LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

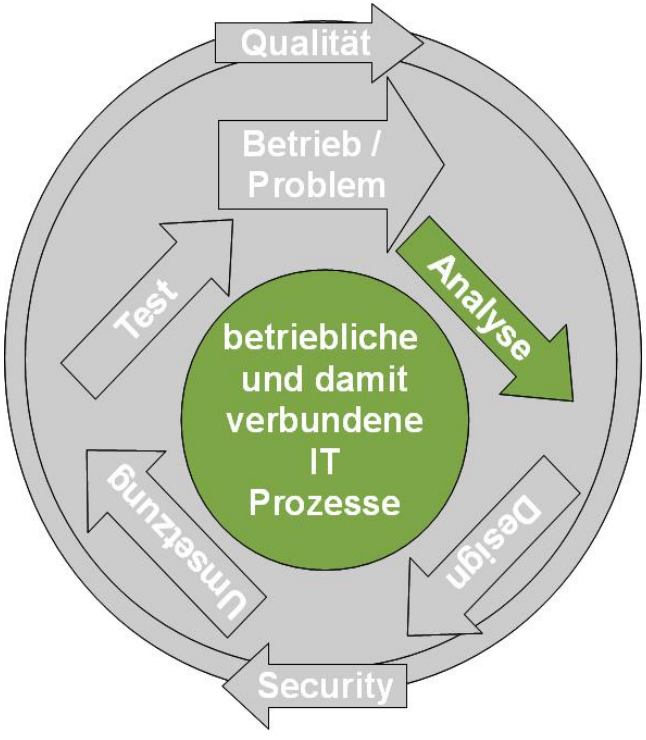
3.6: TLS-Variant

v. 26.06.2013

3.6 TLS-Variant

Einordnung
Sicheres Software Design

- Geschäfts-Prozesse
- Modelbasierte Softwareenwicklung
- Sicheres Software Design
  - Sicherheitsanforderungen
  - UMLsec
  - UML-Analysis
  - Design Principles
  - Examples
    - TLS Variant
    - CEPS Purchase



Security Protocols

System distributed over untrusted networks.

„Adversary“ intercepts, modifies, deletes, inserts messages.

Cryptography provides security.

Cryptographic Protocol: Exchange of messages for distributing session keys, authenticating principals etc. using cryptographic algorithms

Security Protocols: Problems

Many protocols have vulnerabilities or subtleties for various reasons

- weak cryptography
- core message exchange
- interfaces, prologues, epilogues
- deployment
- implementation bugs

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,
- may injects produced messages in some links
- may access certain nodes.

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, simulate jointly with adversary model.

Cryptography

Keys are symbols, crypto-algorithms are abstract operations.

- Can only decrypt with right keys.
- Can only compose with available messages.
- Cannot perform statistical attacks.

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

- $\forall$ E,K. $Dec_K^{-1}(\{E\}_K)=E$
- $\forall$ E,K. $Ext_K(Sign_K^{-1}(E))=E$
- $\forall$ E_1,E_2. $head(E_1::E_2)=E_1$
- $\forall$ E_1,E_2. $tail(E_1::E_2)=E_2$
- Associativity for ::.

Write E1::E2::E3 for E1::(E2::E3) and fst(E1::E2) for head(E1::E2) etc.

Can include further crypto-specific primitives and laws (XOR, …).

Variant of the Internet Protocol TLS I

The Handshake Protocol

Goal:

- Establish a secure channel over untrusted communication link between client and server.
  - Supposed to provide secrecy and server authenticity, specified by {secrecy} and {authenticity} .

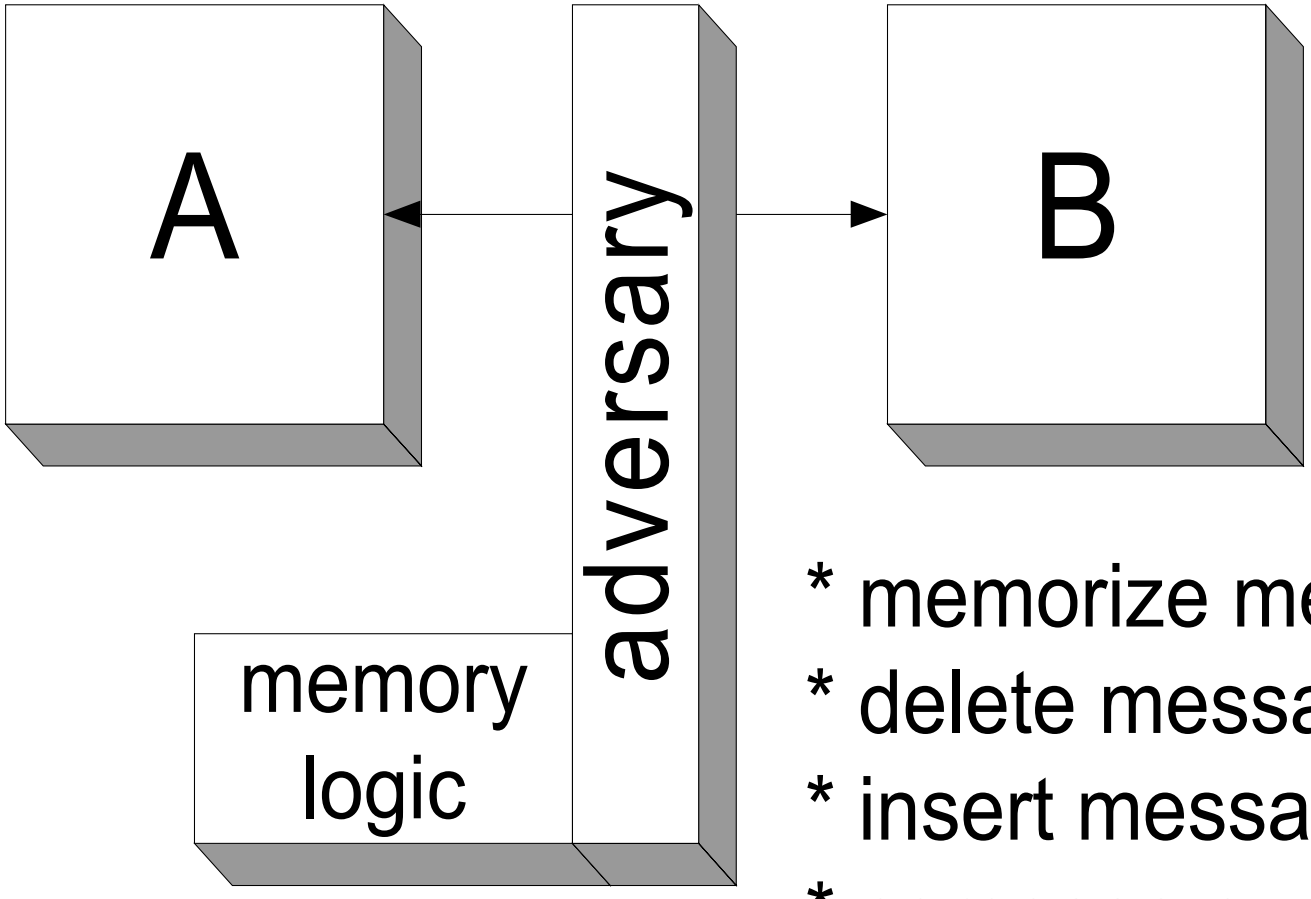
Variant of the Internet Protocol TLS II

- Set of data values Data includes names C and S for each instance C: Client and S: Server.
- Assumes: each client C is given the server name S_i before the ith execution round of the protocol part under consideration.
- Server is not given the client name in advace
  - since no client authenticity is to be provided by the protocol.
- Restrict ourselves to considering the first I executions of protocol.
  - I: arbitrary but fixed natural number.
- Note:
  - each C may be given a different sequence of server names.
  - More precisely: would have to be referred to as $C.S_i$.
- Omit instance prefixing for readability where no confusion can arise.

Variant of the Internet Protocol TLS III

- C and S: variables representing arbitrary names.
- Client and server can run protocol with arbitrary servers and clients.
- Adversary controls communication link between client and server.
  - Captured by enabling adversary to read, delete, and insert messages at corresponding link queue.
    - able to insert any message communicated over the link in adversary's current knowledge.
  - Adversary may perform the protocol with either client or server:
    - by taking on the role of server or client.

Adversary Model

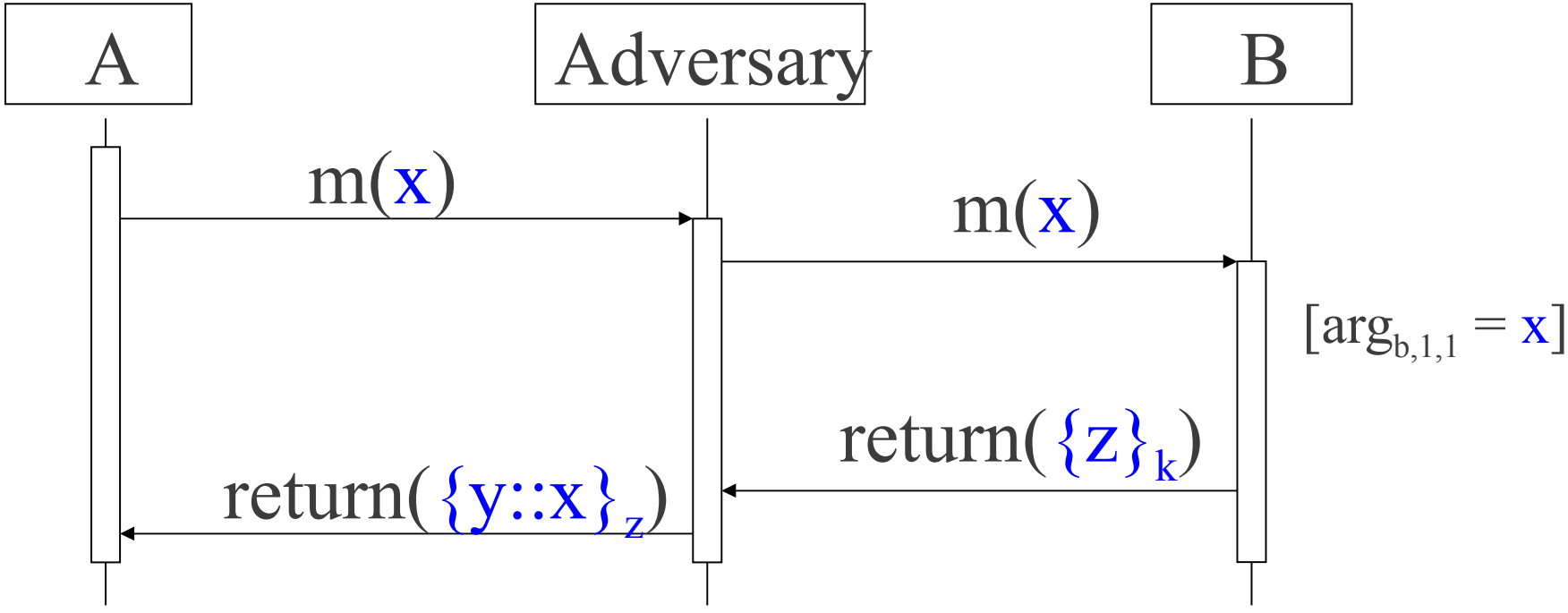


* memorize message
* delete message
* insert message
* compose own message
* use cryptographic primitives

Advanced Adversary

- Note:
  - one may also specify the adversary to actually be a server or client
    - by adding the access threat to a suitable node stereotype attached to relevant object in diagram.
  - Not considered here:
    - we assume all servers and clients are trustworthy.

Protocol: Attack Scenario



Adversary knowledge:

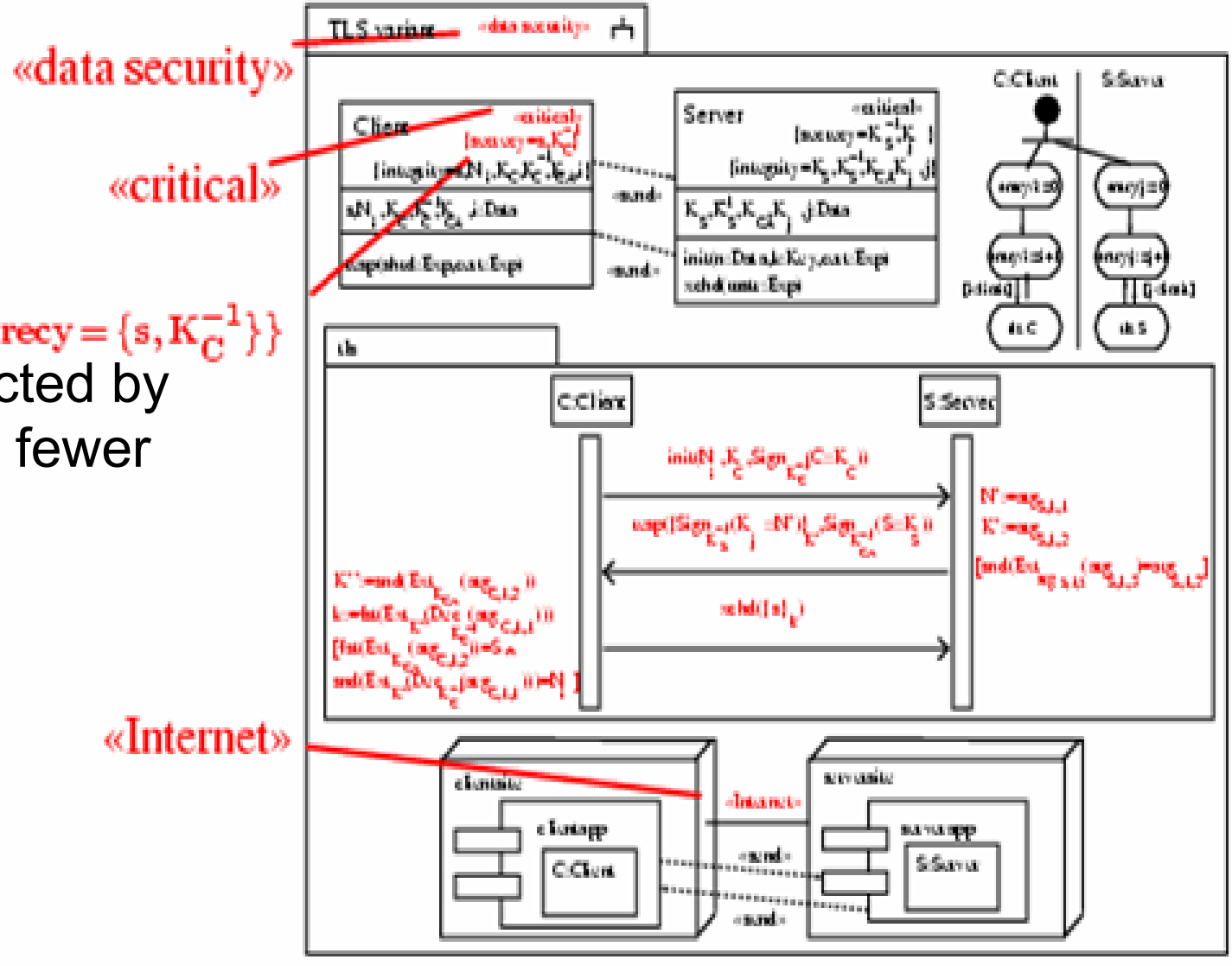
k^{-1}, y, x
$\{z\}_k$, Z

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

Proposed Variant of TLS (SSL)

IEEE Infocom 1999.
Goal:

- send secret protected by session key using fewer server resources.



Handshake Details I

- Each C/S has
  - public key K_C/K_S with associated private key K^{-1}_C / K^{-1}_S
- Assume: C able to obtain K_{CA}[1] guaranteeing CAs integrity.
- S securely obtains certificate $Sign_{K^{-1}_{CA}}(S :: K_S)$ signed by CA
  - contains its name and public key.
- Each client is given
  - sequence of secrets $s_1,...,s_l \in Data^s$ to be transmitted
  - nonces $N_1,...,N_l \in Data$.
- Write $s\_ : Data$ to denote an array with fields s_i in Data.

1 public key of the certification authority

Handshake Details II

- Nonces specified to be created freshly by receiver before protocol execution, as stated by {fresh}.
- Values N_i belong to scope of Client within subsystem specification TLS variant
  - Since expressions N_x, for any subexpression x, only appear within Client object and associated view of sequence diagram.
- Similarly: session keys $k_1,\ldots,k_l \in Keys^s$ given to each server is specified to be fresh.

Enhance Readability

- Leave out :
  - Explicit assignment of initial values to constant attributes:
    - e.g. keys, the nonces, and the values s_x and S_x
    - Instead: constants as attribute names.
      - There for, relevant type names are underlined.
  - Not relevant for security requirements under consideration.
    - Time-stamp
    - Session id
    - Choice of cipher suite and compression method
    - Use of temporary key by S

Enhance Readability

- Recall:
  - For each method msg in diagram and each number n
    - msg_n represents nth argument of operation call msg, most recently accepted according to the sequence diagram.
- Use notation var ::= exp to write expression exp more shortly as var

Behavioral Specification in Sequence Diagrams

- Associate behavioral specifications to their context:
  - Add: name of relevant states in activity diagram next to the sequence diagram.
  - Sequence diagram tls involving the objects C and S_i
    - used to specify activities tls.C, tls.Si for any objects C and S_i.
    - Sequence diagram specified w.r.t. S_i,
      - this is where C sends its messages, depending on attr. i.

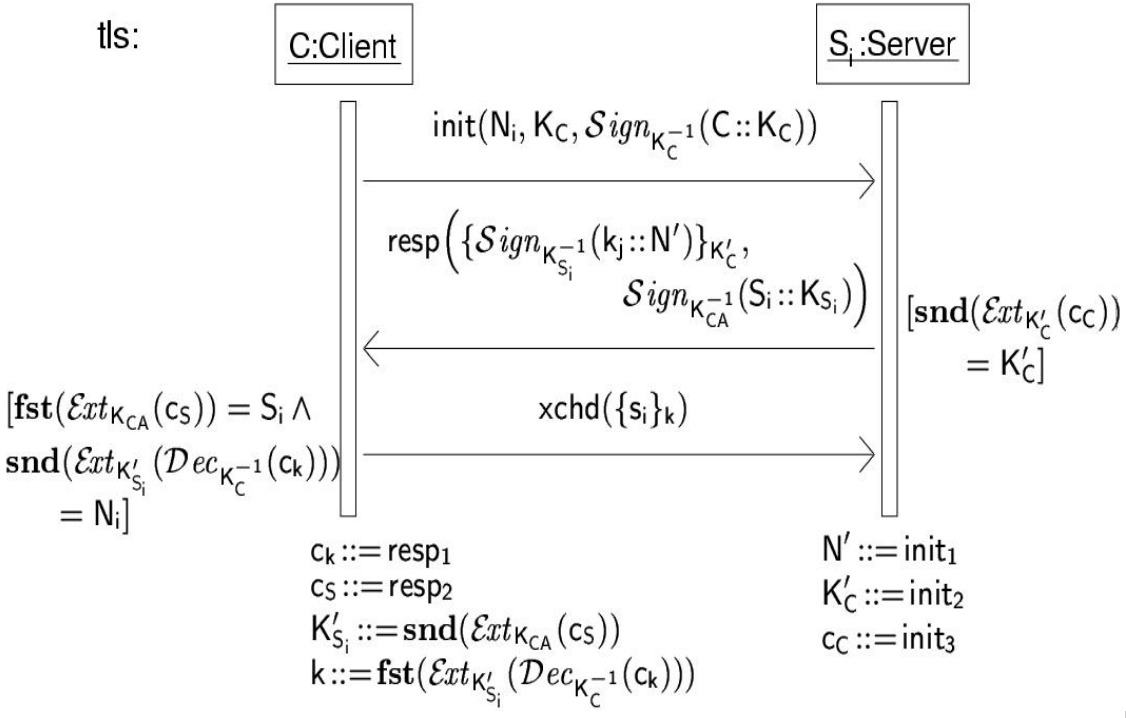
    In contrast
    - Activity diagram specified w.r.t. S, because:
      - all clients and servers are executed in parallel (independently of the value of i in any C.)

Behavioral specification Note

- Well-defined to use a sequence diagram specifying S_i to specify activity tls.S
  - because S_i, in the sequence diagram, does not use parameter i.

Protocol Procedure I

- Consider
  - ith execution round C(i) of client C
  - jth execution round $S_i(j)$ of server S_i
  - assume $S_i(j) = S$.
- C aims to communicate with instance S_i (jth execution round).
- C initiates the protocol by sending init(...) to S.
- Suppose condition $[snd(...)=K'_C]$ holds:
  - K_C contained in the signature matches the one transmitted in clear.
- S sends message resp(...) back to C

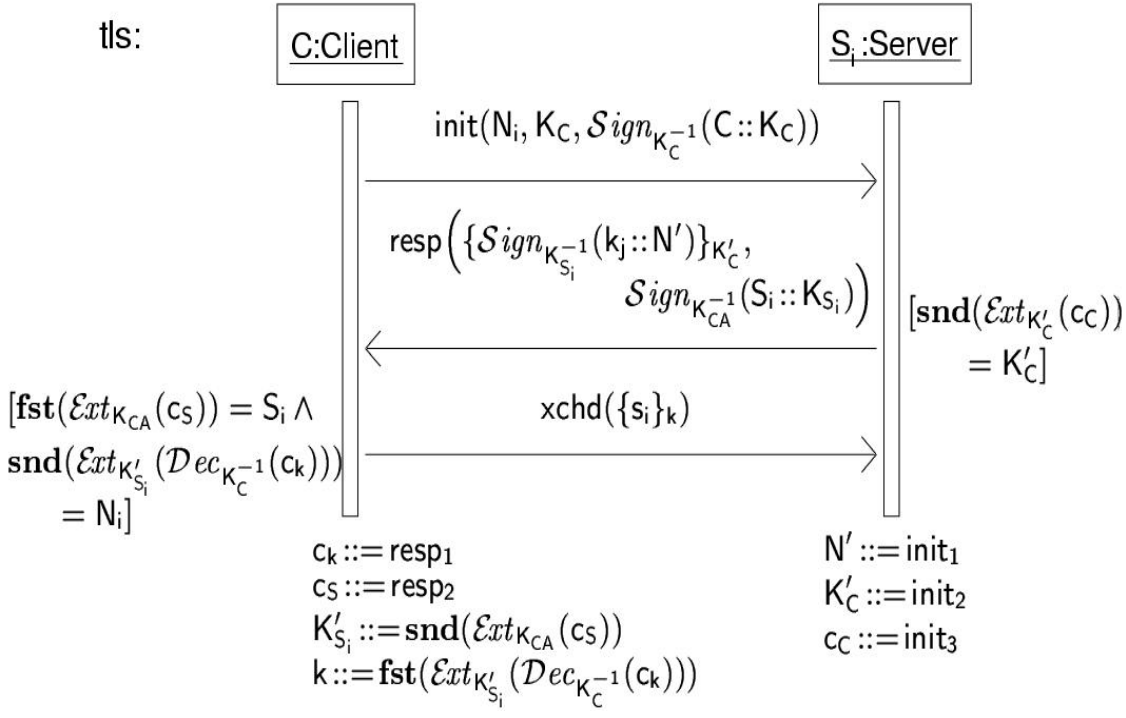


Protocol Procedure II

- Suppose condition [fst(...)=S ∧ snd(...)=Ni] holds
  - Certificate is actually for S and the correct nonce is returned.

  Then C sends xchd(...) to S.

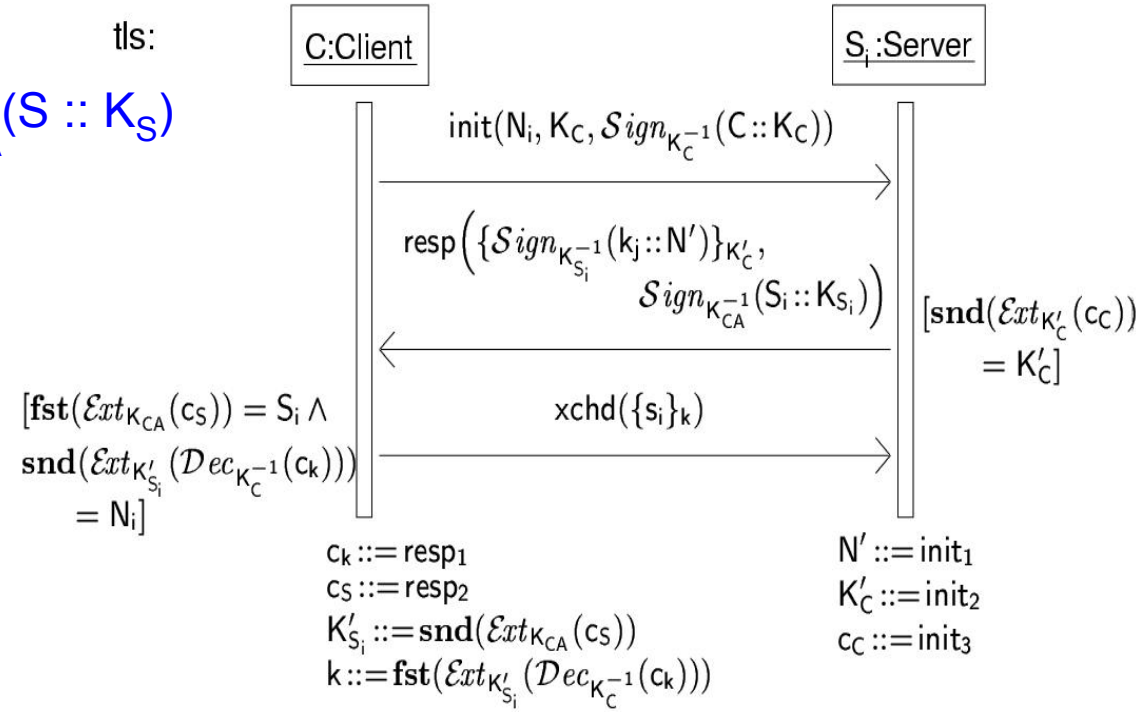
- If any check fails:
  - Respective protocol participant stops execution of the protocol.



Alternative Notation

- Traditional informal notation e.g. in [NS78]
- Protocol would be written as follows:
  - $C \rightarrow S : N_i, K_C, Sign_{K^{-1}_C}(C :: K_C)$
  - $S \rightarrow C : \{Sign_{K^{-1}_S}(k_j :: N_i)\}_{K_c}, Sign_{K^{-1}_{CA}}(S :: K_S)$
  - $C \rightarrow S : \{s_i\}_{k_j}$.
- May seem simpler
- Needs to be interpreted with care [Aba00].

tls:



[NS78] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. Communications of the ACM, 21(12):993-999, 1978.

[Aba00] M. Abadi. Security protocols and their properties. In F. L. Bauer and R. SteinbrÄuggen, editors, Foundations of Secure Computation, pages 39-60. IOS Press, Amsterdam, 2000. 20th International Summer School, Marktoberdorf, Germany.

Alternative Notation Example Interpretation

1. $C \rightarrow S : N_i, K_C, Sign_{K^{-1}_C}(C :: K_C)$
2. $S \rightarrow C : \{Sign_{K^{-1}_S}(k_j :: N_i)\}_{K_c}, Sign_{K^{-1}_{CA}}(S :: K_S)$
3. $C \rightarrow S : \{s_i\}_{k_j}$.

(1) C sends $N_i, K_C, Sign_{K^{-1}_C}(C :: K_C)$ to the network,

- intended recipient S

(2) S expects a message of form $N, K_1, Sign_{K^{-1}_3}(X :: K_2)$,

- seemingly coming from C.

Message sent over untrusted network: cannot conclude e.g. $K_1 = K_C$.

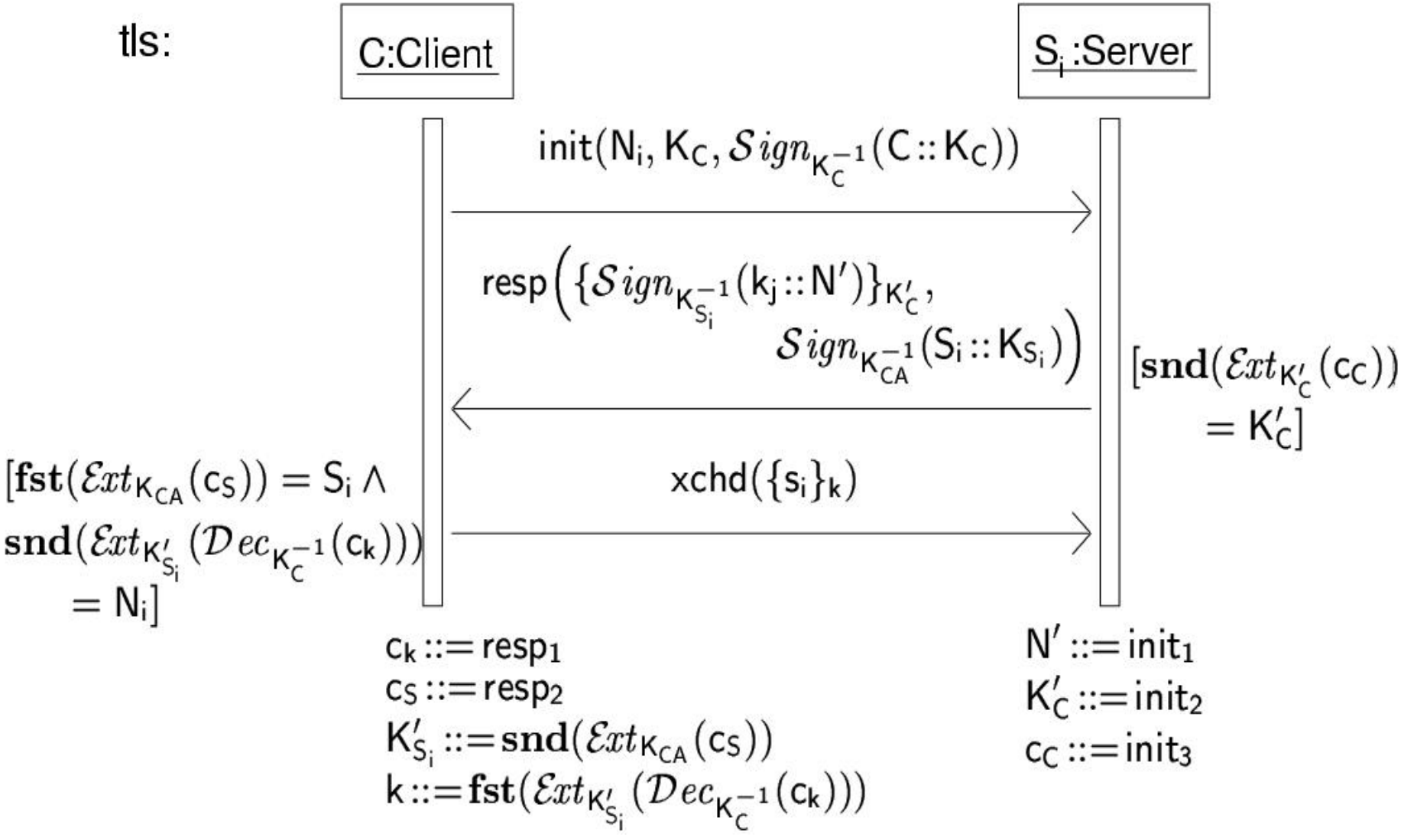
Therefore, need to make assumptions: e.g. S checks that occurrences of K_C do indeed coincide ($K_1 = K_2 = K_3$).

Notes

- Major source of security weaknesses in practice: Misinterpretation of protocol specifications.
  - Assure assumptions are understood by implementor of a protocol
- Aim: Use UML as a notation that is widely used among software developers beyond the community of security experts
  - without deviating from its standard definition any more than necessary.
- UML sequence diagram semantics does not entail the assumptions.
  - Include assumptions explicitly
    - By referring to sent and received values in different ways, and
    - Including checks in sequence diagram
    - To ensure that they actually coincide.

Protocol

tls:



The Flaw

- Analyzing the specified protocol for relevant security requirements using the automated analysis tools[1]:
  - Observed Man-in-the-middle-attack.
- Theorem: For given C and i, the UML subsystem T, given in IEEE Infocom 1999, does not preserve the secrecy of s_i from adversaries of type A = default with $\{K_S, K_A, K^{-1}_A\} \subseteq K^p_A$.
  - Protocol does not provide secrecy of s_i, against a realistic adversary.

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 6.2.1

Man-in-the-Middle Attack

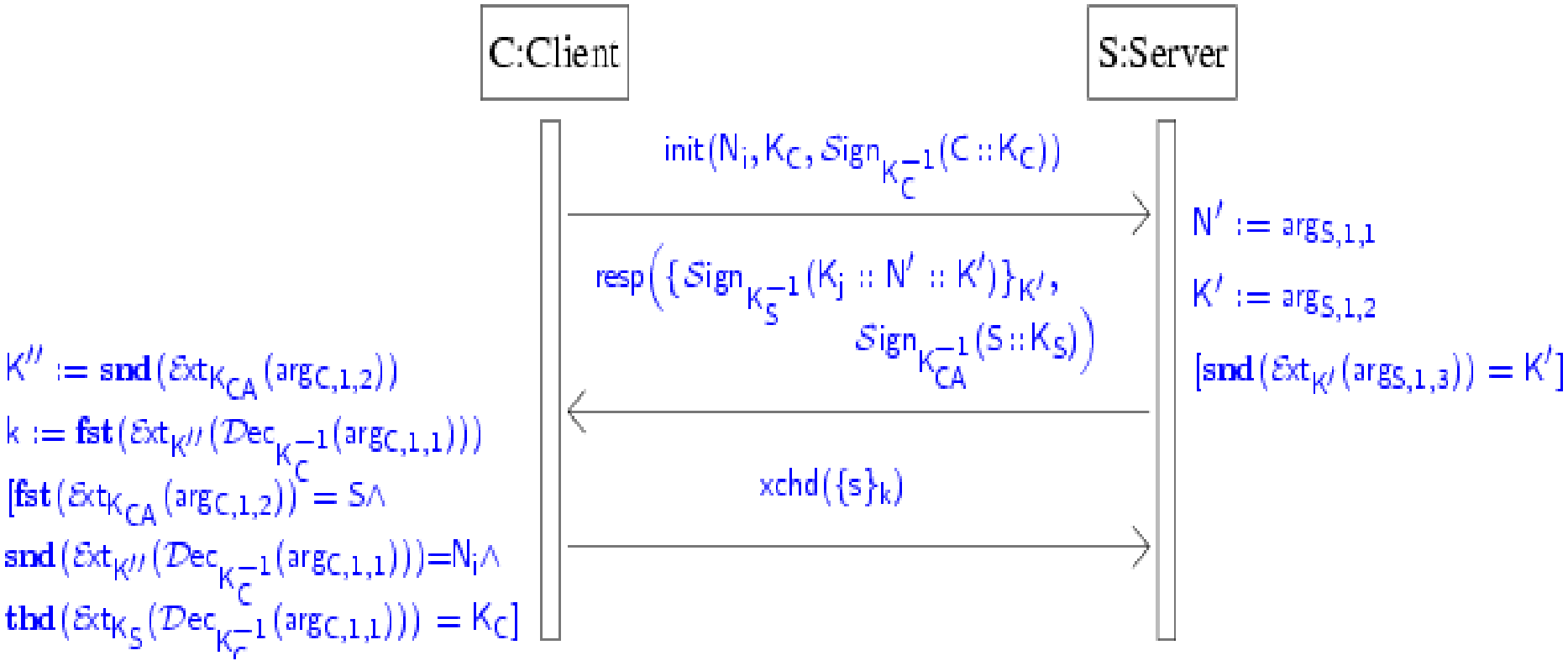
$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

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

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

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

The Fix



Can prove that secure.

The Flaw Proof I

- Show existence of successful attacker adv.
- Fix instances C and S with execution rounds i and j ($S_i = S$)
  - Denote link between C and S as l_{CS}.
- Adversary adv proceeds as follows:
  - Message of form $S.init(N_i, K_C, Sign_{K^{-1}_C}(C :: K_C))$ in l_{CS}
    - replaced by $S.init(N_i, K_C, Sign_{K^{-1}_A}(C :: K_A))$;
      - Public key K_C of C replaced by public key K_A of A at each occurrence and as signature key.

The Flaw Proof II

- When S sends message $resp(\{Sign_{K^{-1}_S}(k_j :: init_1)\}_{K_A}, Sign_{K^{-1}_{CA}}(S :: K_S))$ using K_A to encrypt session key k_j,
  - adv can obtain k_j and replace message by $resp(\{Sign_{K^{-1}_S}(k_j :: init_1)\}_{K_C}, Sign_{K^{-1}_{CA}}(S :: K_S))$
- When C subsequently returns $\{s_i\}_{k_j}$, adv can extract secret s_i
  - And forward message.
- Adversary machine that achieves this is defined as follows:

The Flaw
Proof: Adversary Machine

$$
\begin{array}{l}
\textbf{Rule Exec } adv: \\
\textbf{do} - \textbf{in} - \textbf{parallel} \\
\quad \textbf{if } \mathsf{linkQu}_{\mathcal{T}}(l_{\mathsf{CS}}) = \{\!\{ e \}\!\} \wedge \textbf{msgnm}(e) = \mathsf{S.init} \\
\qquad \textbf{then } \mathsf{linkQu}_{\mathcal{T}}(l_{\mathsf{CS}}) := \{\!\{ \mathsf{S.init}(\textbf{Arg}_1(e), \mathsf{K}_A, \\
\qquad\qquad \mathcal{S}ign_{\mathsf{K}_A^{-1}}(\textbf{fst}(\mathcal{E}xt_{\textbf{Arg}_2(e)}(\textbf{Arg}_3(\mathsf{e}))) :: \mathsf{K}_A)) \}\!\} \\
\quad \textbf{if } \mathsf{linkQu}_{\mathcal{T}}(l_{\mathsf{CS}}) = \{\!\{ e \}\!\} \wedge \textbf{msgnm}(e) = \mathsf{C.resp} \textbf{ then} \\
\qquad \textbf{do} - \textbf{in} - \textbf{parallel} \\
\qquad\quad \mathsf{linkQu}_{\mathcal{T}}(l_{CS}) := \{\!\{ \mathsf{C.resp}(\{\mathcal{S}ign_{\mathsf{K}_A^{-1}}(\textbf{Arg}_1(e))\}_{K_C}, \textbf{Arg}_2(e)) \}\!\} \\
\qquad\quad local := \{\!\{ \textbf{fst}(\mathcal{E}xt_{K_S}(\mathcal{D}ec_{\mathsf{K}_A^{-1}}(\textbf{Arg}_1(e)))) \}\!\} \\
\qquad \textbf{enddo} \\
\quad \textbf{if } \mathsf{linkQu}_{\mathcal{T}}(l_{\mathsf{CS}}) = \{\!\{ e \}\!\} \wedge \textbf{msgnm}(e) = \mathsf{S}.xchd \textbf{ then} \\
\qquad secret := \{\!\{ \mathcal{D}ec_{local}(\textbf{Arg}_1(e)) \}\!\} \\
\textbf{enddo}
\end{array}
$$

- Thus adversary gets to know secrets $\mathsf{C.s_i}$.

The Fix

- Change protocol to get a specification T'
  - Substituting $k_j :: N_i$ in message resp by $k_j :: N_i :: K_C$ and
  - Including a check regarding this new message part at client.
- Public key K_C of C is representatively for identity of C.
- One could also use $k_j :: N_i :: C$ instead.
- Traditional informal notation, of modified protocol:

(1) $C \rightarrow S : N_i, K_C, Sign_{K^{-1}_C}(C :: K_C)$

(2) $S \rightarrow C : \{Sign_{K^{-1}_S}(k_j :: N_i :: K_C)\}_{K_c}, Sign_{K^{-1}_{CA}}(S :: K_S)$

(3) $C \rightarrow S : \{s_i\}_{k_j}$.

Conversely

- C will only send the secret under the session key received if signed by the server concatenated with public key of C.
- This certificate, S only sends out encrypted under the same public key
  - adversary cannot decrypt.
- Essential:
  - session keys differ for different iterations of the protocol.

Informal Explanation

Certificate sent in first message

- Self-signed
- Does not provide full client authenticity.
- Adversary can still send certificate to S claiming that public key of the adversary belongs to C.
- However:
  - When adversary forwards the response from S to C.
  - Server signed certificate contains the public key received by S in first message.
  - If adversary again forwards this certificate to the C.
    - C will notice a false public key has been submitted on C's behalf and stop execution because the newly introduced check fails.

Little Confidence

- Arguments may convince that the particular attack is prevented by the modification
- Little confidence that the modified protocol is immune against all other possible attacks.
- To give confidence
  - Prove formally that Protocol specification is secure with regards to adversary model.
- More specifically:
  - show that the protocol specification fulfills constraints associated with <<data security>> w.r.t. adversary.
  - Proving this for {secrecy}.
    - {integrity} and {authenticity} can be established similarly.

Notes on {integrity}

- {integrity} goals
  - Adversary should not be able to make the atttributes take on values previously known only to him
  - Straightforward to verify
    - Only concerns attributes that remain constant
      - apart from i and j, simply counted upwards.
  - Could formulate other integrity requirements.
    - e.g.value C.k coincides with $S.k_j$, we do not consider this here.

Theorem TLS Variant Is Secure

- Theorem: Given a particular execution of repaired TLS variant subsystem T' including all client/server instances, a client C, and number I with $S=S_I$.
- Suppose server S, is in Jth execution round(ER) in the current execution when C, Ith ER, initiates the protocol ($C.i = I$ and $S.j = J$).
- Then this execution of T' preserves secrecy of $C.s_I$ against adversaries of type A = default whose previous knowledge K^p_A fulfills:
- We have:

$$\Big(\{C.s_I, K^{-1}_C, K^{-1}_S\} \cup \{S.k_j : j \geq J\} \cup \{\{Sign_{K^{-1}_S}(X :: C.N_I :: K_C)\}_{K_C} : X \in Keys\}\Big) \cap K^p_A = \emptyset;$$

- for any $X \in Exp$, $Sign_{K^{-1}_C}(C :: X) \in K^p_A$ implies $X = K_C$, and
- for any $X \in Exp$, $Sign_{K^{-1}_{CA}}(S :: X) \in K^p_A$ implies $X = K_S$.

Theorem In Detail I

- Condition means:
  - Previous adversary knowledge K^p_A may not contain
    - Current secret $C.s_I$
    - Secret keys K^{-1}_C ;K^{-1}_S of sender and receiver,
    - Current and future session keys $S.k_j$,
    - Any encryptions of form $\{Sign_{K^{-1}_S}(X :: C.N_I :: K_C)\}_{K_C}$,
    - Any signatures $Sign_{K^{-1}_S}(C :: X)$ (except for $X = K_C$) and $Sign_{K^{-1}_{CA}}(C :: X)$ (except for $X = K_S$).
- Result covers possibility: adversary may gain information from previous or parallel executions of protocol,
  - possibly with other instances of C or S.

Theorem In Detail II

- Parallel executions of other instances:
  - Restrictions on adversary knowledge allow adversary to simulate other instances of the two classes.
    - by giving adversary access to their private keys and certificates.
- Previous executions,
  - Note: previous adversary knowledge K^p_A refers to knowledge of the adversary before overall execution of the system,
    - Not at the point of the system execution where C.i=I and S.j=J (see the definition[1] or discussion and corollary later on).

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.4

Theorem Conditions

In particular:

Condition

- Doesn't prevent adversary from remembering information gained from earlier iterations or Current iteration of the protocol and use it in later iterations.
- It does assume: adversary doesn't know $\{Sign_{K^{-1}_S}(k_j :: N_i :: K_C)\}_{K_C}$ of server in the current protocol run before current protocol.
  - Assumption is necessary, otherwise the attack would still work:
    - Adversary would already have the certificate the client expects (which includes the client's key K_C) and can in addition still get the current session key from the server
      - as in earlier attack by sending $N_i :: K_A :: Sign_{K^{-1}_A}(C :: K_A)$ containing adversary's key K_A to S.

Theorem "forward security"

- Since we allow
  - $S.k_j$ for $j < J$ to be included in previous adversary knowledge K^p_A
- Theorem establishes a form of "forward security"
  - Compromise of a current key doesn't necessarily expose future traffic.
- Not sufficient to only require $S.k_j \notin K^p_A$.
  - because adversary may initiate an intermediate interaction with S to increase its counter j.

Theorem "rely-guarantee"

- Statement of theorem concerns particular instances of Client and Server and particular execution rounds.
- Formulated in a "rely-guarantee" way,
  - Stating that if the knowledge previously acquired by adversary satisfies the conditions, then this execution preserves secrecy.
  - Allows to consider security mechanisms such as security protocols in system context.
    - To do this: Specify explicitly which values the remaining part of the system has to keep secret to function securely.
    - e.g.: Theorem needs to assume that the CA doesn't issue any false certificates,
      - Third pre-condition in the theorem.

Conditions Conclusion

- Conditions only concern previous knowledge of the adversary before overall execution of the system,
  - Follows: adversary knowledge before each iteration of the system satisfies conditions as well.
  - Each iteration of execution of the system preserves conditions on the adversary knowledge:
    - If conditions on adversary knowledge were violated in iterations before the one under consideration:

      Result of the theorem would not be valid.
      - For each “current“ iteration.
      - Since theorem holds: cannot be the case.

Proof Theorem: TLS Variant Is Secure

Proof: Use Theorem 7.27 [1]. We show the following claim:

- For each knowledge set $K^e_{adv}(A)$ for adversary adv of type A, after overall execution e of T', whose previous knowledge K^p_A satisfies the conditions in the statement of the theorem,
  - there exists a subalgebra X_0 that is minimal w.r.t. the subset relation among subalgebras X of Exp fulfilling the following two conditions,* such that X_0 contains $K^e_{adv}(A)$.

* $K^e_{adv}(A)$ is not contained in every such subalgebra X, because the actual messages exchanged may differ depending on the adversary behavior.

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 7.5.2

Proof
Claim: Condition (1)

Condition (1) is required to hold:

- $K^p_A \cup \{c.N_i, K_c, \mathrm{Sign}_{K^{-1}_C}(c :: K_c) : i \in \mathbb{N} \wedge c \in \mathrm{Client}\}$

$\cup \{\mathrm{Sign}_{K^{-1}_{CA}}(s :: K_s) : s \in \mathrm{Server} \wedge (s = S \Rightarrow K_s = K_S)\}$

$\cup \{\{c.s_i\}_k : k \in \mathrm{Keys} \wedge i \in \mathbb{N} \wedge c \in \mathrm{Client}$

$\wedge \exists K \in \mathrm{Keys}, E \in \mathrm{Exp}, E' \in X$

$(\mathrm{Sign}_{K^{-1}_{CA}}(E) \in X \wedge \mathrm{fst}(E) = c.S_i \wedge \mathrm{snd}(E) = K$

$\wedge \mathrm{Ext}_K(\mathrm{Dec}_{K^{-1}_C}(E')) = (k, c.N_i, K_c)\ X)\}$

$\subseteq X.$

Proof
Claim: Condition (2)

Requires that

- For each $j \in N$ and s : Server and

  For an associated fixed key $k_{j,s} \in Keys \cap X$,

  - a fixed expression $x_{j,s} \in Exp$, and
  - a fixed nonce $n_{j,s} \in Data \cap X$ with $Sign_{K^{-1}_{j,s}}(x_{j,s} :: k_{j,s}) \in X$,*

  We have

  - $\{Sign_{K^{-1}_{s}}(s.k_j :: n_{j,s} :: k_{j,s})\}k_{j,s} \in X$.

* Condition (1) guarantees existence of these unique expressions associated with each $j \in \mathbb{N}$ and s : Server.

Proof
Claim: Note On Condition (2)

- Possible that $k^{-1}_{j,s} \in K^{e}_{adv}(A)$,
  - But then $k_{j,s} \neq K_c$ for any client c
    - because $K^{-1}_{c} \notin K^{e}_{adv}(A)$

      since $K^{-1}_{c} \notin K^{p}_{A}$ and K^{-1}_{c} is never sent out.
  - c will notice something is wrong in the corrected protocol
    - because counter j is increased, adversary **cannot** make the server publish another signature with the same k_j and correct K_c.

Proof of Claim: Intuitively I

- Claim holds because each knowledge set $K^e_{adv}(A)$ is by definition the subalgebra of the algebra of expressions Exp built up from K^p_A in interaction with the protocol participants during the protocol run e.
- What knowledge adversary can gain from interaction with the protocol participants.
  - From first message of c, adversary can learn expressions $c.N_i$, K_c, and $Sign_{K^{-1}_c}(c :: K_c)$.
  - From first message of s, the adversary can firstly learn $Sign_{K^{-1}_{CA}}(s :: K_s)$.

Proof of Claim: Intuitively II

Secondly,

- For each encryption key $K \in Keys$ in the knowledge of the adversary
  - such that adversary knows $Sign_{K^{-1}}(x :: K)$ for some $x \in Exp$,

  and for each N known to the adversary, adversary learns

  $\{Sign_{K^{-1}_s}(s.k_j :: vN :: K)\}_K \in X$,

  but only a unique such expression for a given
  - server s,
  - protocol run e, and
  - transaction number j,

  because transaction number j is increased as long as protocol is iterated (reflected by the fact that X_0 is required to be minimal).

Proof of Claim: Intuitively III

- From second message from c, for each encryption key $K \in Keys$ such that,
  - $Sign_{K^{-1}_{CA}}(E)$ is known to adversary for $E \in Exp$ with $fst(E) = c.S_i$ and $snd(E) = K$, such that
    - $\exists\, E' \in Exp$ which is known to adversary such that $Ext_K(Dec_{K^{-1}_c}(E')) = (k, c.N_i, K_c)$ for some $k \in Keys$,
- Adversary learns $\{c.s_i\}_K \in X$.

Proof of Claim: Intuitively IV

- Since no other messages are sent out by the system
  - Claim holds by definition of the adversary knowledge as the algebra generated by exchanged messages and initial adversary knowledge.
- Completes proof.

Proof of Claim: Contradiction I

- Sufficient to show: $C.s_I \notin X_0$ for every X_0 defined above,
  - Because $K_A(A) := \bigcup_{adv,e} K^e_{adv}(A)$ is contained in union of all such X_0 by the above argument.
- Proof by contradiction:
  - Fix such an X_0
  - Assume:
    - $C.s_I \in X_0$.
    - X_0 is defined to be a minimal subalgebra
      - satisfying conditions (1) and (2)

Proof of Claim: Contradiction II

- Recall Algebra of expressions Exp[1]:
  - as a free algebra it follows that $C.s_I$ is different from any other expression not containing it
    - since no equation with such an expression is defined.
- We have $C.s_I \neq c.s_i$ for any client c and number i with $c \neq C$ or $i \neq I$.
  - Only occurrence in conditions defining X_0 in a minimal way,
    - where $C.s_I$ may be introduced as a subterm,

    is in the requirement that X_0 contains $\{C.s_I\}_k$ for each key $k \in Keys$ for which there exist $K \in Keys$, $E \in Exp$, $E' \in X_0$ such that
    - $Sign_{K^{-1}_{CA}}(E) \in X_0 \wedge fst(E) = S \wedge snd(E) = K$
      $\wedge\ Ext_K(Dec_{K^{-1}_c}(E')) = (k, C.N_I, K_C)$

    in condition (1).

1 Jan Jürjens, Secure Systems Development with UML, Springer 2004. Sect. 3.3.3

Proof of Claim: Contradiction III

Assumption $C.s_I \in X_0$ implies

- $\exists k \in Keys$ for which there exist $K \in Keys$, $E \in Exp$, $E' \in X_0$ such that
  - $Sign_{K^{-1}_{CA}}(E) \in X_0 \wedge fst(E) = S \wedge snd(E) = K$
    $\wedge\ Ext_K(Dec_{K^{-1}_c}(E')) = (k, C.N_I, K_C)$.
- By definition of X_0 and assumption on K^p_A
  $Sign_{K^{-1}_{CA}}(E) \in X_0 \wedge fst(E) = S \wedge snd(E) = K$
  implies $K = K_S$
  - (since any expression of this form in K^p_A must satisfy this, and any such expression introduced in X_0).

Proof of Claim: Contradiction IV

- Similarly, $E' \in X_0$ with $Ext_{K_S}(Dec_{K^{-1}_c}(E'))=(k, C.N_I, K_C)$ implies $k=S.k_j$ for some j
  - $E' \notin K^p_A$ by assumption on adversary knowledge K^p_A
    - because K^{p-1}_S never communicated, and
    - $\{Sign_{K^{-1}_S}(S.k_j :: n_{j,s} :: k_{j,s})\}_{k_{j,s}}$ (condition (2)) only expression with sub-term of form $Sign_{K^{-1}_S}(k :: n_{j,s} :: k_{j,s})$ introduced (in this term $n_{j,s} = C.N_I$ and $k_{j,s} = K_C$).
  - Conclude $j \geq J$
    - by assumption: S is in its Jth execution round when C is in its Ith.
    - by requirement: $C.N_i$ should be fresh (each value distinct from any other).
  - By assumption on K^p_A, we have

    $S.k_j \notin K^p_A$ since $j \geq J$,
  - Thus adversary must have learned $S.k_j$ in protocol interaction.

Proof of Claim: Contradiction V

- By Freshness on $S.k_j$:
  - Only message containing $S.k_j$ is a term of form $\{Sign_{K^{-1}_S}(S.k_j :: n_{j,s} :: k_{j,s})\}k_{j,s}$.
- By condition (2) and minimality of X_0,
  - We know $n_{j,s} = C.N_I$ and $k_{j,s} = K_C$ for any such term.
  - Term has to be decrypted with K^{-1}_c to get $S.k_j$.
  - Only participant possessing K^{-1}_c and could provide this service for the adversary is C.
    - other's don't have K^{-1}_c in initial knowledge, and it is never exchanged.
- None value in $\{Sign_{K^{-1}_S}(S.k_j :: n_{j,s} :: k_{j,s})\}k_{j,s}$ sent out by C.
- Thus $K^{-1}_c \in K^p_A$: contradicts initial assumption about K^p_A ↯

Proof of Claim: Satisfied Conditions I

- Adversary knowledge before each iteration satisfies conditions as well:

(1) Ith execution round of client C

- no data of form $X.s_i$ is output except $C.s_I$ (is kept secret from adversary)
- Secret keys K^{-1}_C, K^{-1}_S ($\forall$ C, S) are never output.
- Key $S.K_J$: only sent out during Jth executing round of S.
  - Above theorem: in that round key is not leaked to adversary
    - Otherwise adversary would gain knowledge of $C.s_I$ by decrypting contents of xchd message.
- Expression of form $\{Sign_{K^{-1}_S}(X :: C.N_I :: K_C)\}_{K_C}$ (for $X \in Keys$) is only output in Ith execution round of C (and is of no use in any later round).

Proof of Claim: Satisfied Conditions I

Adversary knowledge before each iteration satisfies conditions as well:

(2)For any $X \in Exp$, $Sign_{K^{-1}_S}(X :: C)$ is sent out only for $X = K_C$

- K^{-1}_C not sent out at all.

(3) For any $X \in Exp$, $Sign_{K^{-1}_{CA}}(X :: C)$ is sent out only for $X = K_S$

- K^{-1}_{CA} not sent out at all.

Generalization of Theorem

Assume: clients and servers are finite:

- Any execution of T' (over all clients,servers and all execution rounds) preserves secrecy of each $C.s_I$ (C : Client, $1 \leq I \leq I$)
  - against adversaries of type A = default with previous knowledge K^p_A before overall execution T'

  fulfills the following conditions:

We have: $\left(\{K^{-1}_C ; K^{-1}_S, c.s_i; s.k_j. \{Sign_{K^{-1}_S}(k_j :: N_i :: K_C)\}_{K_C}:\right.$

$\left. c{:}Client \wedge s{:}Server \wedge 1 \leq i \leq I \wedge 1 \leq j \wedge X \in Keys\right) \cap K^p_A = \emptyset$

- for any $X \in Exp$ and any c : Client, $Sign_{K^{-1}_C}(c :: X) \in K^p_A$ implies $X = K_C$.
- for any $X \in Exp$ and any s : Server, $Sign_{K^{-1}_{CA}}(s :: X) \in K^p_A$ implies $X = K_S$.

Proof Of Generalization

- Given an execution e of T', a client C, and a number I.
- S_I = S for a server S
- Within e, when C.i = I,
  - we have S.j = J for a number J.
- Conditions on previous adversary knowledge in the generalization imply those of previous theorem
  - ➔ Can directly apply theorem. □

Generalization of Theorem Explanation

- Condition generalizes that of Theorem to arbitrary clients, servers.
- Protocol rounds of each client c and server s do not have to correspond in any particular way.
  - Any combination of c, s, secrets $c.s_i$, and session keys $s.k_j$ may occur.
  - Same session key is not to be used repeatedly.
    - In particular, $C.s_I$ under consideration could be transmitted encrypted under $s.k_j$ for any s and server round j.
    - Need to assume $s.k_j \notin K^p_A$ for any s and j.
    - Again: K^p_A denotes knowledge of adversary prior to even the first execution round of the protocol.
    - Session keys: not leaked during any of the protocol runs between trustworthy participants follows from our result.

Automated Security Analysis: Adversary Knowledge

- Specify set K_A^0
  - initial knowledge of an adversary of type A.
- Let K_A^{n+1} be the Exp-subalgebra
  - generated by K_A^n and expressions received after n+1st protocol iteration.

Definition (Dolev, Yao 1982).
S keeps secrecy of M against attackers of type A if there is no n with $M \in K_A^n$.

Security Analysis in First-order Logic

Idea:

- approximate set of possible data values flowing through system from above.
- Predicate knows(E)
  - adversary may get to know E during execution of the protocol.
- For any secret s, check whether can derive knows(s) using automated theorem prover.

First-order Logic: Basic Rules

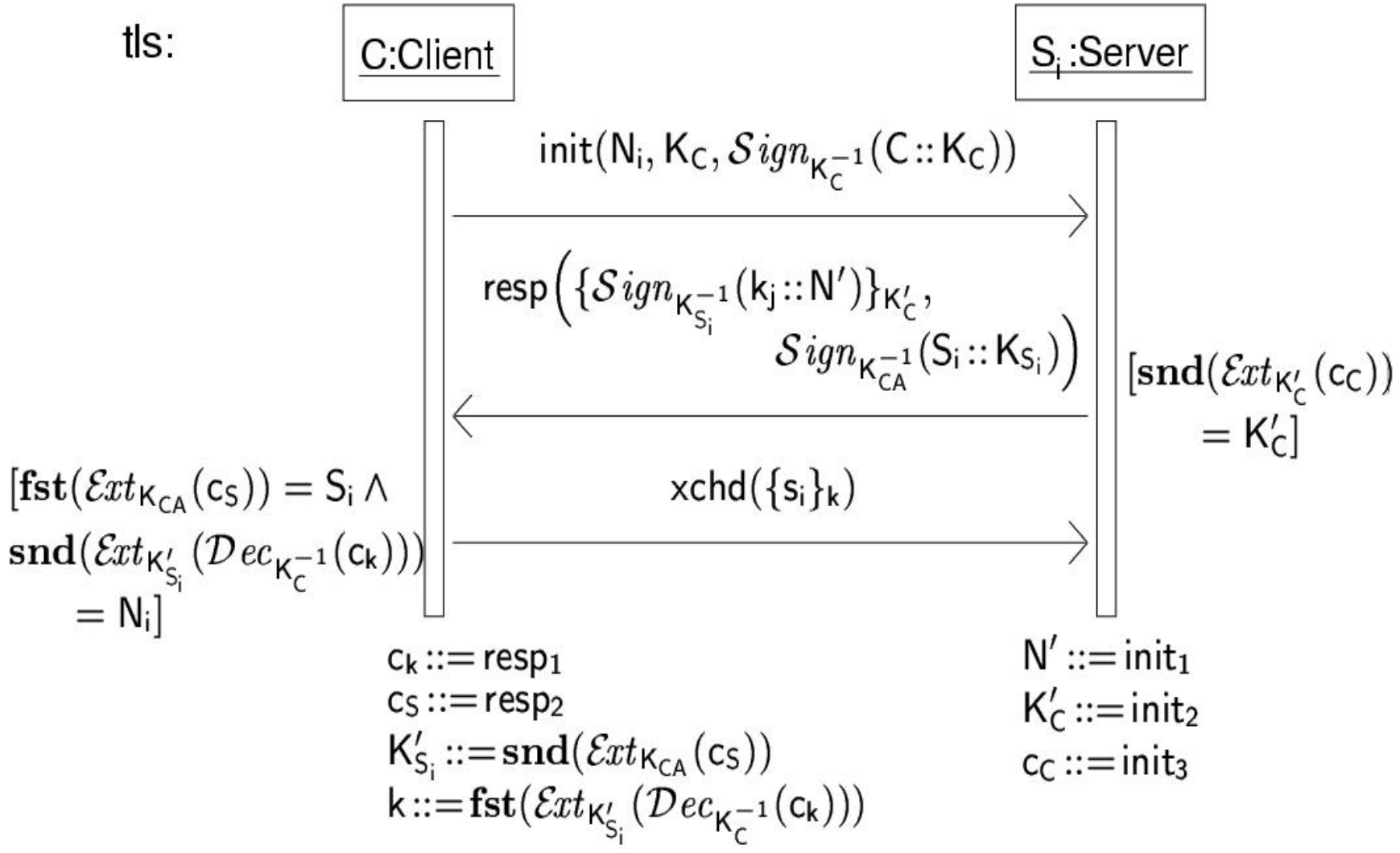
Initial adversary knowledge (K^0):

- Define knows(E) for any E initially known to the adversary (protocol-specific, e.g. K_A, K_A^{-1}).
- Define above equations.
- For evolving knowledge (K^n) define

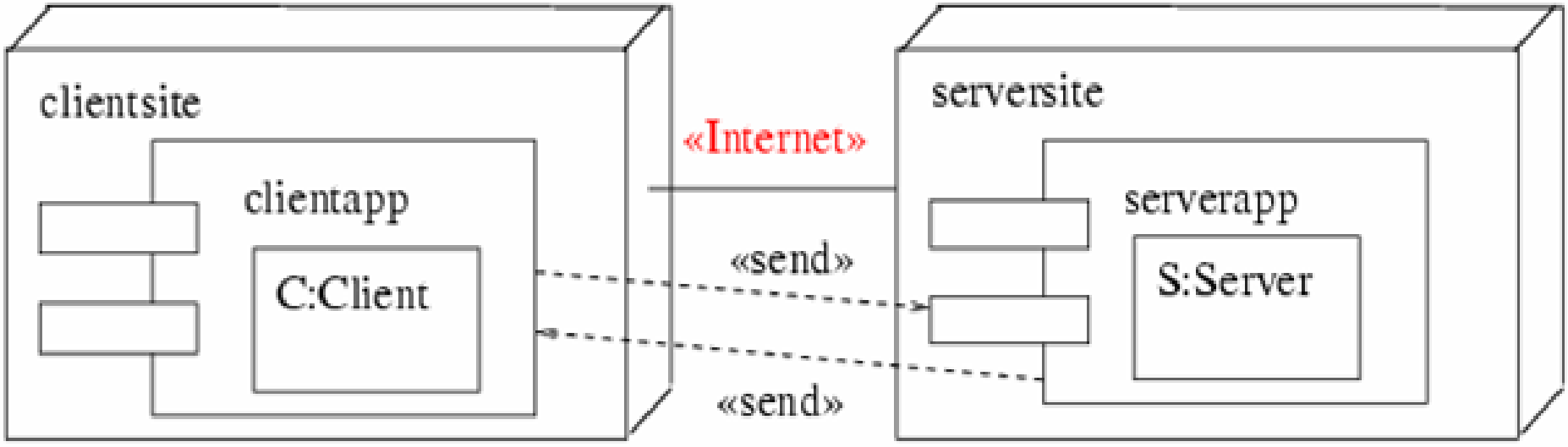
$$\forall E_1,E_2.(knows(E_1) \wedge knows(E_2) \Longrightarrow knows(E_1::E_2) \wedge knows(\{E_1\}_{E2}) \wedge knows(Dec_{E2}(E_1)) \wedge knows(Sign_{E2}(E_1)) \wedge knows(Ext_{E2}(E_1)))$$

$$\forall E.(knows(E) \Longrightarrow knows(head(E)) \wedge knows(tail(E)))$$

Given Sequence Diagram …



… and Physical Layer Model …



Deployment diagram.

Derived adversary model: read, delete, insert data.

… Translate to 1st Order Logic

Connection (or statechart transition)

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

followed by TR2 gives predicate

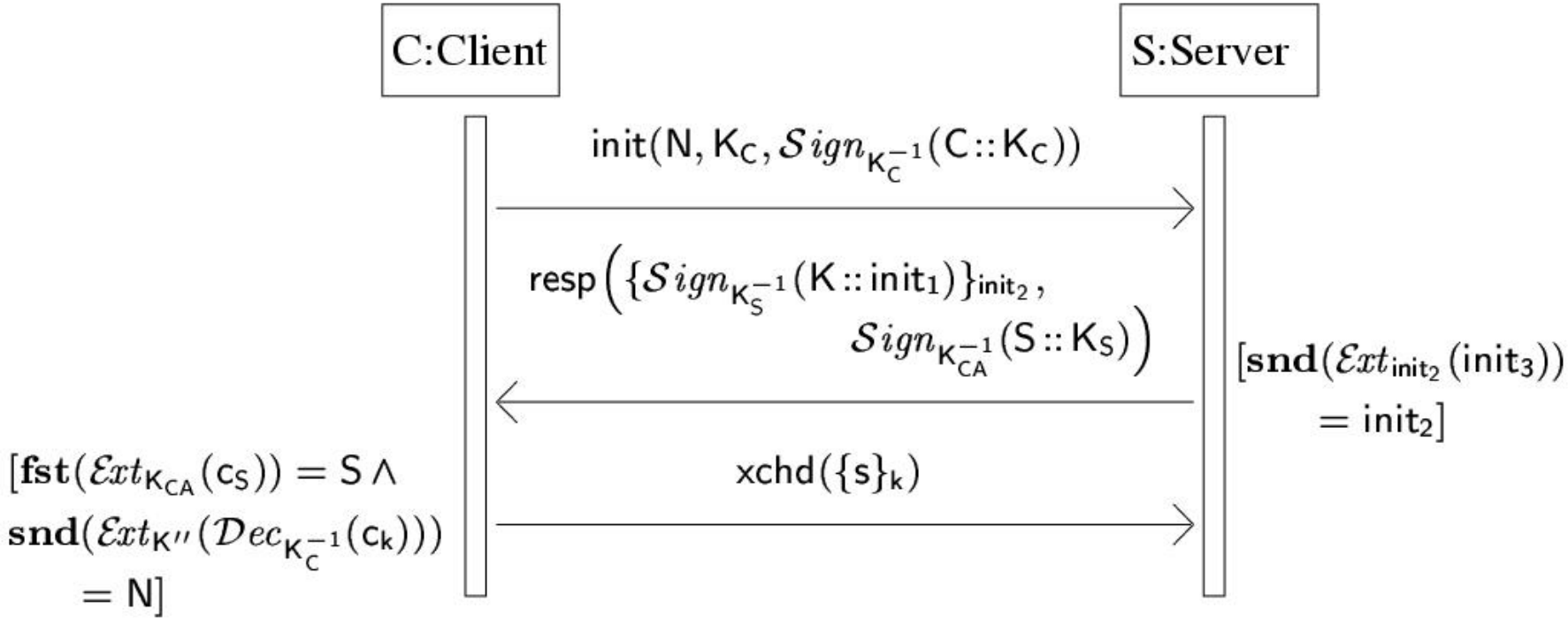
- $PRED(TR1) = \forall\, msg\_in.\ [knows(msg\_in) \wedge cond(msg\_in) \Rightarrow knows(msg\_out) \wedge PRED(TR2)]$

(Assume: order enforced (!).)

Can include senders, receivers in messages.

Abstraction: find all attacks, may have false positives.

Example



$knows(N) \wedge knows(K_C) \wedge knows(Sign_{K_C^{-1}}(C::K_C)) \wedge$

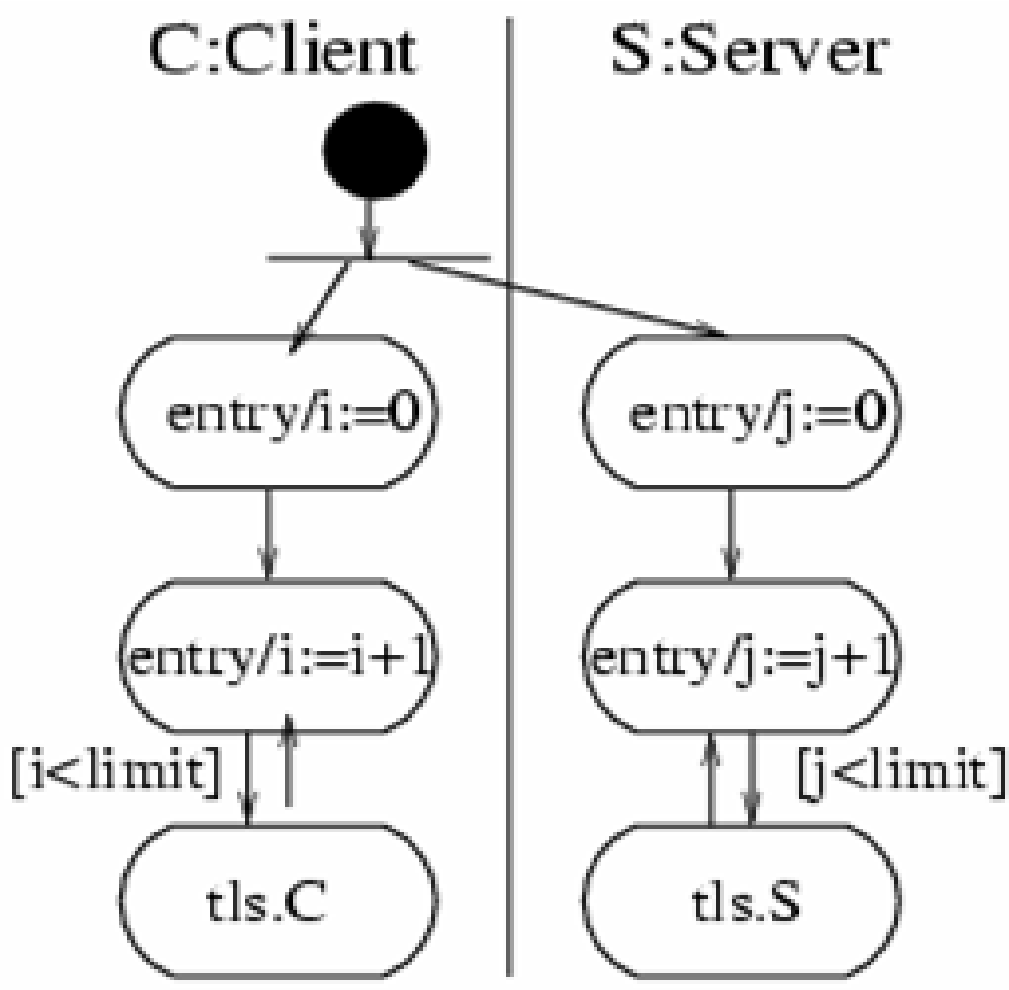
$\forall\ init_1, init_2, init_3 [knows(init_1) \wedge knows(init_2) \wedge knows(init_3) \wedge$

$snd(Ext_{init_2}(init_3)) = init_2 \Rightarrow knows(\{Sign_{K_S^{-1}}(\ldots)\}_{\ldots}) \wedge [knows(Sign\ldots)] \wedge$

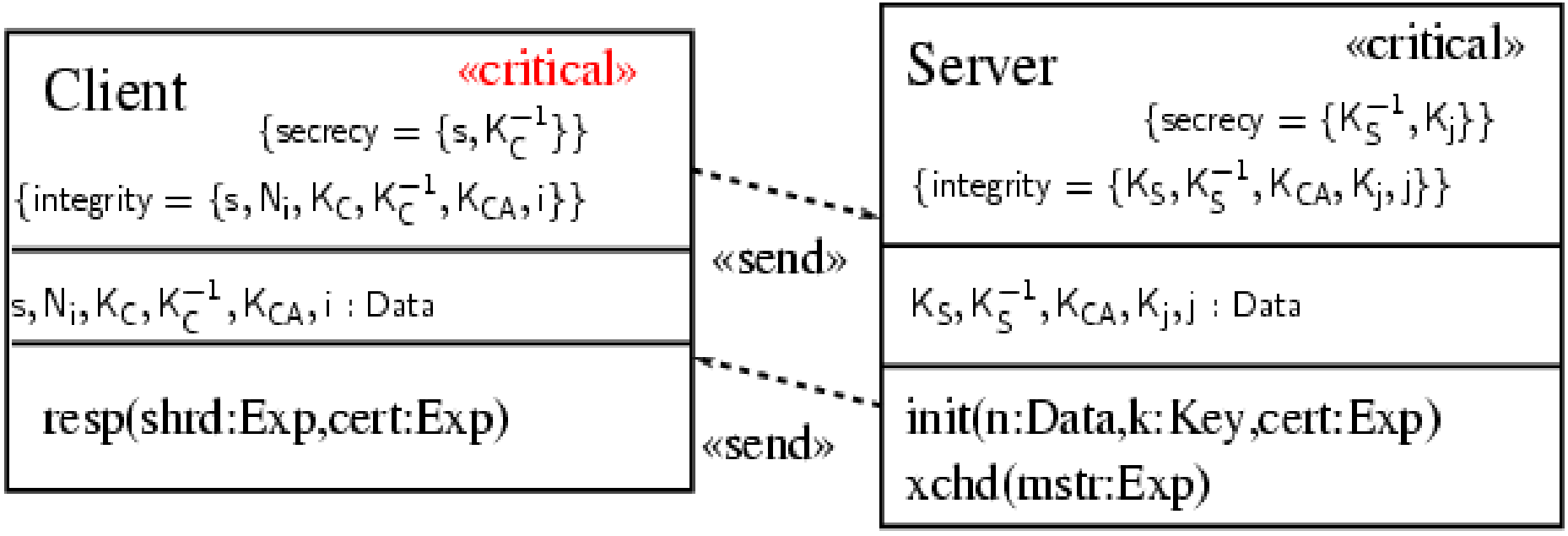
$. \forall\ resp_1, resp_2.\ [\ldots \Rightarrow \ldots]]$

Execute in System Context

Activity diagram.



Formulate Data Security Requirements



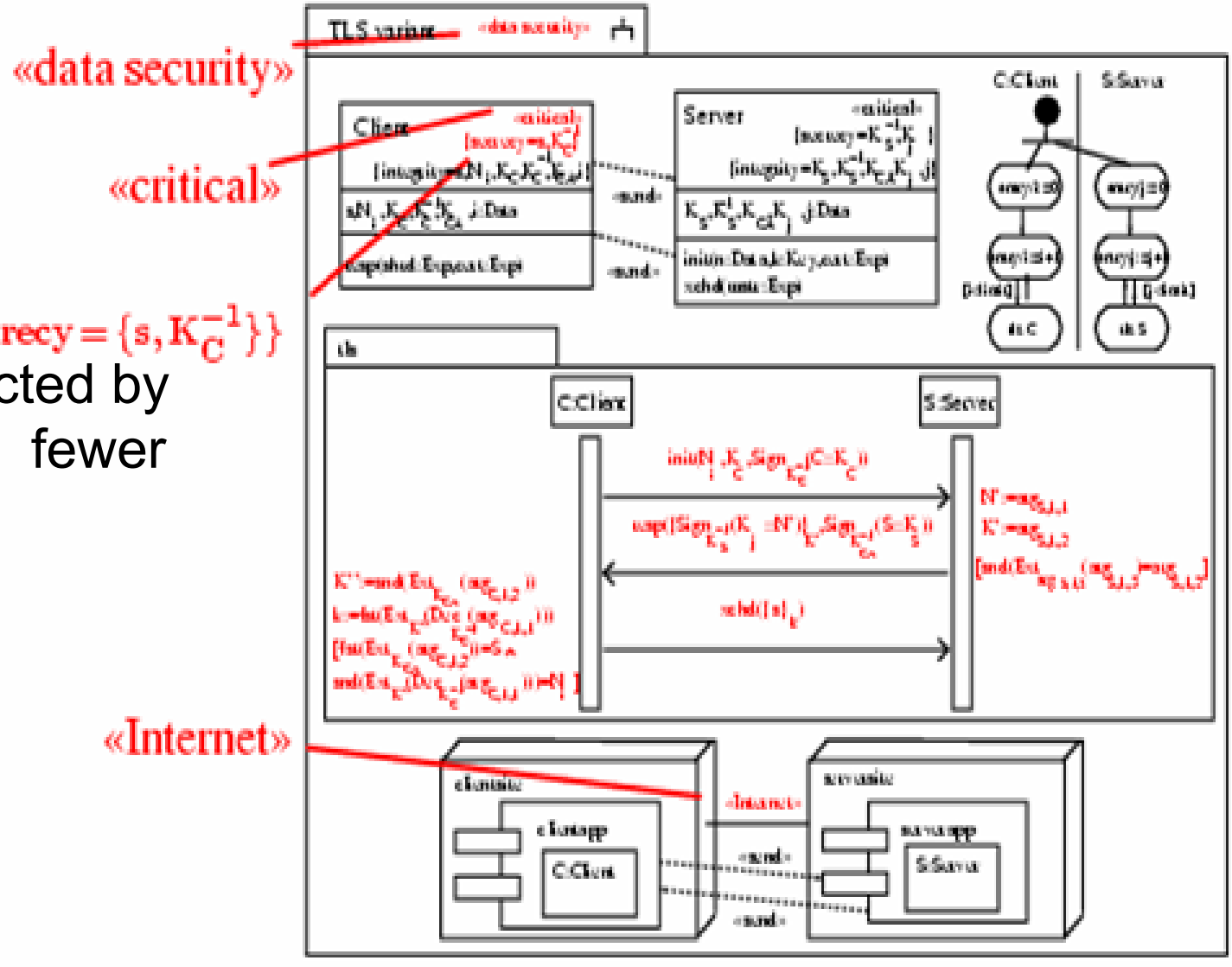
Class diagram.

Gives conjecture: *knows(s)* derivable ?

Proposed Variant of TLS (SSL)

IEEE Infocom 1999.
Goal:

- send secret protected by session key using fewer server resources.



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) ) ) )  )
```

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

Result

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

tlsvariant-freshkey-check.tptp
...
time limit information: 300 total (entering statistics module).
problem analysis ...
testing if first-order ...
first-order problem
...
statistics: 19 0 7 46 3 6 2 0 1 2 14 8 0 2 28 6
...
schedule selection: problem is horn with equality (class he).
schedule:605 3 300 597
...
entering next strategy 605 with resource 3 seconds.
...
analyzing results ...
proof found
time limit information: 298 total / 297 strategy (leaving wrapper).
...
e-SETHEO done. exiting
```

Surprise ...

Attack

… Which Means:

- Can derive *knows(s)* (!).
- Protocol does not preserve secrecy of *s* against adversaries.

  ➔ Completely insecure w.r.t. stated goals.
- But why ?
- Could look at proof tree.
- Or: use prolog-based attack generator.

Man-in-the-Middle Attack

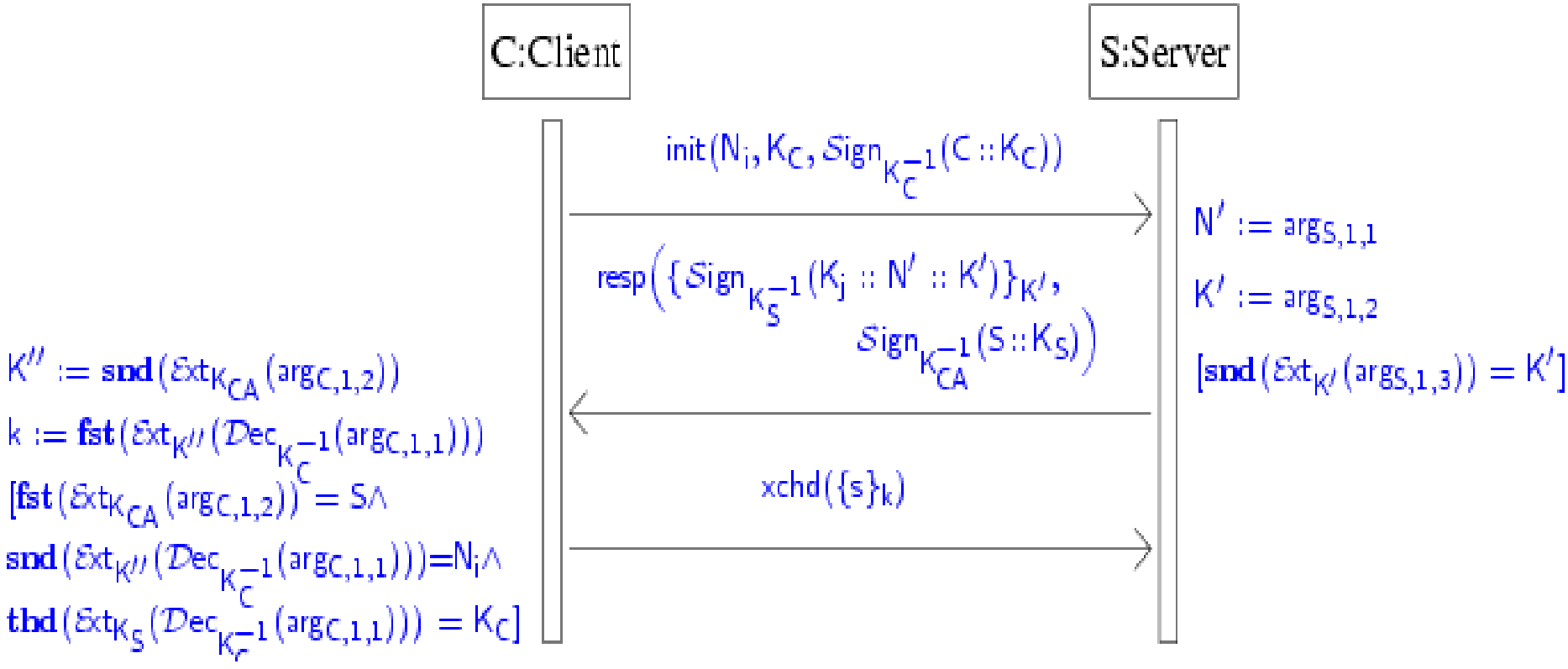
$$C \xrightarrow{N_i::K_C::Sign_{K_C^{-1}}(C::K_C)} A \xrightarrow{N_i::K_A::Sign_{K_A^{-1}}(C::K_A)} S$$

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

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

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

The Fix



- E-Setheo[1]: *knows(s)* not derivable
  ➔secure.

1 E-SETHEO is a strategy-parallel compositional theorem prover for first-order logic with equality.

Summary

- Example security analysis
  - Practical use of UMLsec
  - Formal proof
  - Apply fix for vulnerability

Zusammenfassung
Sicheres Software Design

- Geschäfts-Prozesse
- Modelbasierte Softwareenwicklung
- Sicheres Software Design
  - Sicherheitsanforderungen
  - UMLsec
  - UML-Analysis
  - Design Principles
  - Examples
    - TLS Variant
    - CEPS Purchase