Code Security Analysis of a Biometric Authentication System Using Automated Theorem Provers

Jan Jürjens

Software & Systems Engineering

Technical University of Munich

juerjens@in.tum.de

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

Security Analysis of C Programs

Logic-based program understanding of crypto protocols in C which is as

- automatic and
- complete

as possible.

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

Abstract and approximate safely.

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.

Abstraction, Preprocessing

Enable efficient automated analysis by abstraction (e.g. 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. Transform to SSA. Eliminate side effects.

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.

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

First-order Logic: Basic Rules

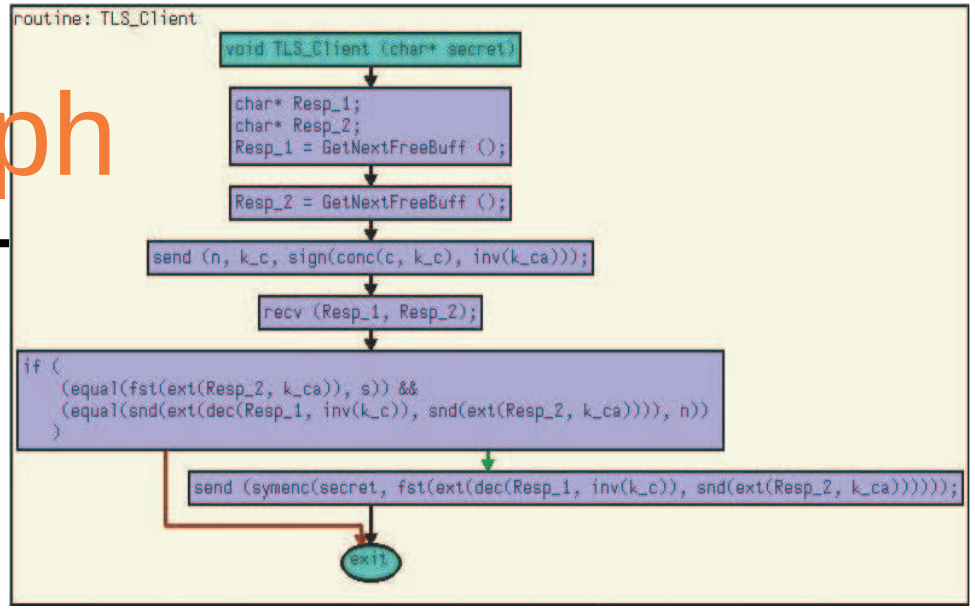
Define *knows(E)* for any *E* initially known to the adversary.

For evolving knowledge define

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

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

Control Flow Graph



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

Transform to state machine:

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

where action can be outpattern or localvar:=value.

… Translate to First Order Logic

Graph transition

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

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

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

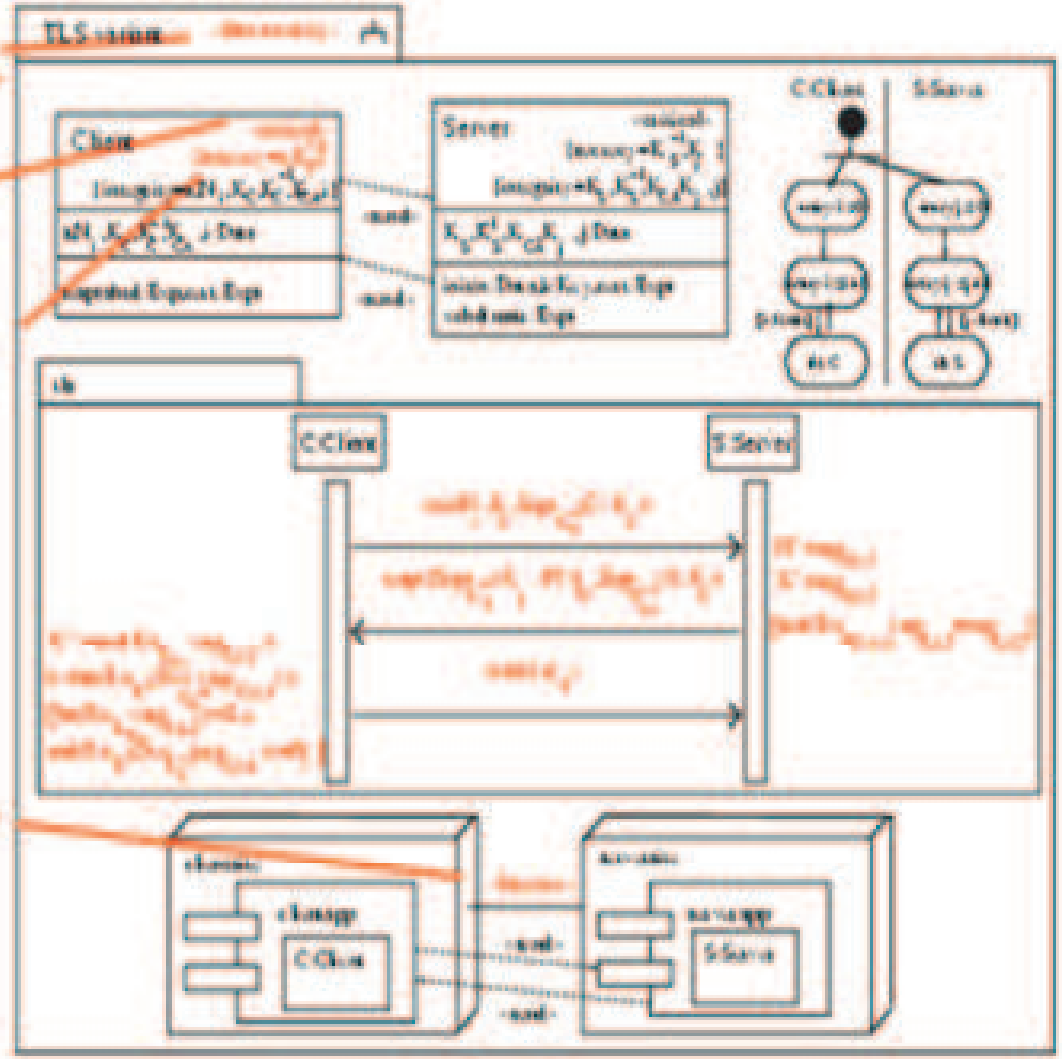
Abstraction (e.g. from senders, receivers): find all attacks, may have false positives.

Analyze with automated prover.

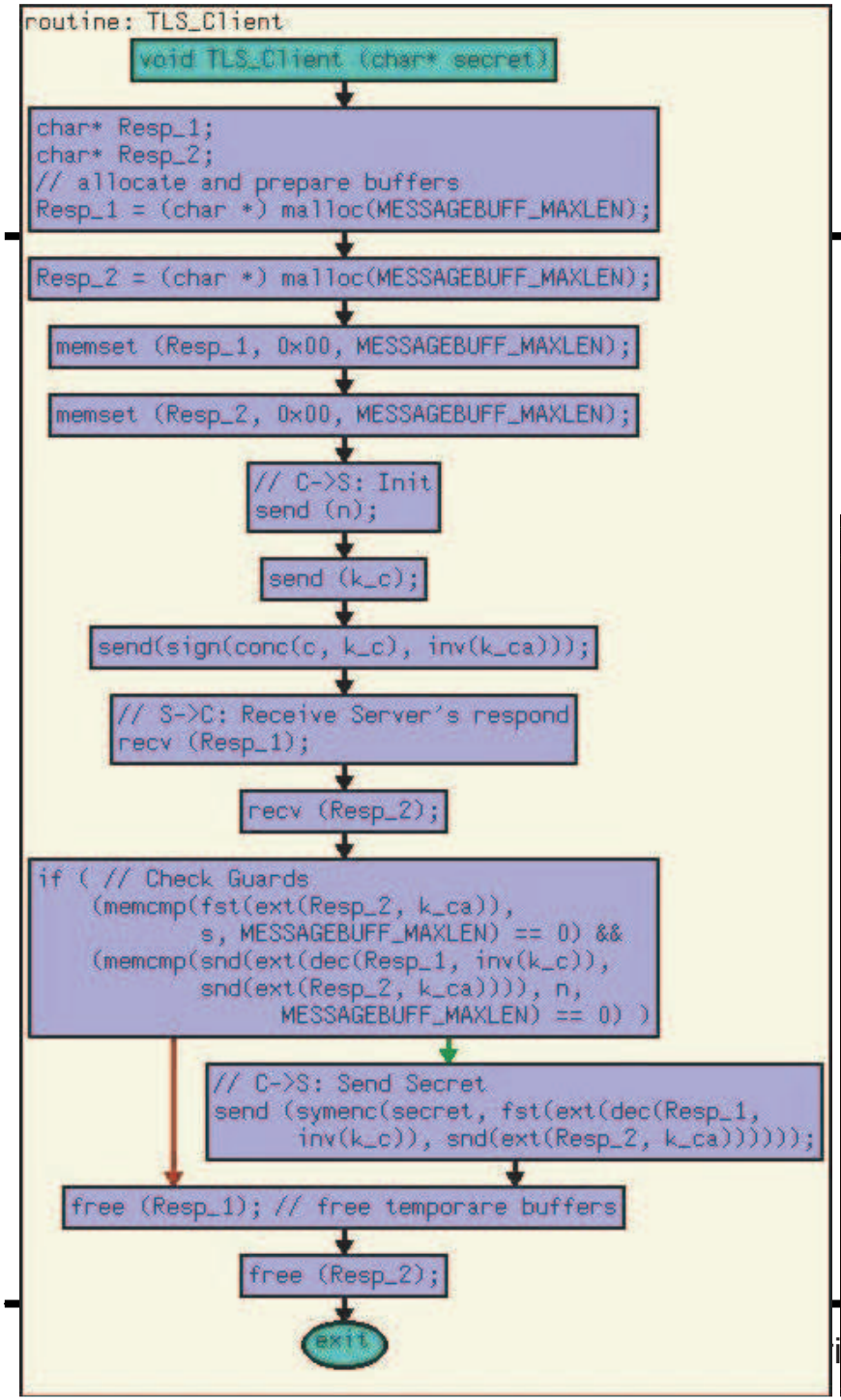
Example: Proposed Variant of TLS (SSL)

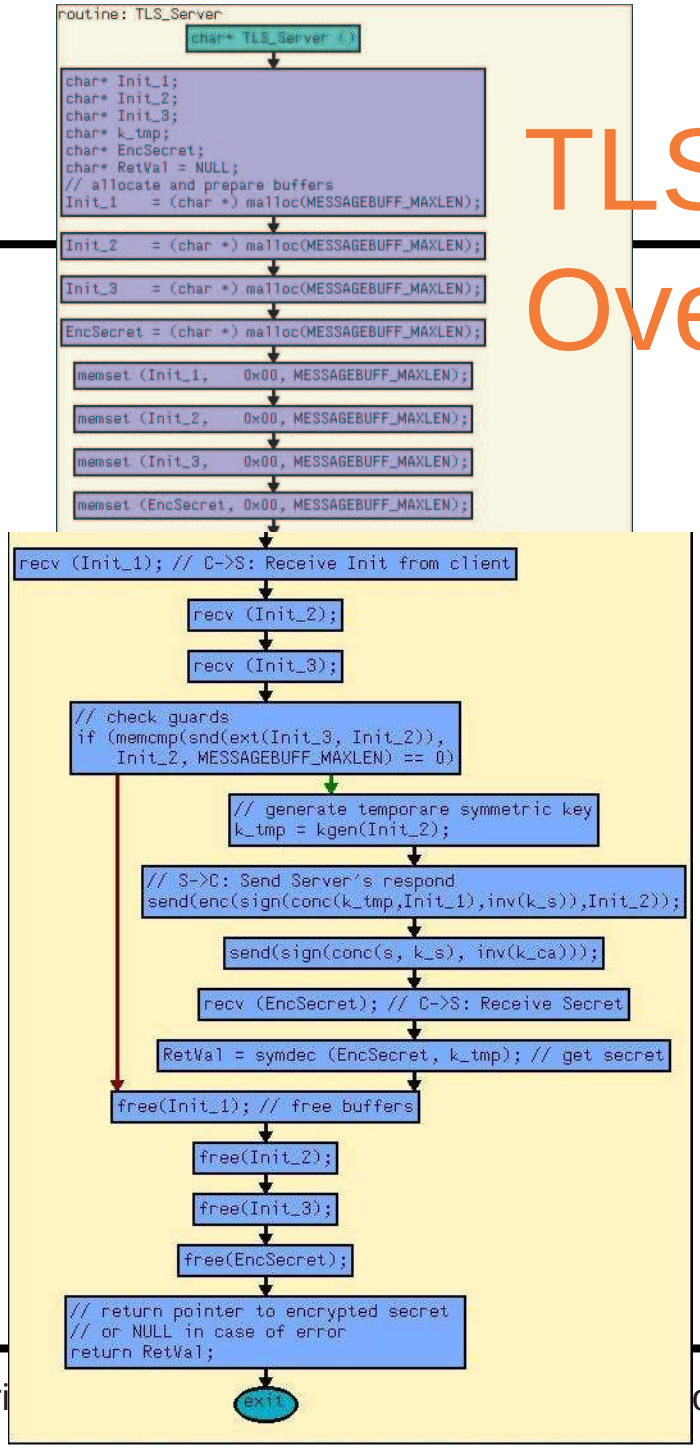
Presented at IEEE Infocom 1999.

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

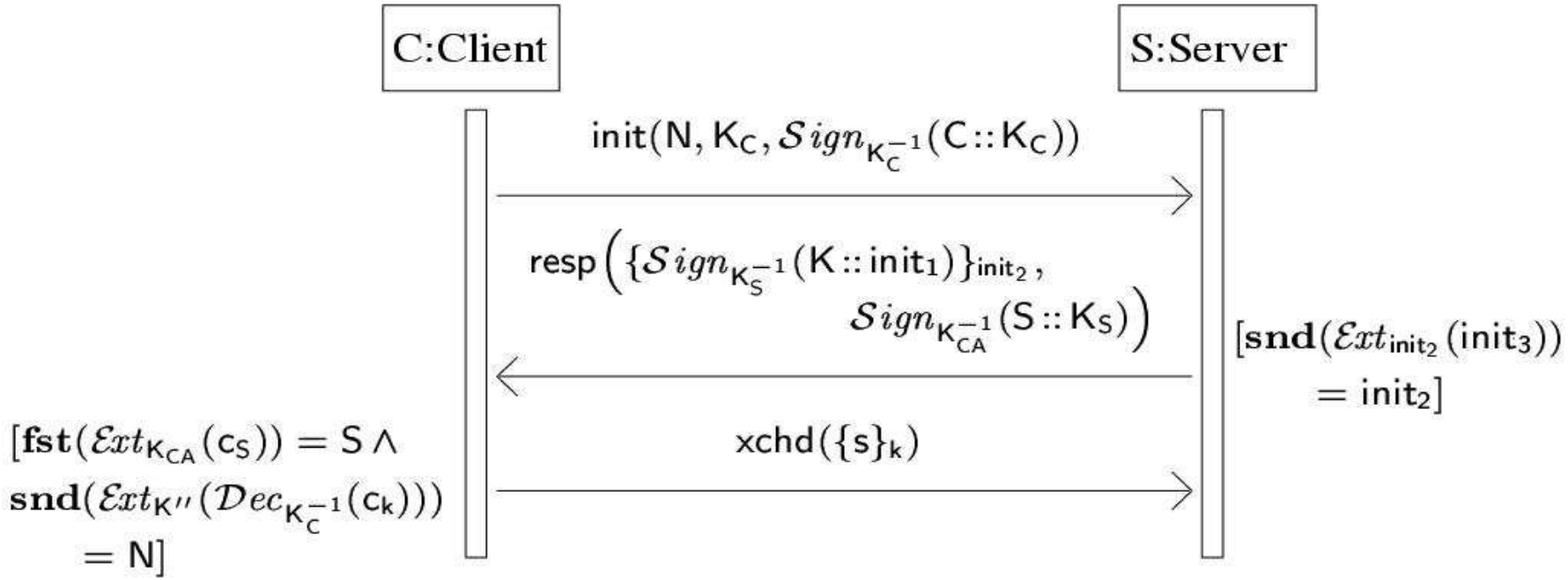


TLS Overview





Example: Translation to Logic



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

Surprise …

Can derive *knows(s)*.

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

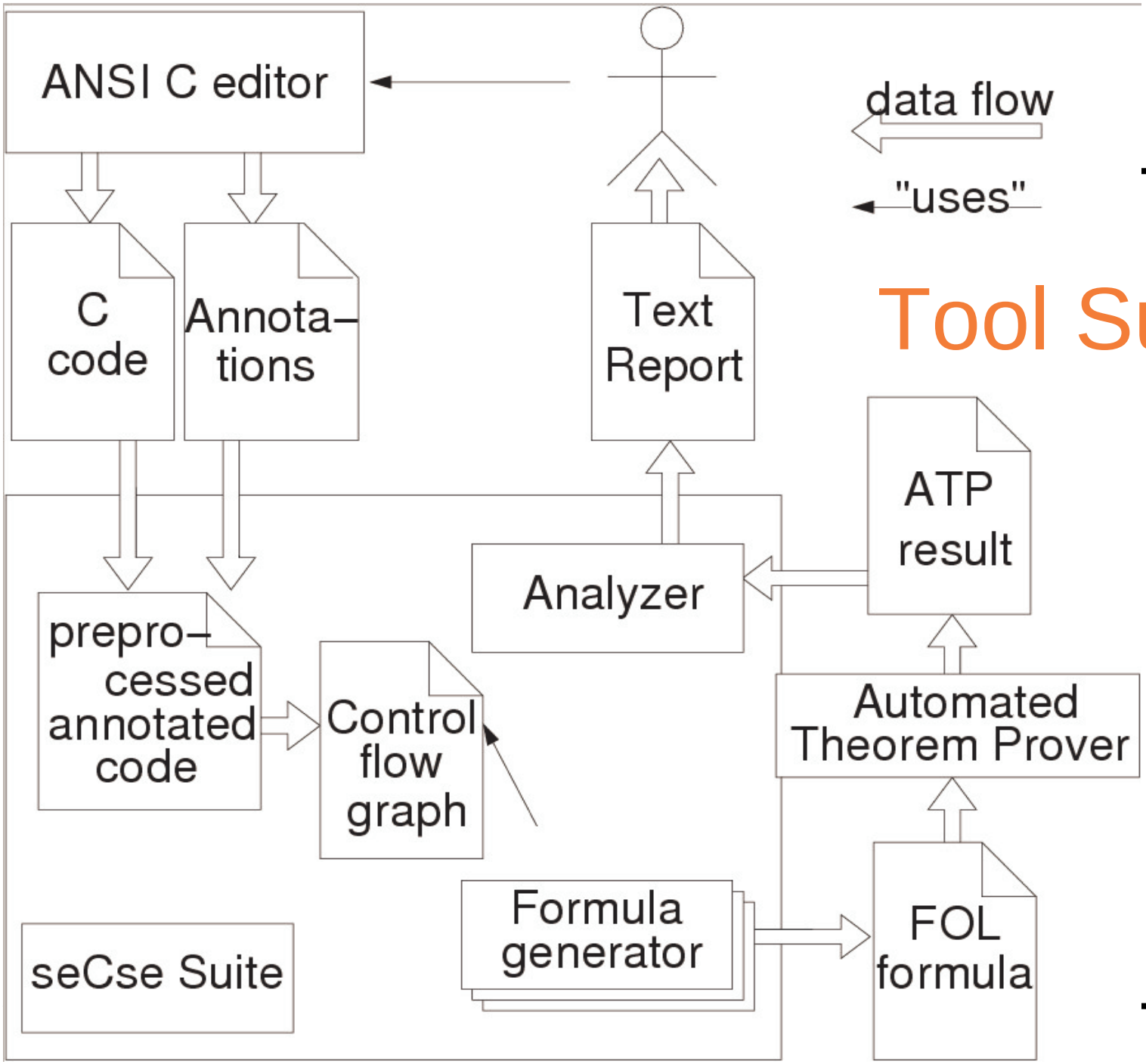
```
E-SETHEO csp03 single processor running on host ..
(c) 2003 Max-Planck-Institut fuer Informatik and
    Technische Universitaet Muenchen

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

Attack

➔ Completely insecure wrt stated goals.

But why ? Use prolog-based attack generator.

Tool Support



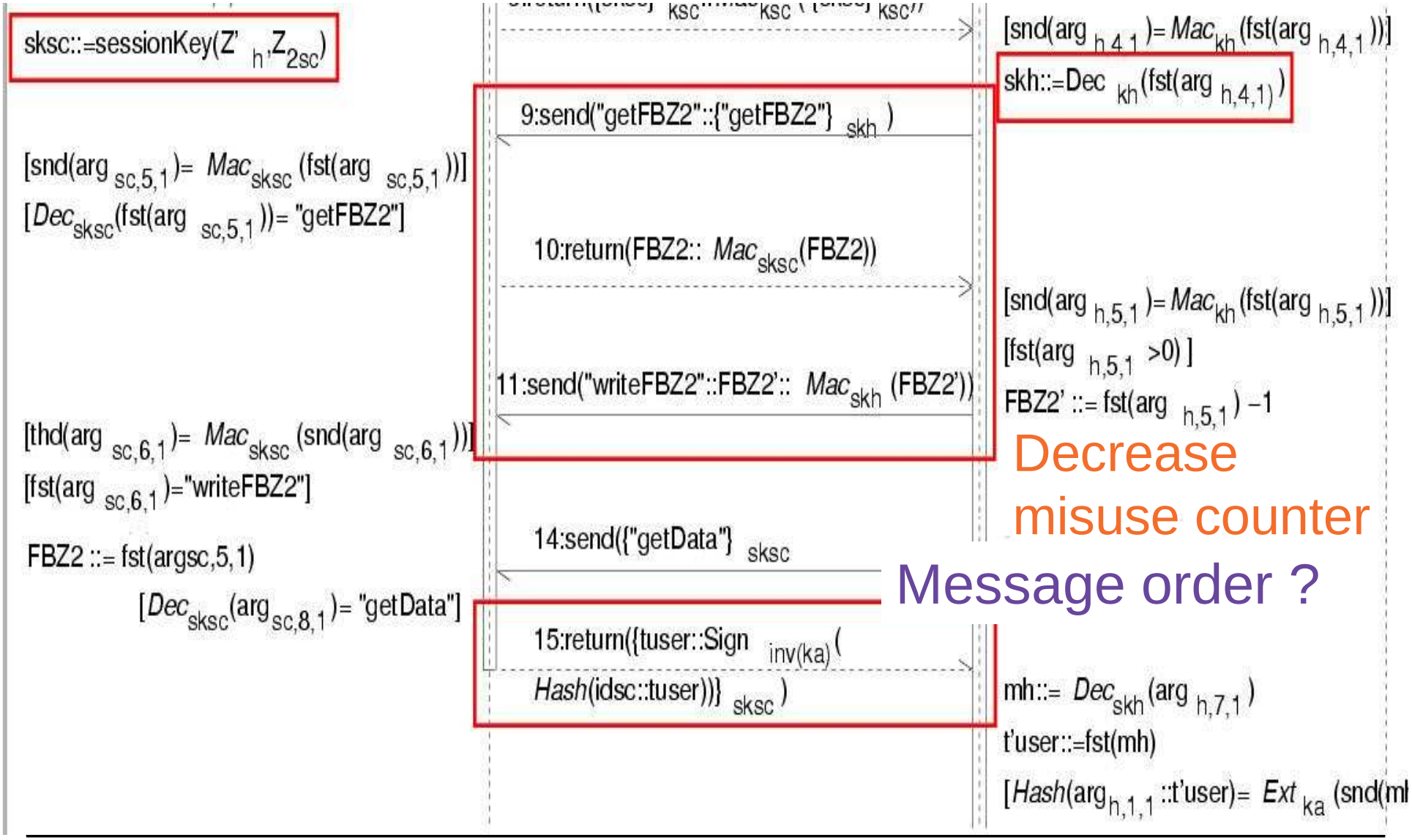
Biometric Authentication System

In development by company in joint project.

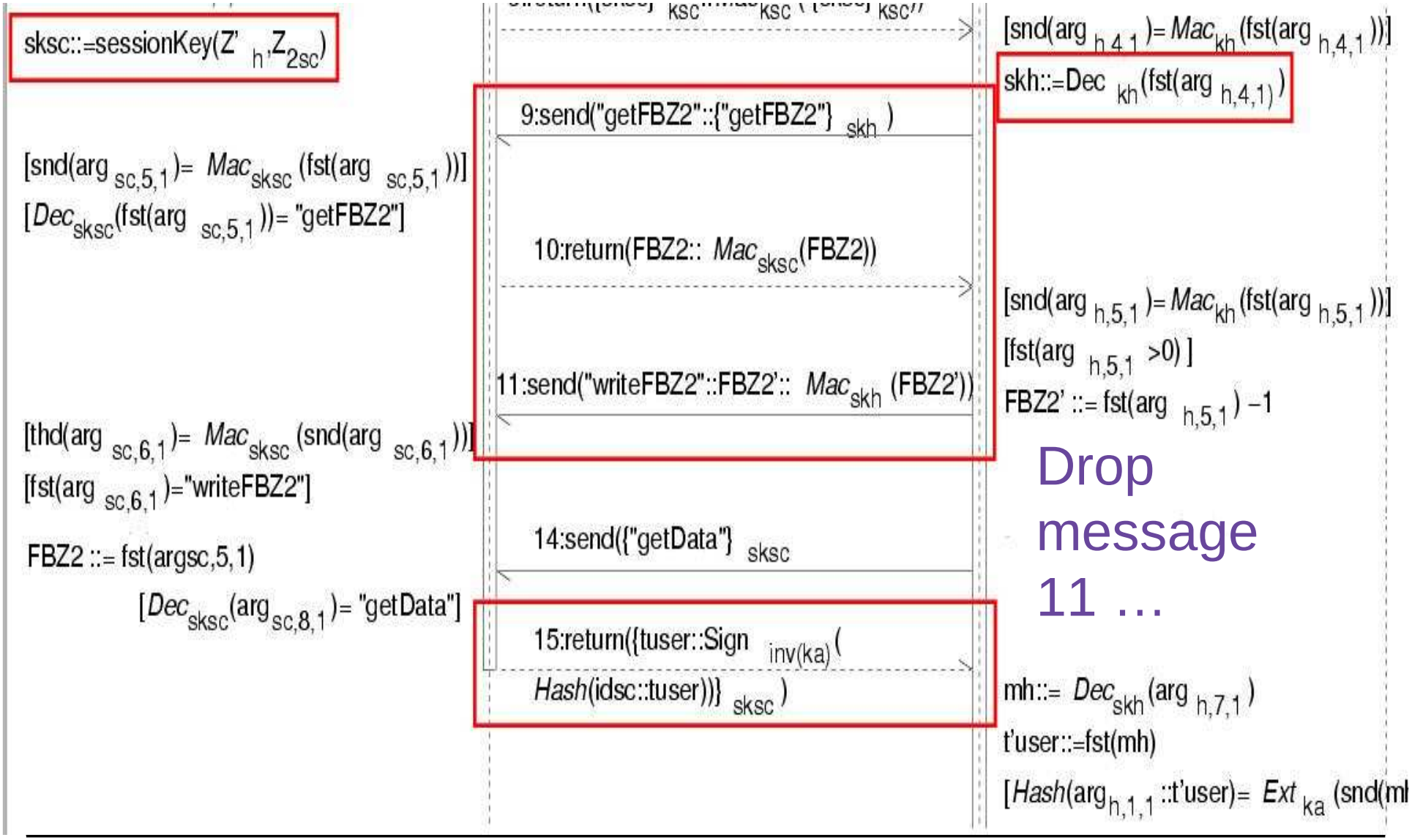
Store bio-reference template on smart-card.

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

Authent. Protocol Pt. 2: Problem ?

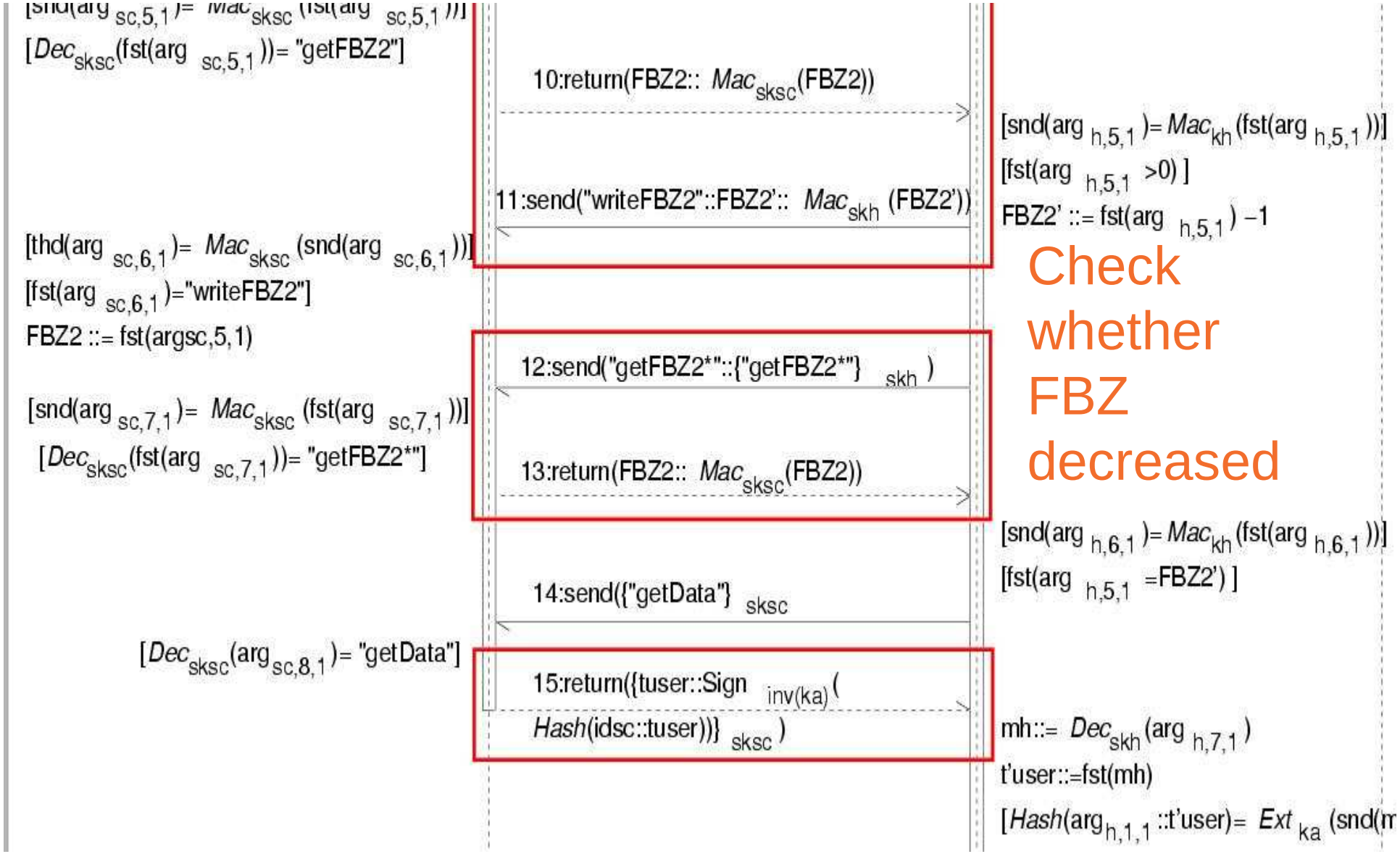


Authent. Protocol Pt. 2: Problem.

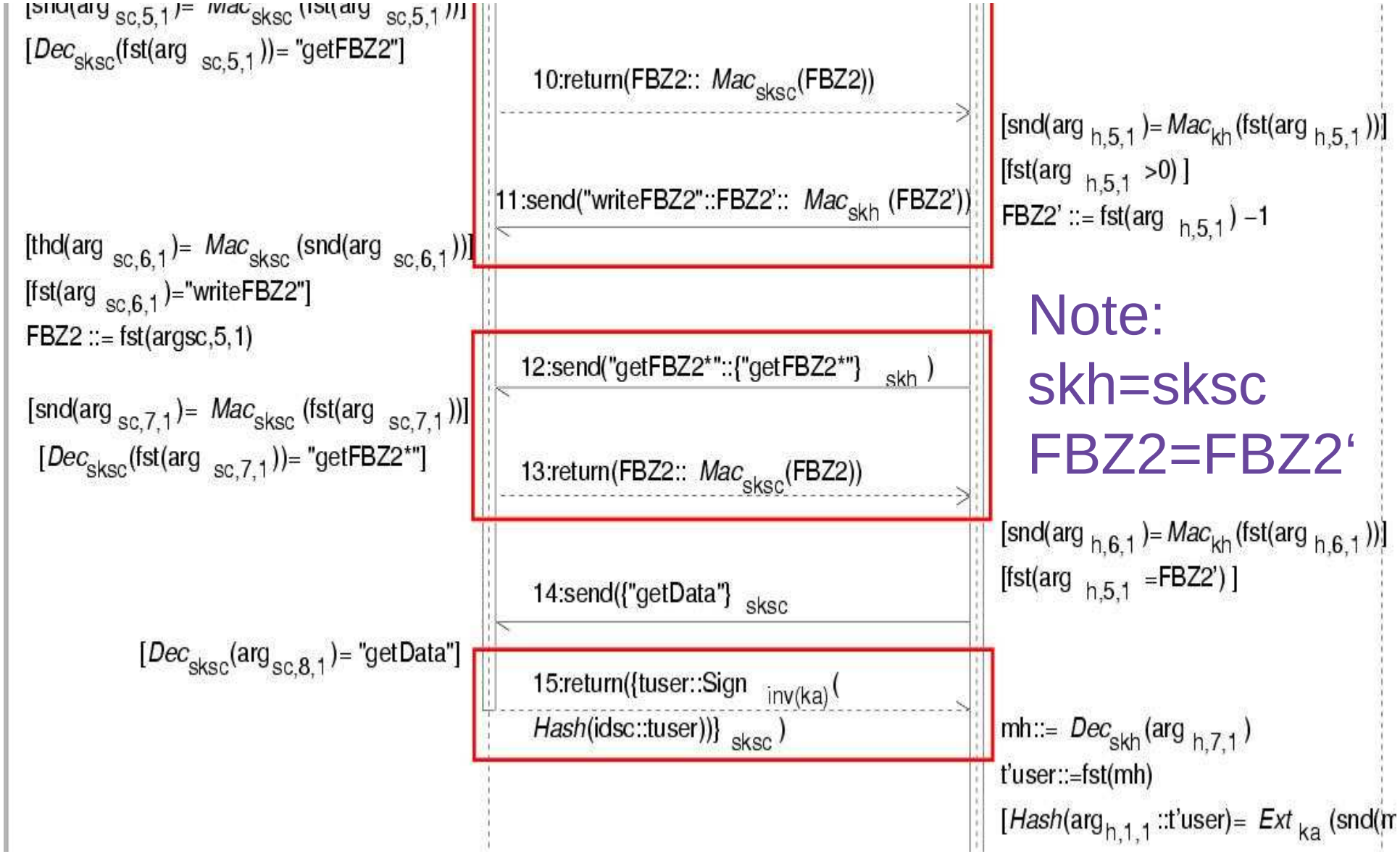


Drop message 11 …

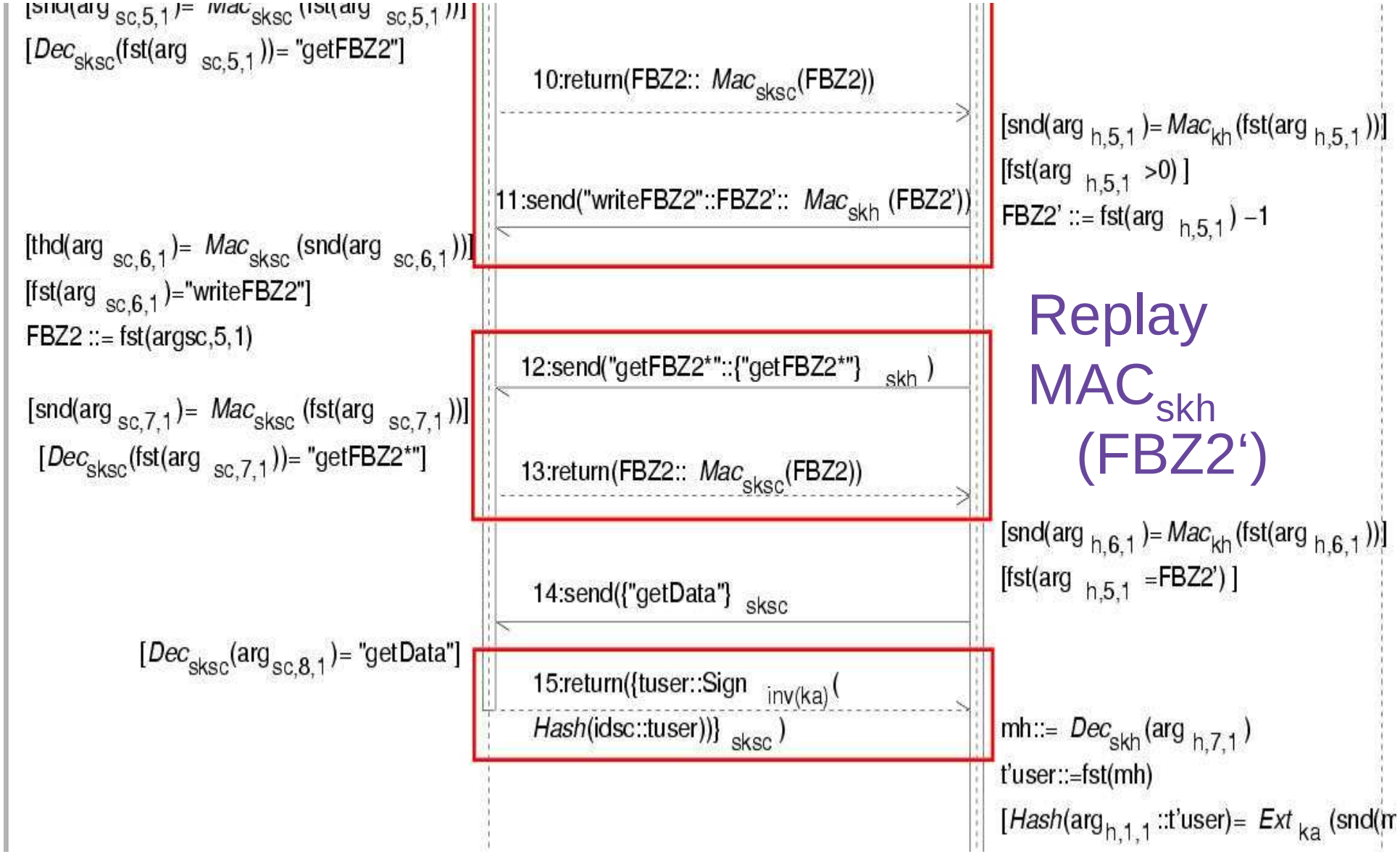
Authent. Protocol Pt. 2: Improvement



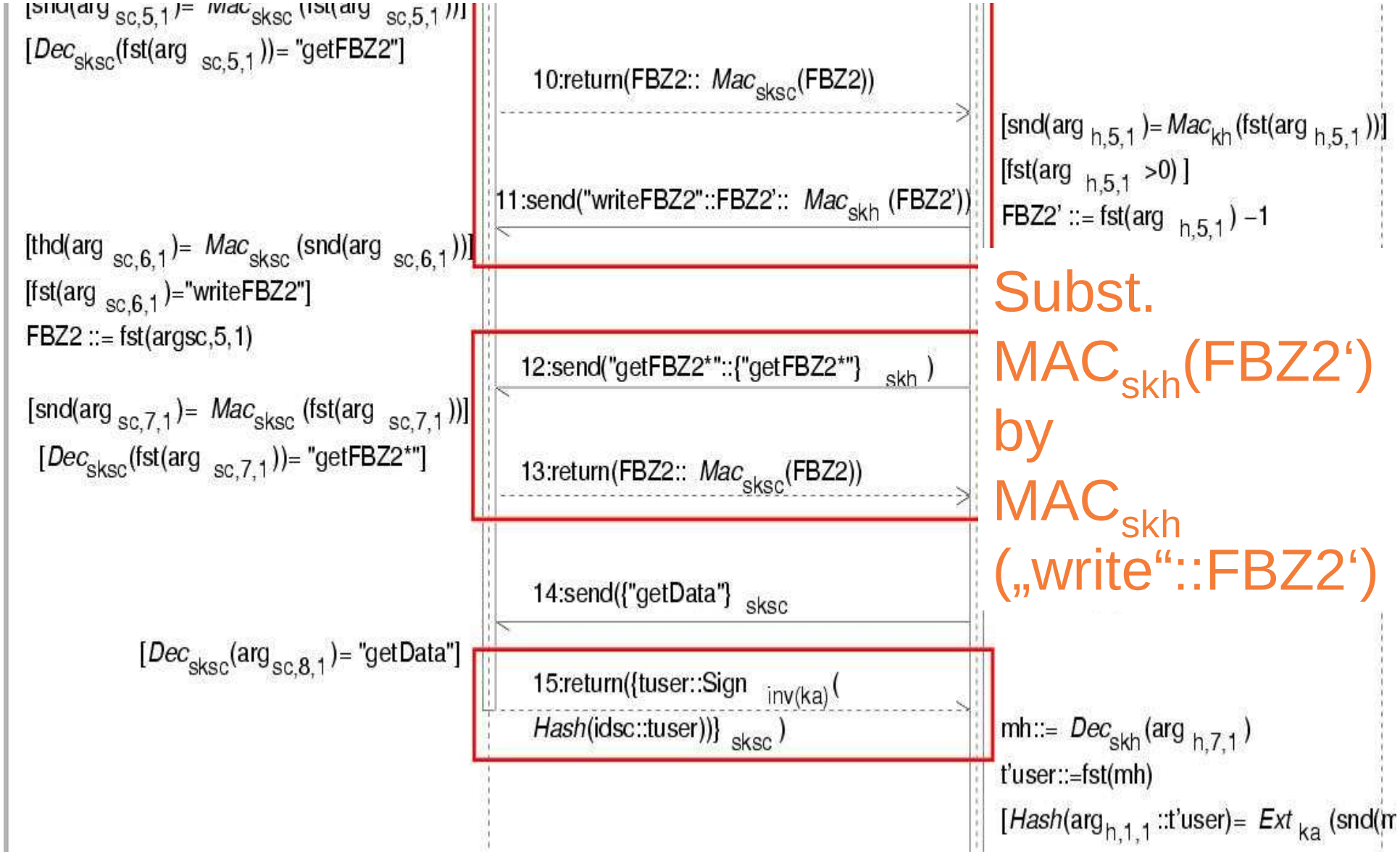
Authent. Prot. Pt. 2: Improvement ?



Authent. Prot. Pt. 2: Problem

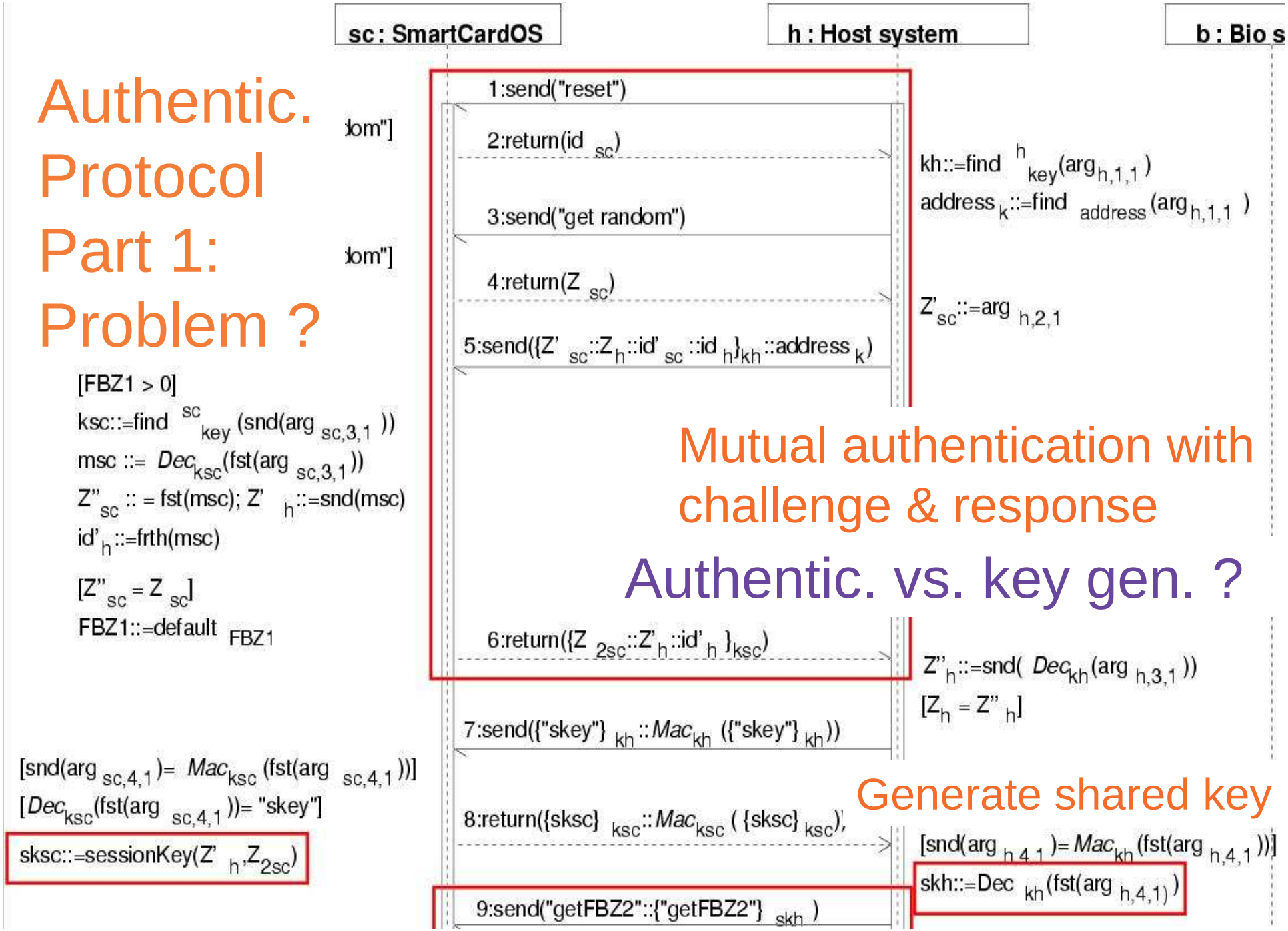


Authent. Prot. Pt. 2: Improvement (?)



Subst. MAC_{skh}(FBZ2‘) by MAC_{skh} („write“::FBZ2‘)

Authentic. Protocol Part 1: Problem ?

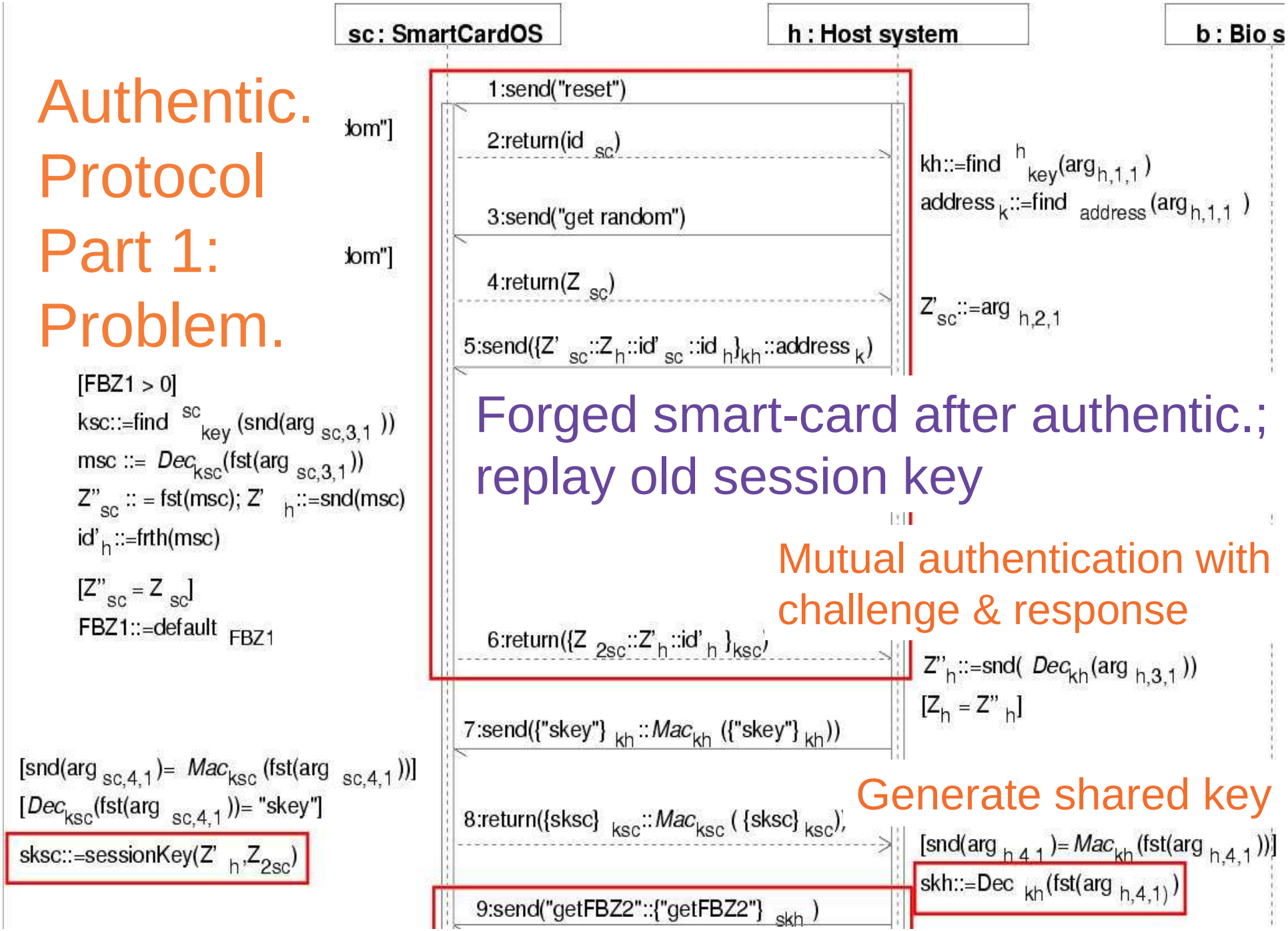


Mutual authentication with challenge & response

Authentic. vs. key gen. ?

Generate shared key

Authentic. Protocol Part 1: Problem.



Authentic. Protocol Part 1: Improve-ment (!)

sc : SmartCardOS | h : Host system | b : Bio s

1:send("reset")

2:return(id $_{sc}$)

kh::=find $^{h}_{key}$(arg$_{h,1,1}$)

address$_k$::=find $_{address}$(arg$_{h,1,1}$)

3:send("get random")

4:return(Z $_{sc}$)

Z'$_{sc}$::=arg $_{h,2,1}$

5:send({Z' $_{sc}$::Z$_h$::id' $_{sc}$ ::id $_h$}$_{kh}$::address $_k$)

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 $_{FBZ1}$

Mutual authentication with challenge & response

6:return({Z $_{2sc}$::Z'$_h$::id'$_h$ }$_{ksc}$)

Use (both) random numbers in Macs

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"]

Generate shared key

8:return({sksc} $_{ksc}$::*Mac*$_{ksc}$ ({sksc} $_{ksc}$)

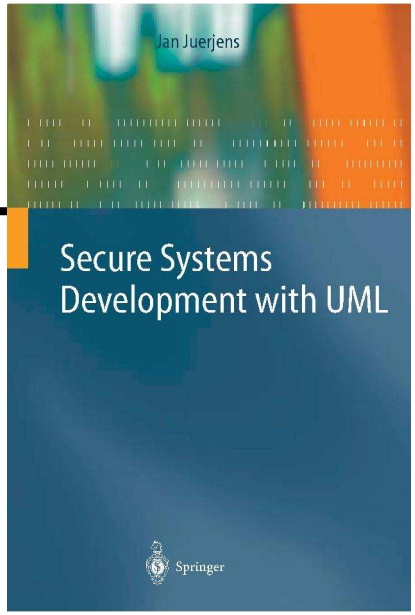
[snd(arg $_{h,4,1}$)= *Mac*$_{kh}$ (fst(arg $_{h,4,1}$))]

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

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

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

Conclusions



Understanding Security Goals using First-Order-Logic:

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

Further work: assertions.

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