

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

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

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

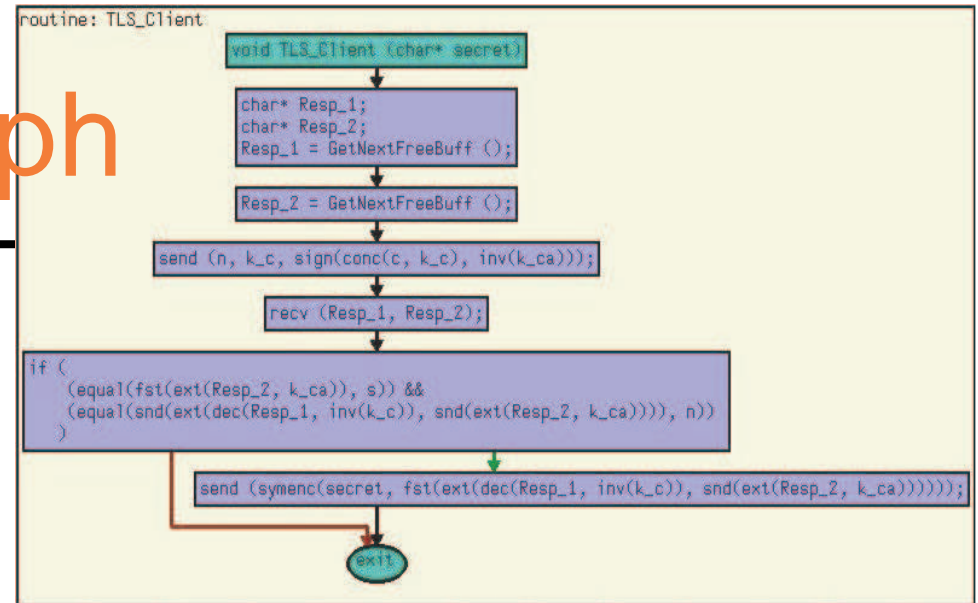
Control Flow Graph

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 $PRED(TR1) =$

$$\begin{aligned} & \forall msg_in. [knows(msg_in) \wedge cond(msg_in) \\ & \quad \Rightarrow knows(msg_out) \\ & \quad \wedge PRED(TR2)] \end{aligned}$$

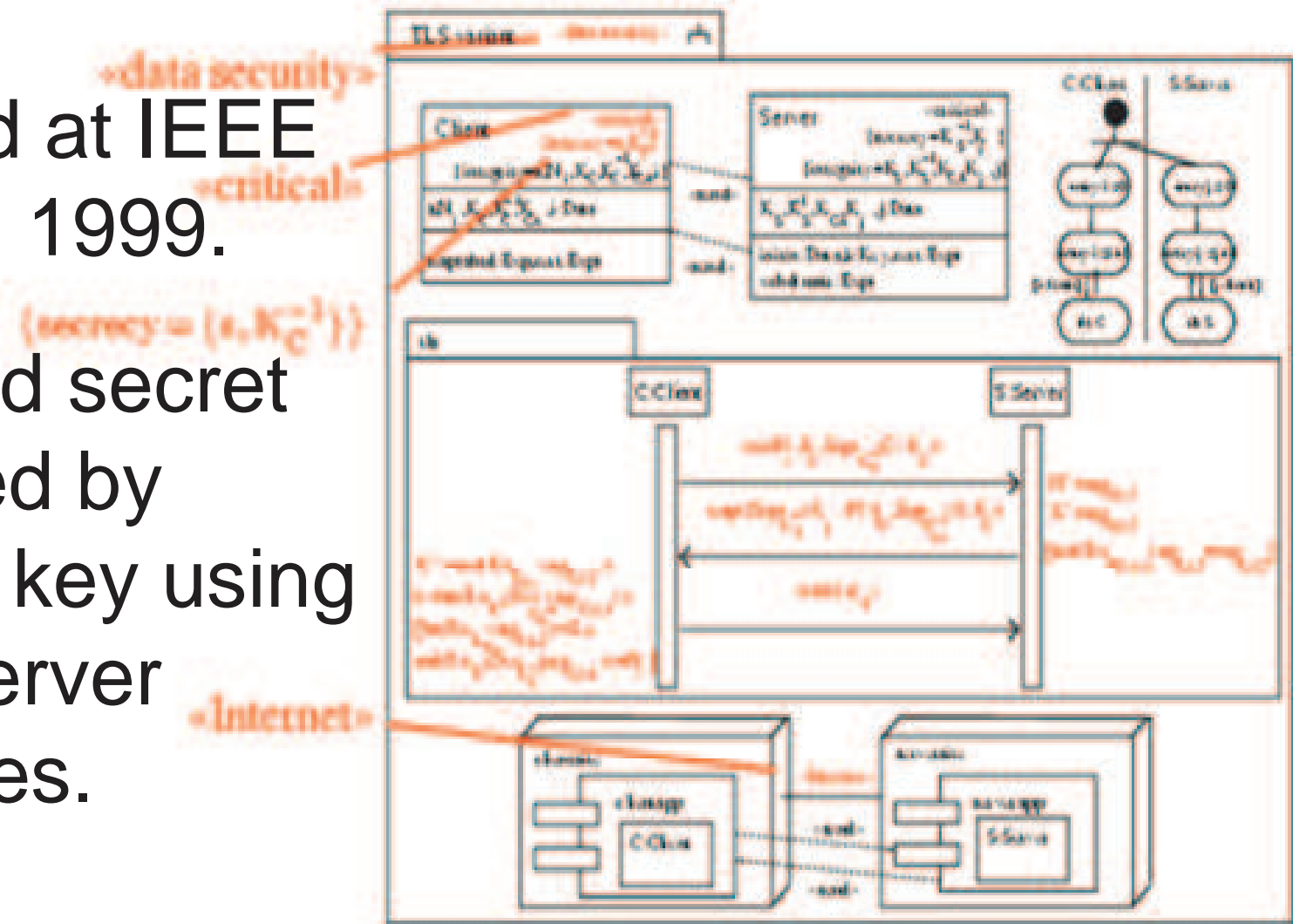
Abstraction (e.g. from senders, receivers): find all attacks, may have false positives.

Analyze with automated prover.

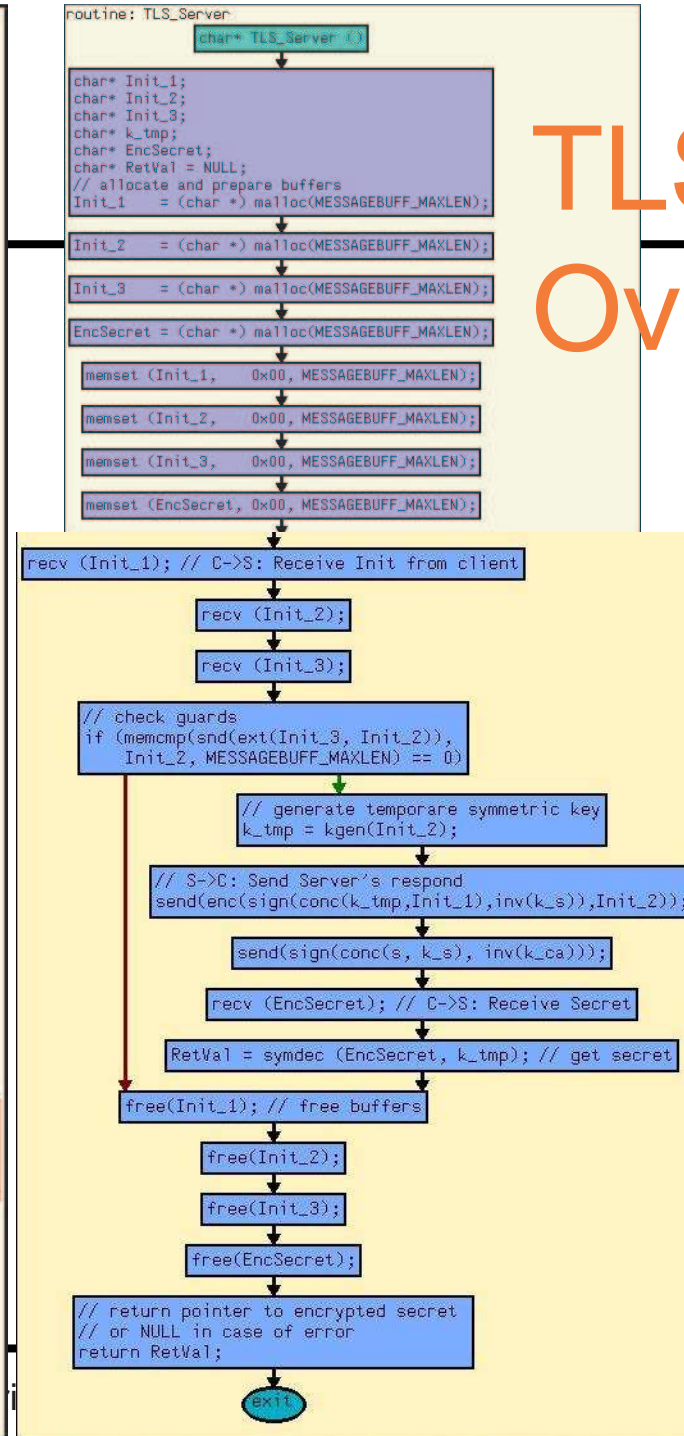
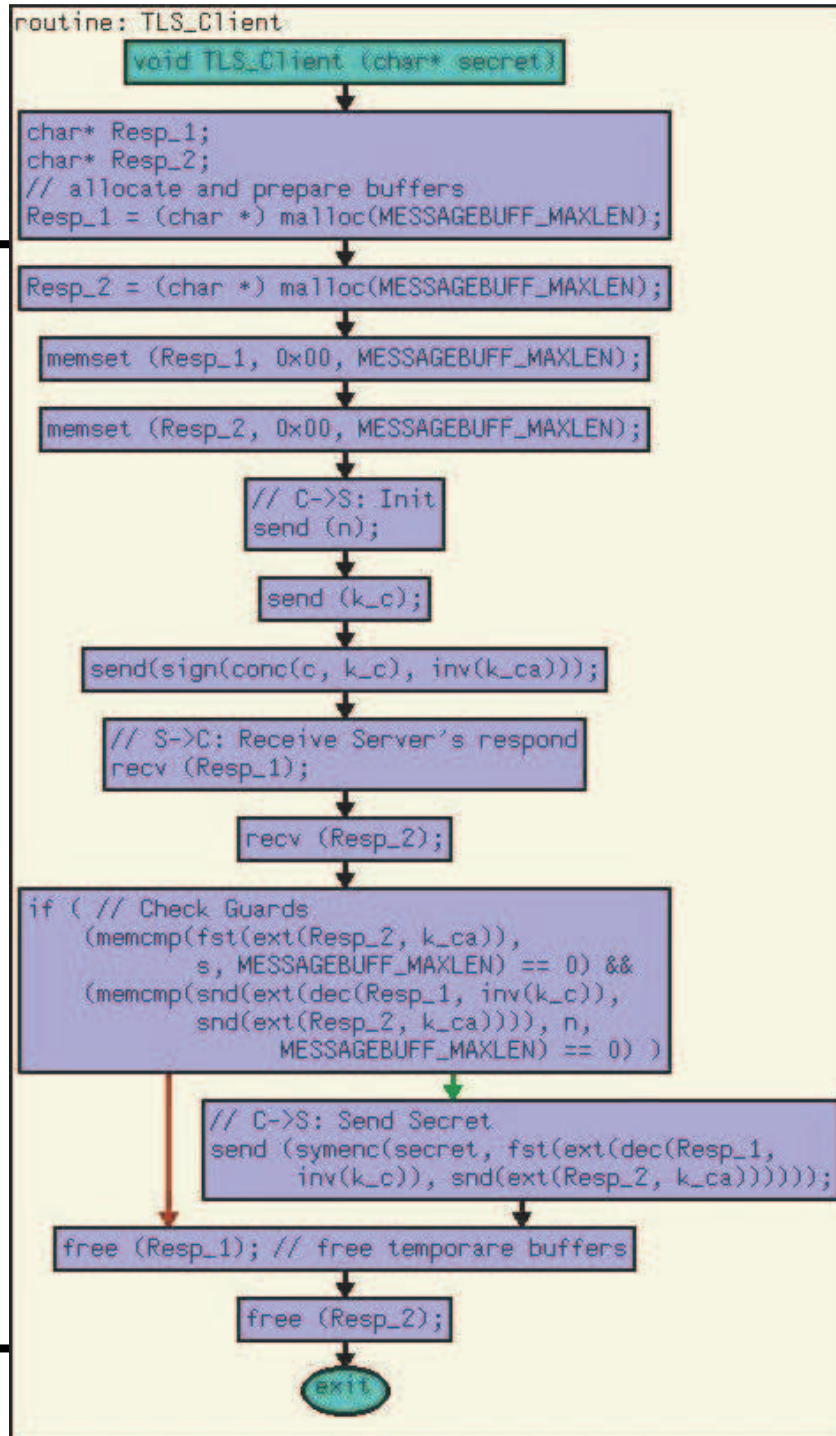
Example: Proposed Variant of TLS (SSL)

Presented at IEEE
Infocom 1999.

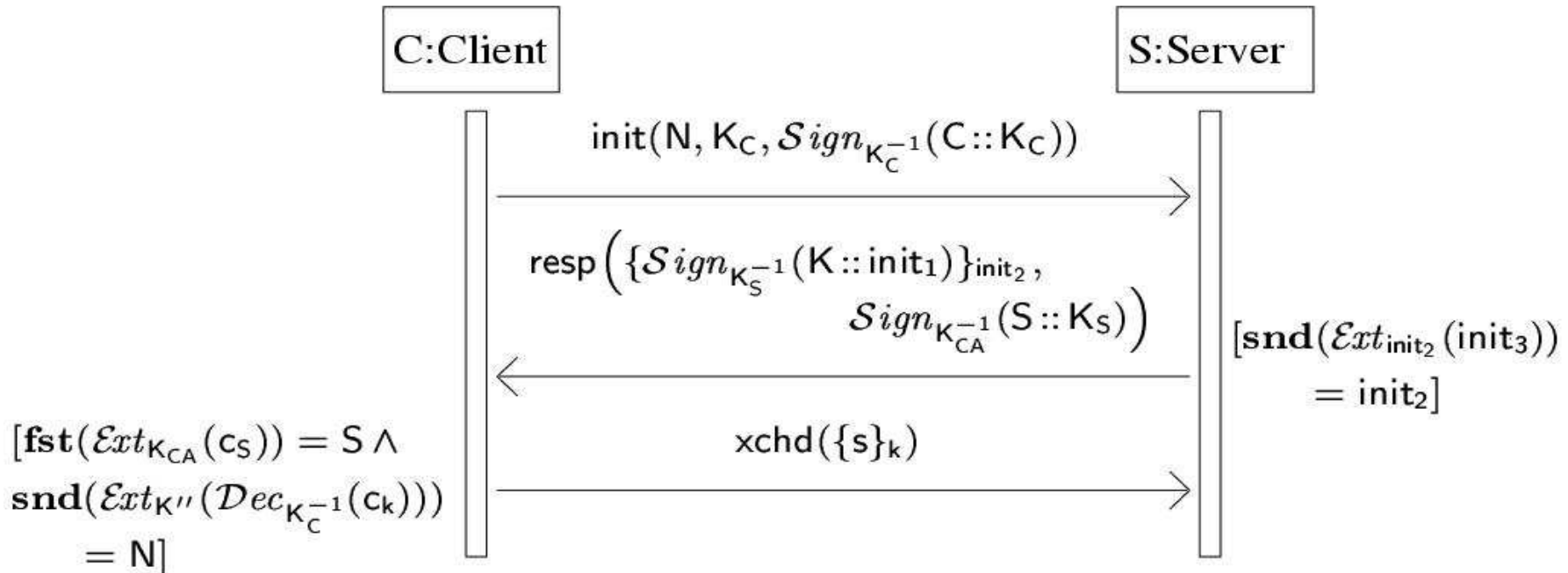
Goal: send secret
protected by
session key using
fewer server
resources.



TLS Overview



Example: Translation to Logic



$knows(N) \wedge knows(K_C) \wedge knows(Sign_{K_C^{-1}}(C::K_C))$
 $\wedge \forall init_1, init_2, init_3. [knows(init_1) \wedge knows(init_2) \wedge$
 $knows(init_3) \wedge snd(Ext_{init_2}(init_3)) = init_2$
 $\Rightarrow knows(\{Sign_{K_S^{-1}}(\dots)\}_{init_2}) \wedge [\dots] \wedge [\dots \Rightarrow \dots] \dots]$

Surprise ...

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

Can derive *knows(s)*.

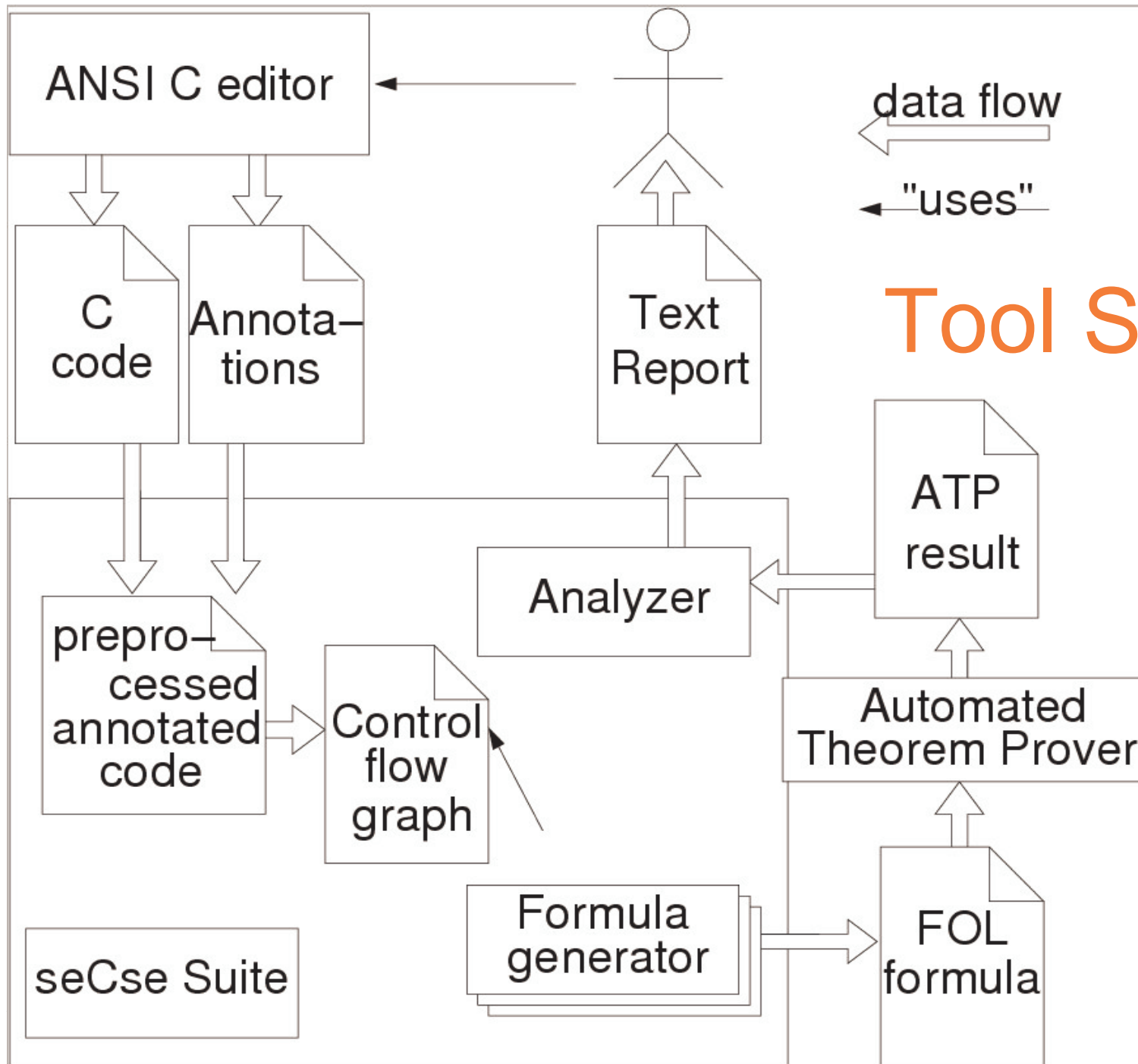
That is: Protocol
does **not** preserve
secrecy of *s* against
adversaries.

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

Attack

→ Completely insecure wrt stated goals.

But why ? Use prolog-based attack
generator.



Tool Support

Biometric Authentication System

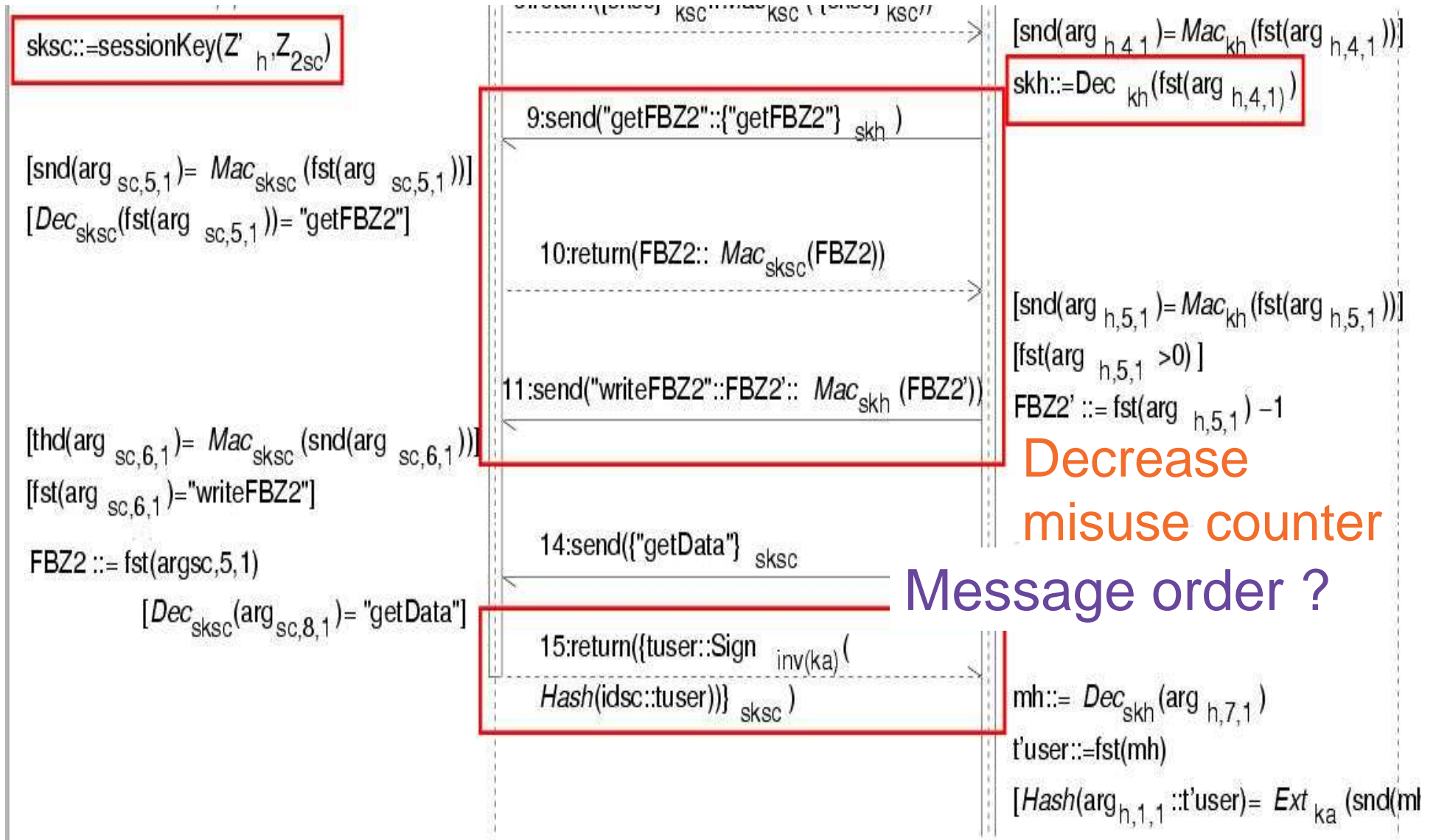
In development by company in joint project.

Store bio-reference template on smart-card.

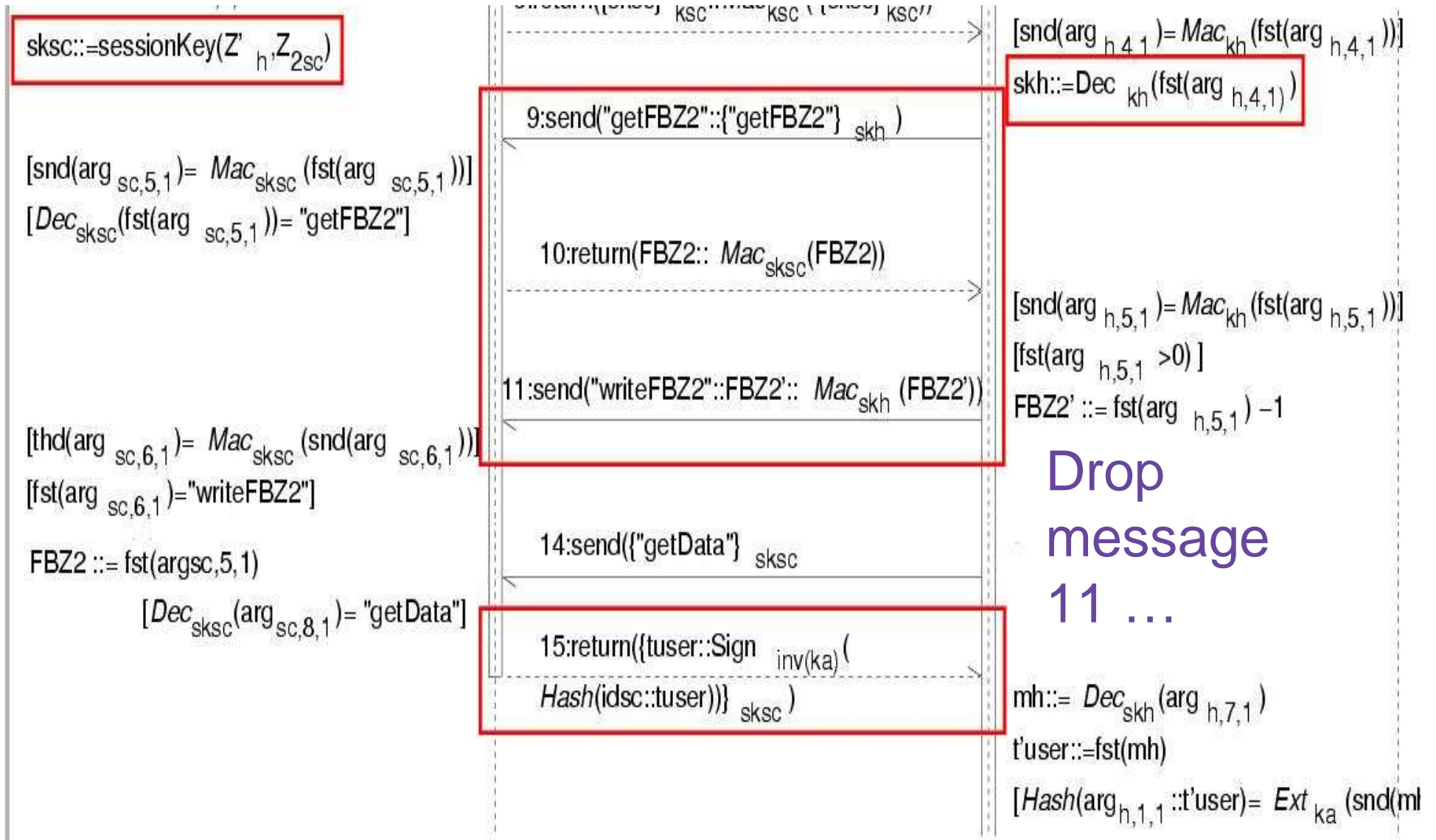
Discovered **three major attacks** against subsequently improved versions (**misuse counter circumvented** by dropping / replaying messages, **smart-card insufficiently authenticated** by recombining sessions).



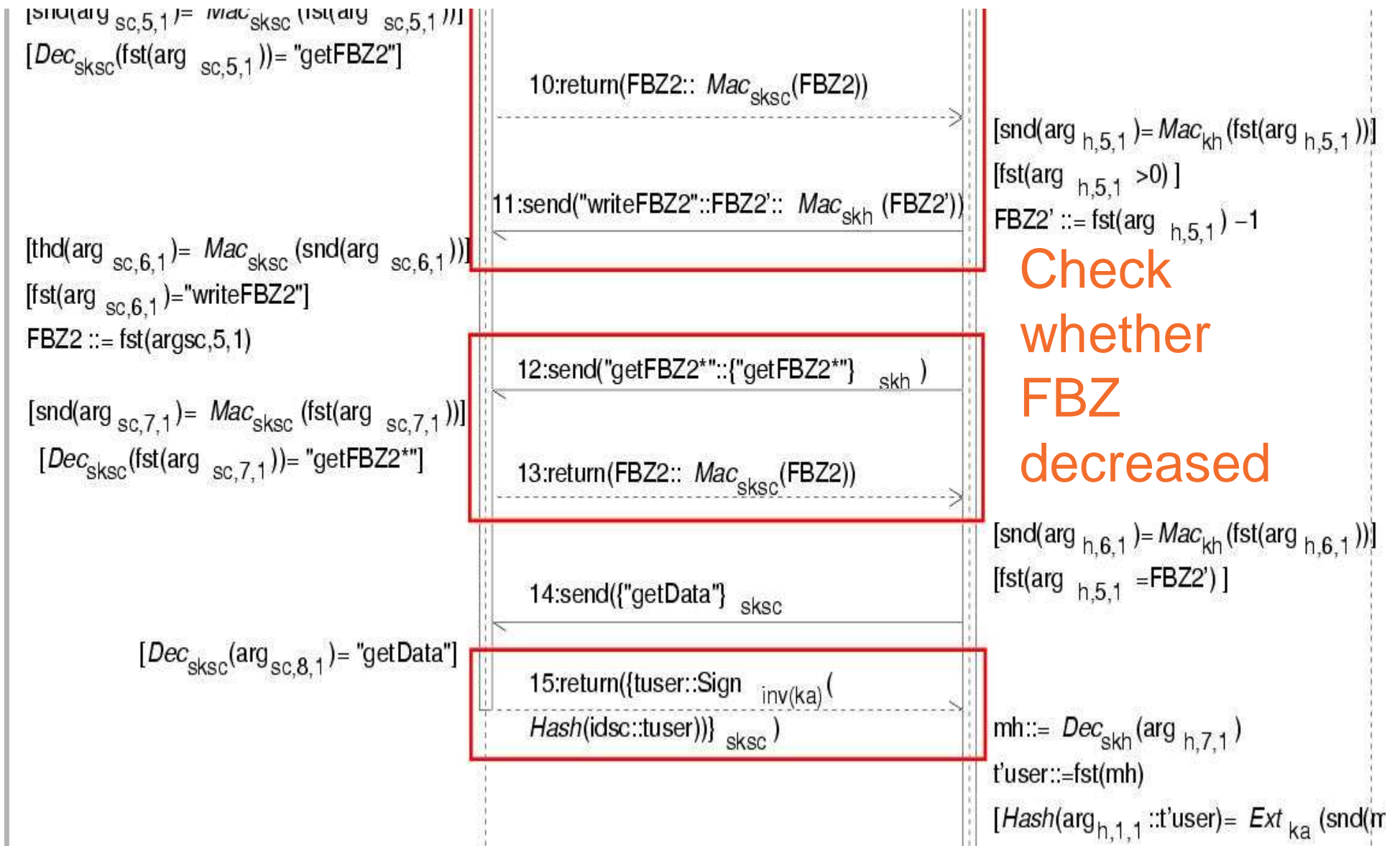
Authent. Protocol Pt. 2: Problem ?



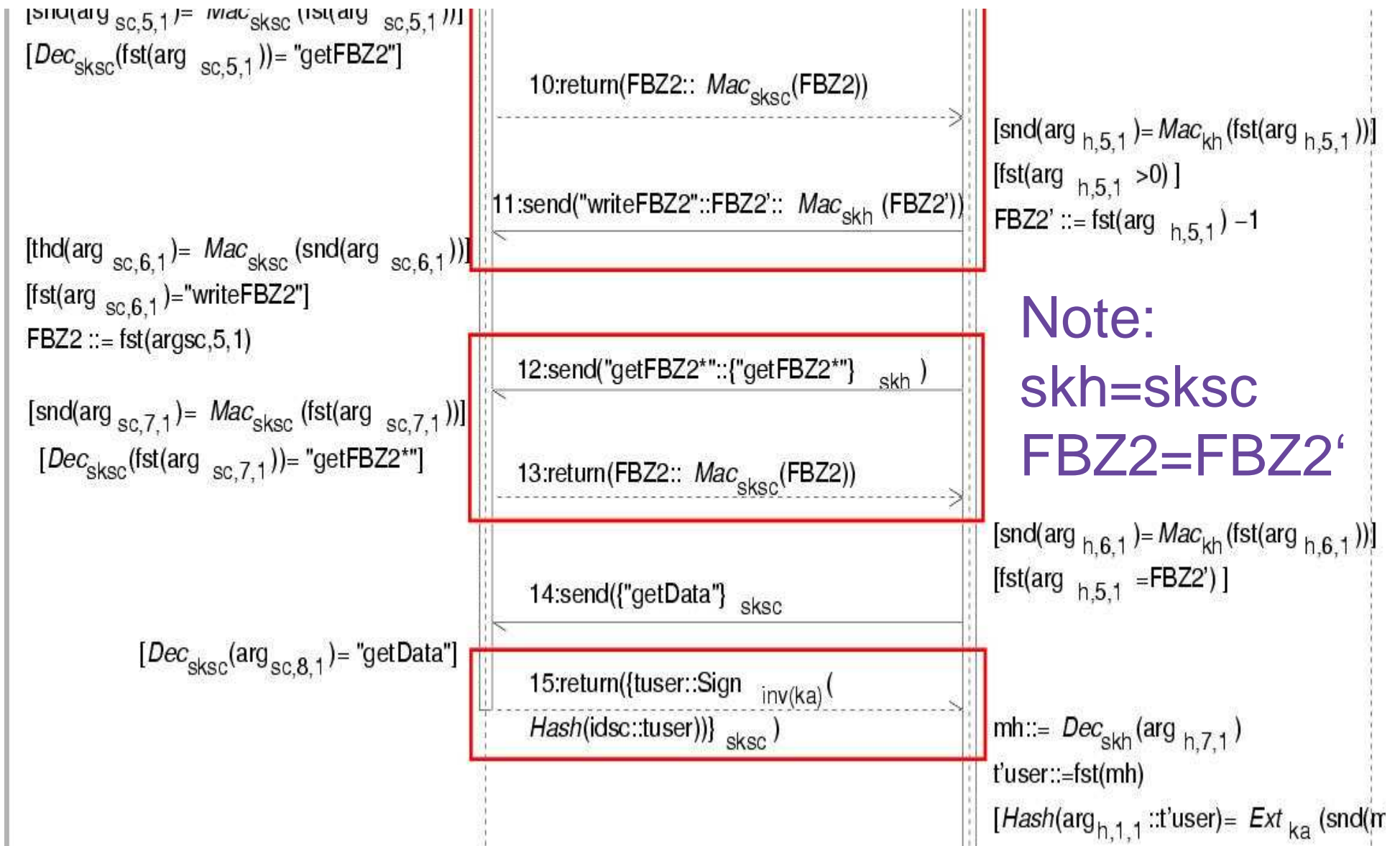
Authent. Protocol Pt. 2: Problem.



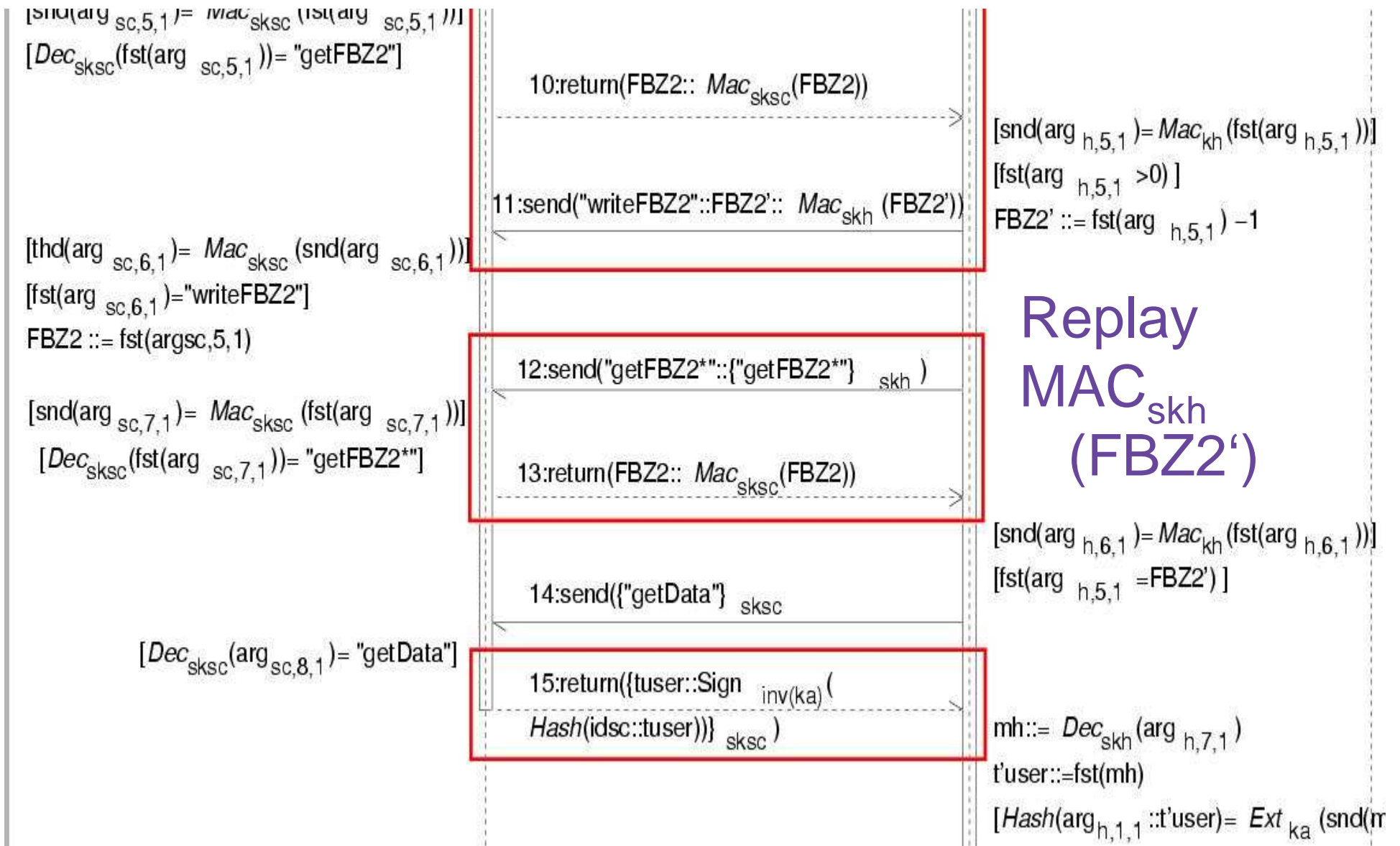
Authent. Protocol Pt. 2: Improvement



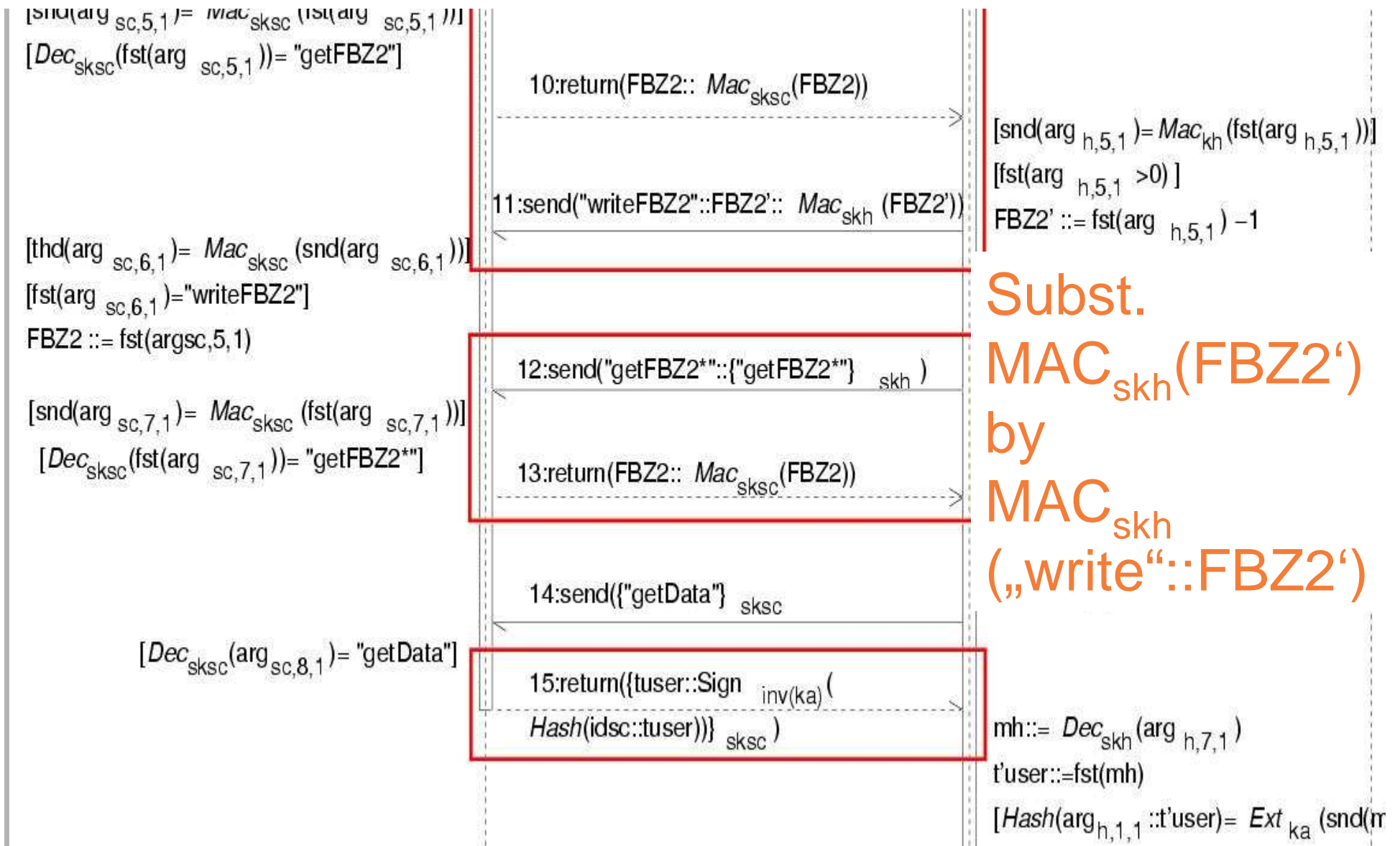
Authent. Prot. Pt. 2: Improvement ?



Authent. Prot. Pt. 2: Problem



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

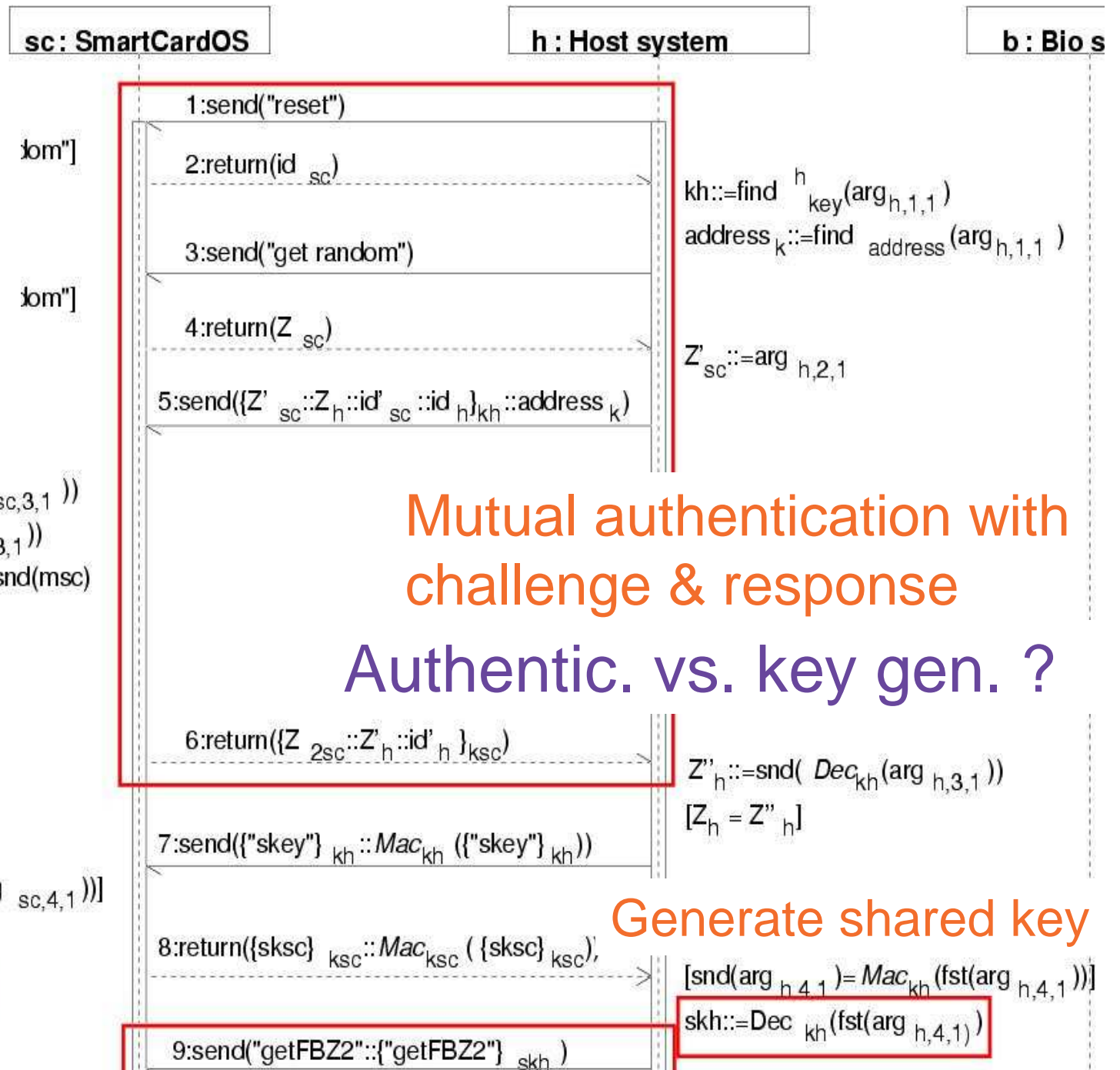


Authentic. Protocol Part 1: Problem ?

[FBZ1 > 0]
 $k_{sc} ::= \text{find}_{key}^{sc}(\text{snd}(\text{arg}_{sc,3,1}))$
 $msc ::= \text{Dec}_{k_{sc}}(\text{fst}(\text{arg}_{sc,3,1}))$
 $Z''_{sc} ::= \text{fst}(msc); Z'_h ::= \text{snd}(msc)$
 $id'_h ::= \text{frth}(msc)$
 $[Z''_{sc} = Z_{sc}]$
 $\text{FBZ1} ::= \text{default}_{FBZ1}$

$[\text{snd}(\text{arg}_{sc,4,1}) = \text{Mac}_{k_{sc}}(\text{fst}(\text{arg}_{sc,4,1}))]$
 $[\text{Dec}_{k_{sc}}(\text{fst}(\text{arg}_{sc,4,1})) = \text{"skey"}]$

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

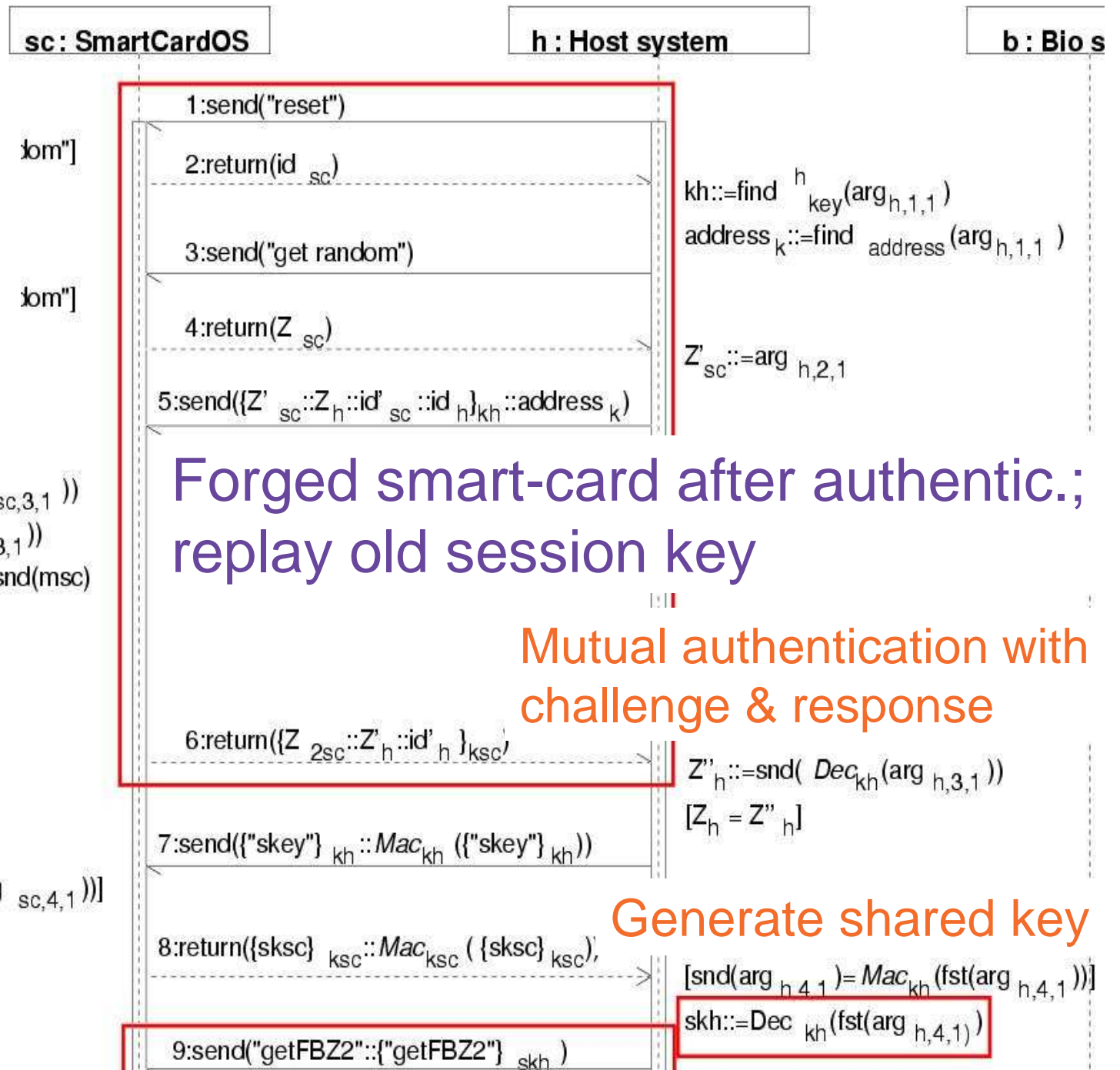


Authentic. Protocol Part 1: Problem.

[FBZ1 > 0]
 $k_{sc} ::= \text{find}_{sc}^{key}(\text{snd}(\text{arg}_{sc,3,1}))$
 $\text{msc} ::= \text{Dec}_{k_{sc}}(\text{fst}(\text{arg}_{sc,3,1}))$
 $Z''_{sc} ::= \text{fst}(\text{msc}); Z'_h ::= \text{snd}(\text{msc})$
 $\text{id}'_h ::= \text{frth}(\text{msc})$
 $[Z''_{sc} = Z_{sc}]$
 $\text{FBZ1} ::= \text{default}_{FBZ1}$

$[\text{snd}(\text{arg}_{sc,4,1}) = \text{Mac}_{k_{sc}}(\text{fst}(\text{arg}_{sc,4,1}))]$
 $[\text{Dec}_{k_{sc}}(\text{fst}(\text{arg}_{sc,4,1})) = \text{"skey"}]$

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

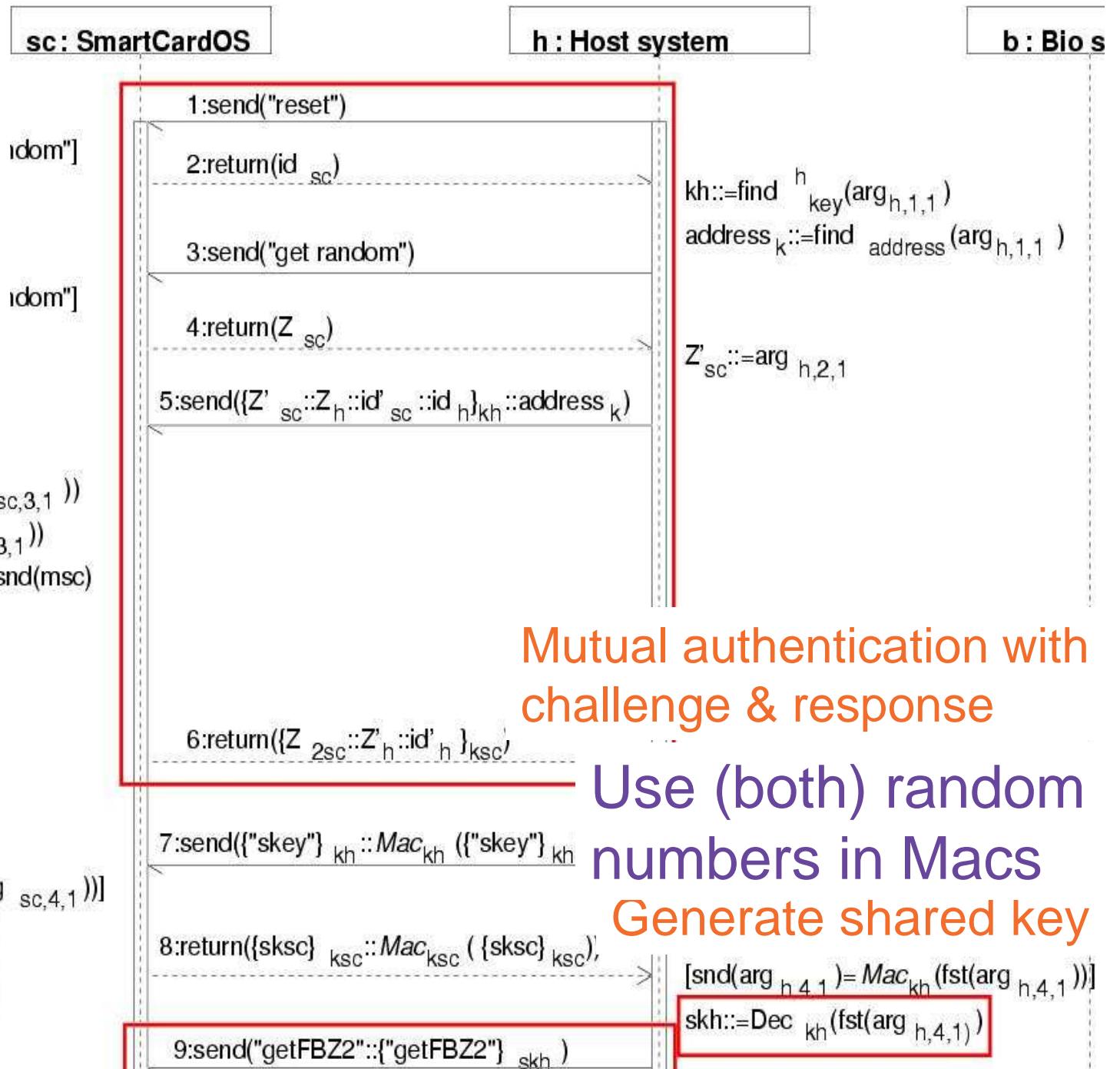


Authentic. Protocol Part 1: Improve- ment (!)

$k_{sc} ::= \text{find}_{key}^{sc}(\text{snd}(\arg_{sc,3,1}))$
 $m_{sc} ::= \text{Dec}_{k_{sc}}(\text{fst}(\arg_{sc,3,1}))$
 $Z'_{sc} ::= \text{fst}(m_{sc}); Z'_h ::= \text{snd}(m_{sc})$
 $id'_h ::= \text{frth}(m_{sc})$
 $[Z'_{sc} = Z_{sc}]$
 $\text{FBZ1} ::= \text{default}_{FBZ1}$

$[\text{snd}(\arg_{sc,4,1}) = \text{Mac}_{k_{sc}}(\text{fst}(\arg_{sc,4,1}))]$
 $[\text{Dec}_{k_{sc}}(\text{fst}(\arg_{sc,4,1})) = \text{"skey"}]$

$\text{sk}_{sc} ::= \text{sessionKey}(Z'_h, Z'_{2sc})$



Mutual authentication with challenge & response

Use (both) random numbers in Macs
Generate shared key

$[\text{snd}(\arg_{h,4,1}) = \text{Mac}_{kh}(\text{fst}(\arg_{h,4,1}))]$
 $\text{sk}_{kh} ::= \text{Dec}_{kh}(\text{fst}(\arg_{h,4,1}))$

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>

