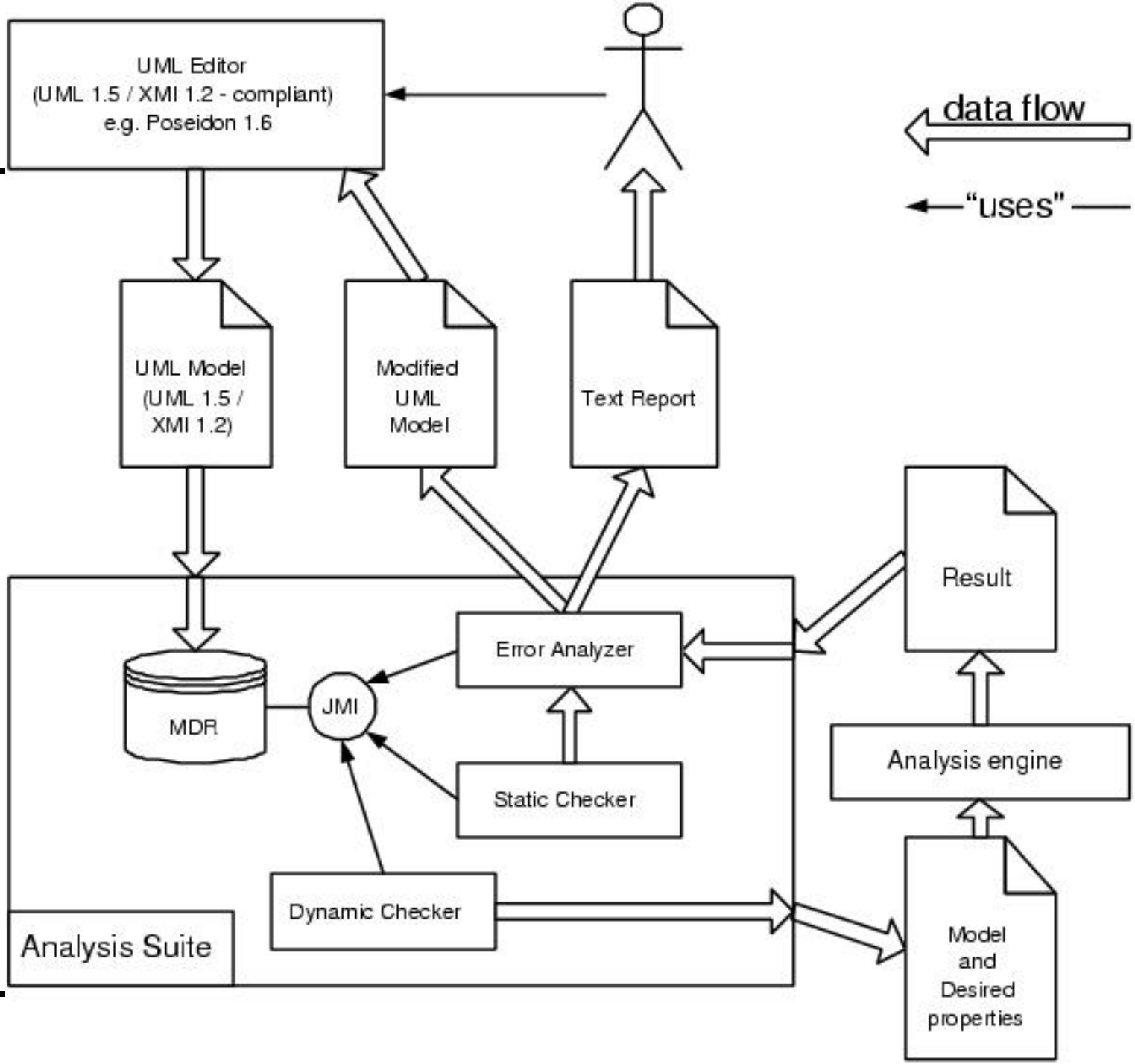
CSDUML Framework

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://www4.in.tum.de/~umlsec .

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


UML Editor
(UML 1.5 / XMI 1.2 - compliant)
e.g. Poseidon 1.6
data flow
"uses"
UML Model
(UML 1.5 /
XMI 1.2)
Modified
UML
Model
Text Report
Result
Error Analyzer
MDR
JMI
Static Checker
Analysis engine
Dynamic Checker
Analysis Suite
Model
and
Desired
properties

Beyond Specification Analysis

Model-based test generation.

Configuration analysis.

- Analyze permission data using Prolog (e.g. SAP R/3). Integrated in CSDUML framework.
- Analyze firewall configurations using model-checkers

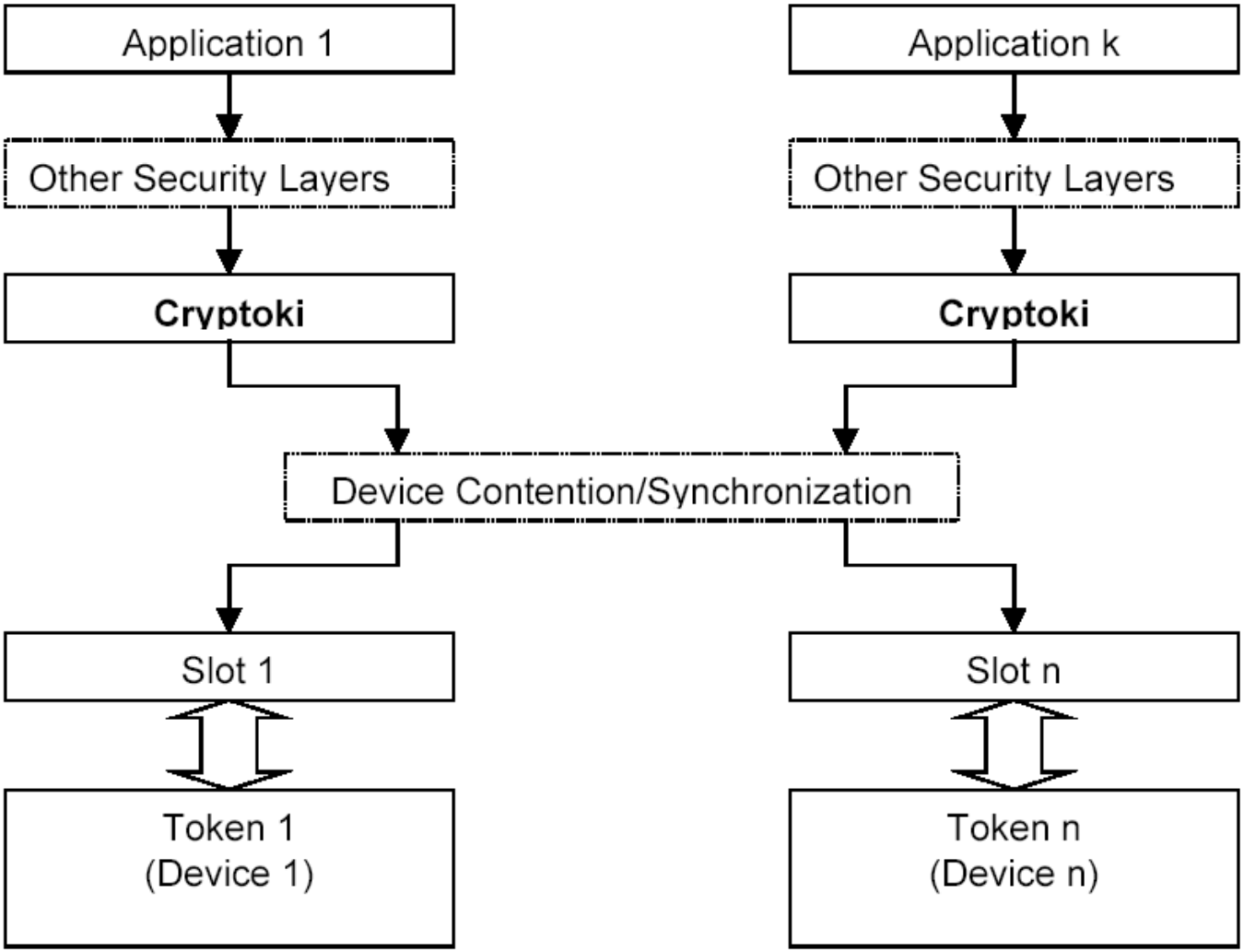
Source-code analysis (C).

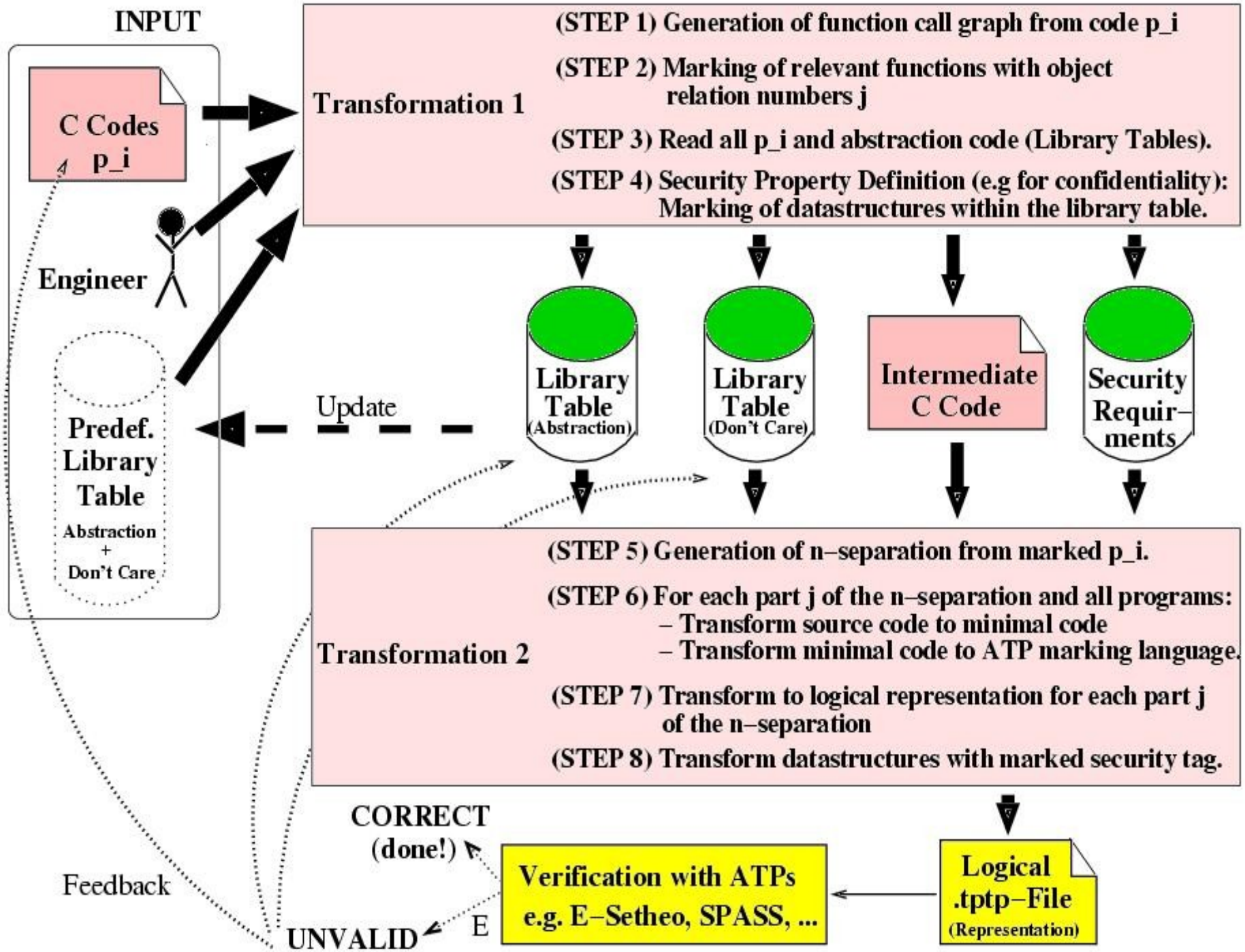
$\mathsf{enc}_S(E, E')$	(encryption)
$\mathsf{dec}_S(E, E')$	(decryption)
$\mathsf{hash}_S(E)$	(hashing)
$\mathsf{sign}_S(E, E')$	(signing)
$\mathsf{ver}_S(E, E', E'')$	(verification of signature)
$\mathsf{kgen}_S(E)$	(key generation)
$\mathsf{inv}_S(E)$	(inverse key)
$\mathsf{conc}_S(E, E')$	(concatenation)
$\mathsf{head}_S(E)$ and $\mathsf{tail}_S(E)$	(head and tail of concat.)

```
/* C_Encrypt encrypts single-part data. */
CK_PKCS11_FUNCTION_INFO(C_Encrypt)
#ifdef CK_NEED_ARG_LIST
(
CK_SESSION_HANDLE hSession,          /*session handle*/
CK_BYTE_PTR       pData,             /*plaintext data*/
CK_ULONG          ulDataLen,         /*plaintext bytes*/
CK_BYTE_PTR       pEncryptedData,    /*gets ciphertext*/
CK_ULONG_PTR      pulEncryptedDataLen/*gets ctext size*/
);
#endif
```

```
input_formula(protocol,axiom,(
 ![S,Init_1, Init_2, Init_3, Resp_1, Resp_2, Xchd_1] : (
 % C -> Attacker
( ( ( true & true )
      => knows(conc(S, n,  conc(S, k_c,  sign(S, conc(S, c, conc(S, k_c, eol)), inv(S, k_c)) ) ))
     & ( (knows(Resp_1) & knows(Resp_2)
         & equal( fst(S, ext(S, Resp_2, k_ca)), s) & equal(snd(S, ext(S, dec(S, Resp_1, inv(S, k_c)),
              snd(S, ext(S, Resp_2, k_ca)))), n ) )
        => knows(enc(S, secret, fst(S, ext(S, dec(S, Resp_1, inv(S, k_c)), snd(S, ext(S, Resp_2, k_ca))))))
       ) ) )
 &  % S -> Attacker
( ( ( knows(Init_1) & knows(Init_2) & knows(Init_3)
       & equal( snd(S, ext(S, Init_3, Init_2)), Init_2 ) )
      => knows(conc(S, enc(S, sign(S, conc(S, kgen(S, Init_2), conc(S, Init_1, eol)), inv(S, k_s)), Init_2),
            sign(S, conc(S, s, conc(S, k_s, eol)), inv(S, k_ca) ) ))
     & ( ( knows(Xchd_1)
         & true )
        => true ) ) ) ) ) ).
```

Function	Description
C_GetAttributeValue	obtains attribute value
C_SetAttributeValue	modifies attribute value
C_Encrypt	encrypts single-part data
C_Decrypt	decrypts encrypted data
C_Digest	digests single-part data
C_Sign	signs single-part data
C_VerifyRecover	verifies a signature where the data is recovered
C_GenerateKey	generates a secret key
C_GenerateKeyPair	generates a key pair
C_GenerateRandom	generates random data


Application 1
Application k
Other Security Layers
Other Security Layers
Cryptoki
Cryptoki
Device Contention/Synchronization
Slot 1
Slot n
Token 1
(Device 1)
Token n
(Device n)


INPUT
C Codes
p_i
Engineer
Predef.
Library
Table
Abstraction
+
Don't Care
Transformation 1
(STEP 1) Generation of function call graph from code p_i
(STEP 2) Marking of relevant functions with object relation numbers j
(STEP 3) Read all p_i and abstraction code (Library Tables).
(STEP 4) Security Property Definition (e.g for confidentiality): Marking of datastructures within the library table.
Update
Library
Table
(Abstraction)
Library
Table
(Don't Care)
Intermediate
C Code
Security
Requir-
ments
Transformation 2
(STEP 5) Generation of n-separation from marked p_i.
(STEP 6) For each part j of the n-separation and all programs:
– Transform source code to minimal code
– Transform minimal code to ATP marking language.
(STEP 7) Transform to logical representation for each part j of the n-separation
(STEP 8) Transform datastructures with marked security tag.
CORRECT
(done!)
Feedback
UNVALID
E
Verification with ATPs
e.g. E-Setheo, SPASS, ...
Logical
.tptp-File
(Representation)

Annotated Code

```
a = b          a[i] = b
```

```
n:    set(n,n+1,a,b,–)
n+1:  ...
```

```
n:    set(n,n+1,a(i),b,–)
n+1:  ...
```

```
{ <line1>; <line2>; ... }
```

```
n:    <atpmarking of line 1>
n+1:  <atpmarking of line 2>
      ...
```

```
if (a==b) then { ... } else { ... }
```

```
n:  cond(n,a==b,i,j,–)
i:  ...
j:  ...
```

```
while (a==b) do { ... }
```

```
n:    cond(n,a==b,i,n+1,–)
i:    ...
n+1:  ...
```

```
f(a, b, ..., z);
```

```
n:     set(n,n+1,locala,a,–)
n+1:   set(n+1,n+2,localb,b,–)
       ...
n+26:  set(n+26,i,localz,z,–)

z:     <atpmarking of function start>
```

```
void simplecall(char *shared_secret, char *kbuf, int klen, char *secret)
/* #ATP# state(1,2) */
{
   char *signature;
   int slen;

   /* read input */
   in_init();
   in_read(server_host_key);
   /* #ATP# io(2,3,serverhostkey,-,-) */
   in_read(signature);
   /* #ATP# io(3,4,sig),-,-) */
   in_read(slen);

   /* compute hash of server_host_key and length with shared secret */
   tmp_hash = simple_crypto_hash(
      server_host_key,    /* server public key (ks) */
      shared_secret   );  /* shared secret for the crypto hash (k) */
   /* #ATP# set(4,5,tmphash,hash(conc(serverhostkey,sharedsecret)),-) */

   /* check signature where hash is the signed value */
   if (key_verify(server_host_key, signature, slen, tmp_hash, 20) == false)
      fatal("key_verify failed for server_host_key");
   /* #ATP# cond(5,verify(sig,serverhostkey,tmphash)==false,0,6,-) */

   /* send out data 'secret' encrypted with server_host_key
   out_init();
   out_write(DES_encryption(secret,server_host_key));
   /* #ATP# io(6,7,-,enc(secretdata,serverhostkey),-) */

   out_close();
   in_close();
   xfree(kbuf);
}
```

Translation to Logic

Similar as explained earlier.

Loops: Abstract away or break up after maximal iteration number.

Example:

```
(true & true => true % state 1
  & (knows(serverhostkey) & true => true % state 2
    & (knows(sig) & true => true % state 3
      & (true & tmphash = hash(conc(serverhostkey,sharedsecret)) => true % state 4
        & (true & not(verify(sig,serverhostkey,tmphash)=false) => true % state 5
          & (true & true => knows(enc(secretdata,serverhostkey))))))) % state 6
```