*Vorlesung*

# *Methodische Grundlagen des Software-Engineering*

im Sommersemester 2014

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

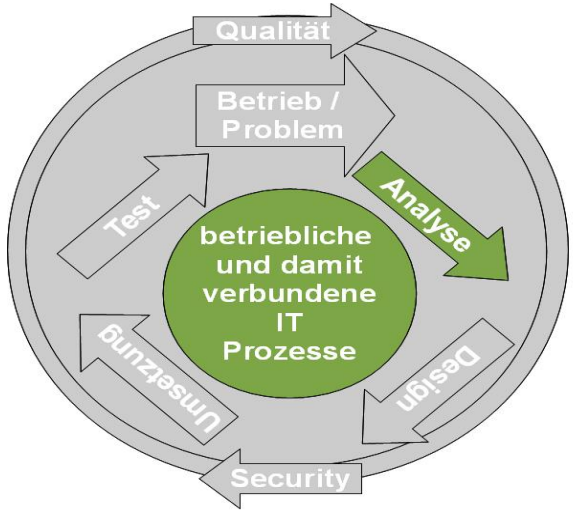
Teil 3.6: Kryptographische Protokolle: Sicherheitsanalyse

v. 14.07.2014

# Einordnung

## 3.6 Kryptographische Protokolle: Sicherheitsanalyse

- Geschäftsprozessmodellierung
- Process-Mining
- Modellbasierte Entwicklung sicherer Software
  - Einführung: Software Security
  - Hintergrund IT-Sicherheit
  - Wiederholung: Modellbasierte Software Entwicklung
  - Modellbasierte Sicherheit mit UML
  - Sichere Architekturen
  - Kryptographische Protokolle
  - Protokollanalyse
  - Biometrische Authentisierung
  - Biometrische Authentisierung: Analyse
  - Elektronische Geldbörsen
  - Clouds
  - Elektronische Signatur
  - Bankarchitektur




**Literatur:**
[Jür05] Jan Jürjens: **Secure systems development with UML**, Springer-Verlag 2005.
Unibibliothek (e-Book):
http://www.ub.tu-dortmund.de/katalog/titel/1361890
Papier-Version:
http://www.ub.tu-dortmund.de/katalog/titel/1091324
**Kap. 5**

- **Letzter Abschnitt:** Kryptographische Protokolle
  - ISO OSI Schichten-Modell
  - Kryptographische Ausdrücke
- **Dieser Abschnitt:** Sicherheitsanalyse von Protokollen
  - Angreifer-Wissen
  - Logik und grundlegende Regeln
  - Man-in-the-Middle Angriff

# UML Modelle analysieren

Sicherheitsanforderungen im UML Modell maschinell analysieren.

→ Analysefähiges Modell nötig:

- UMLsec Profile mit diesem verbunden.
- Sicherheitsrelevante Information von sicherheitsorientieren Stereotypen (z.B. Angreifertyp).

D.h., auf Grundlage des UML Modells Bedingungen formulieren.

→ Sicherheitsanforderungen erfüllen.

Folgende Folien: Eigenschaften eines solchen Modells definiert & erklärt.

→ Nötig um Bedingungen für UML Profil zu formalisieren.

# Angreifer-Wissen

- Gegeben: Menge $K_A^0$ als **initiales Wissen** eines Angreifers des Typs *A*.
- Sei $K_A^{n+1}$ Menge der Krypto-Terme gebildet aus Menge $K_A^n$
  - von Krypto-Termen & aus Ausdrücken,
  - nach *n+1*sten Iteration des Protokolls empfangen,
  - durch Anwendung der Krypto-Operationen.
- Definition (Dolev, Yao 1982):
  Datenwert *M* : Vertraulich gegen Angreifer des Typs *A*, wenn kein *n* mit $M \in K_A^n$ .

# Sicherheitsanalyse in Logik erster Stufe

- **Idee:** Menge der **Datenwerte**, die Angreifer kennenlernen kann, **von oben** approximieren.
- Logisches Prädikat *knows(E)*: Angreifer lernt Ausdruck *E* während Protokollausführung kennen.
- Für jedes Geheimnis *s* überprüfen: *knows(s)* aus logischen Formeln, die Systemverhalten repräsentieren, ableitbar.

# Sicherheitsanalyse: Grundlegende Regeln

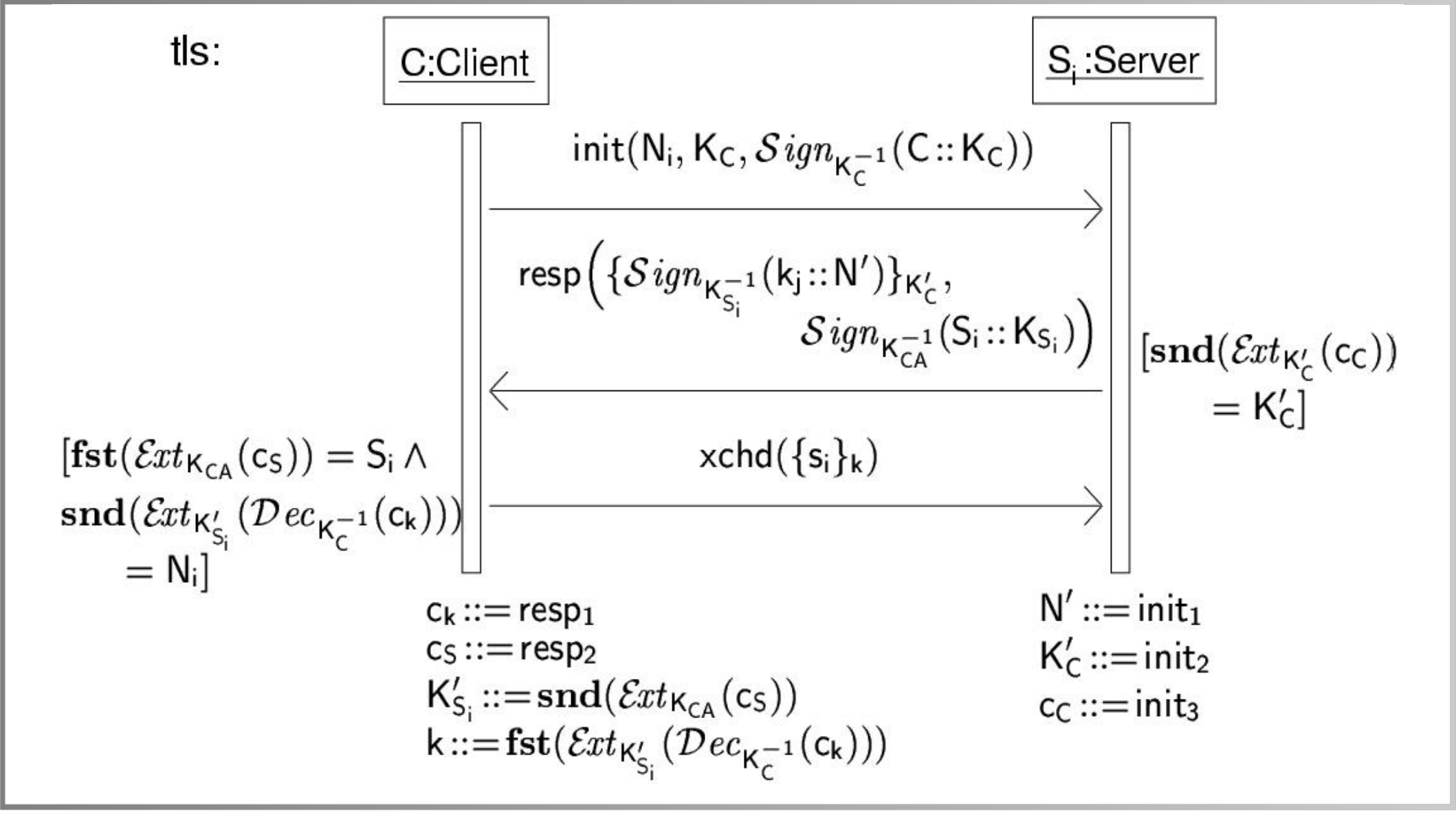
- Für **initiales Angreiferwissen** ($K^0$): Definiere Axiom *knows(E)* für jeden Datenwert $E$ (protokoll-spezifisch, z.B. $K_A$ , $K_A^{-1}$).
- Modellierung: Einsatz von **Krypto-Algorithmen** durch Angreifer selbst. → Definiere Axiome:

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

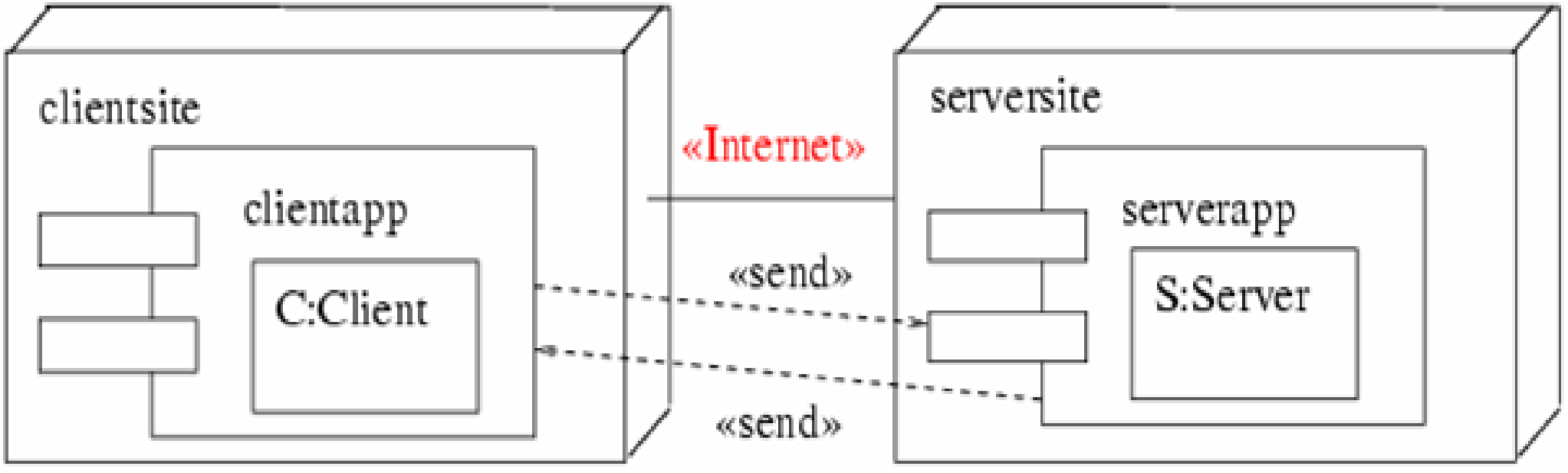
$$\forall E.(knows(E) \Rightarrow knows(head(E)) \wedge knows(tail(E)))$$

# Gegebenes Sequenzdiagramm für o.g. Variante von TLS …





# … und Verteilungsdiagramm …




Abgeleitete Angreifer-Aktionen auf Kommunikationsdaten:

**read, delete, insert.**

# … Übersetzung in Logik erster Stufe

- Gegeben: **Nachrichtenspezifikation**

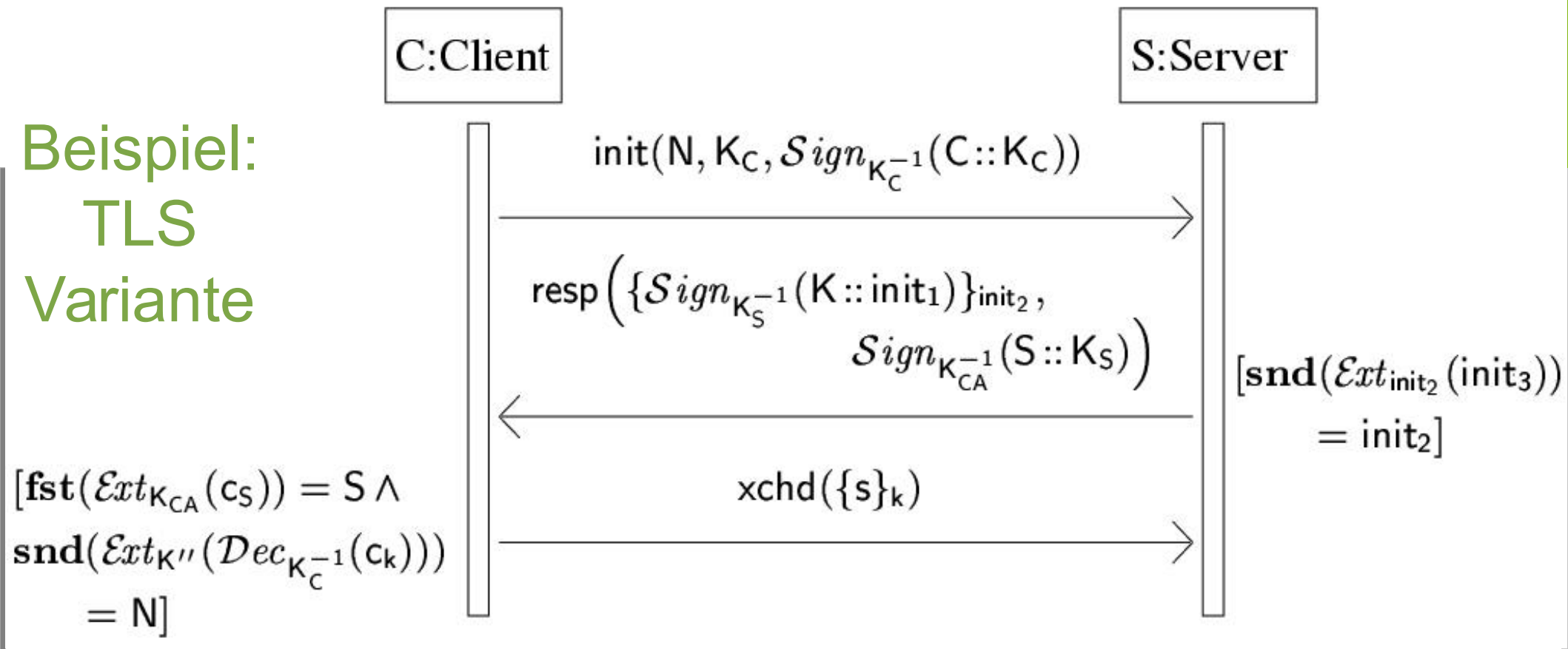
  $TR1=(in(msg\_in),cond(msg\_in),out(msg\_out))$

  im **Sequenzdiagramm**, gefolgt von Nachrichtenspezifikation *TR2*, ergibt Prädikat

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

- (Annahme: Nachrichtenreihenfolge überprüft (!).)
- **Hohe Abstraktion** (z.B. vom Sender & Empfänger der Nachrichten) mit Ziel: effiziente automatische Analyse.
- **Nachteil:** “falsch-positive“ Ergebnisse auftretbar (d.h. angebliche Angriffsmöglichkeiten: unpraktikabel).

# Beispiel: TLS Variante



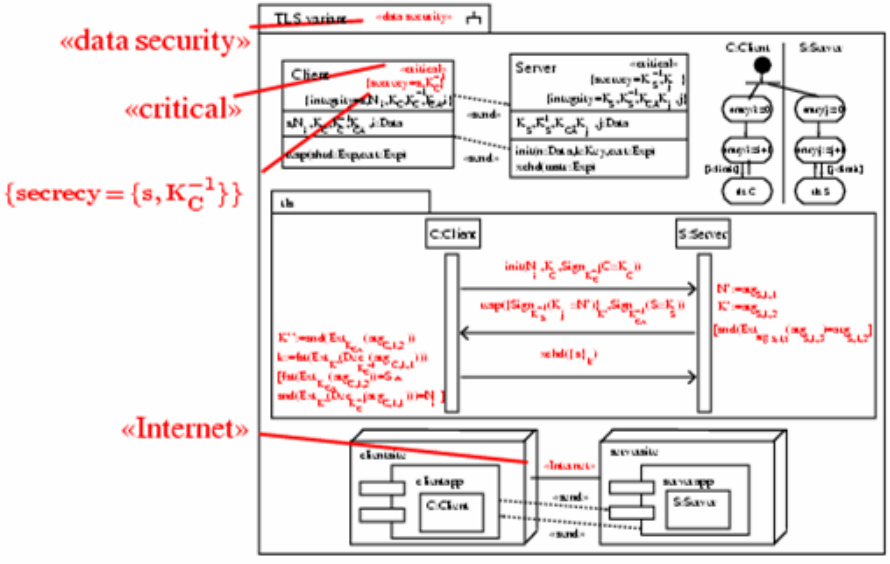

$knows(N) \wedge knows(K_C) \wedge knows(Sign_{K_C^{-1}}(C::K_C))$

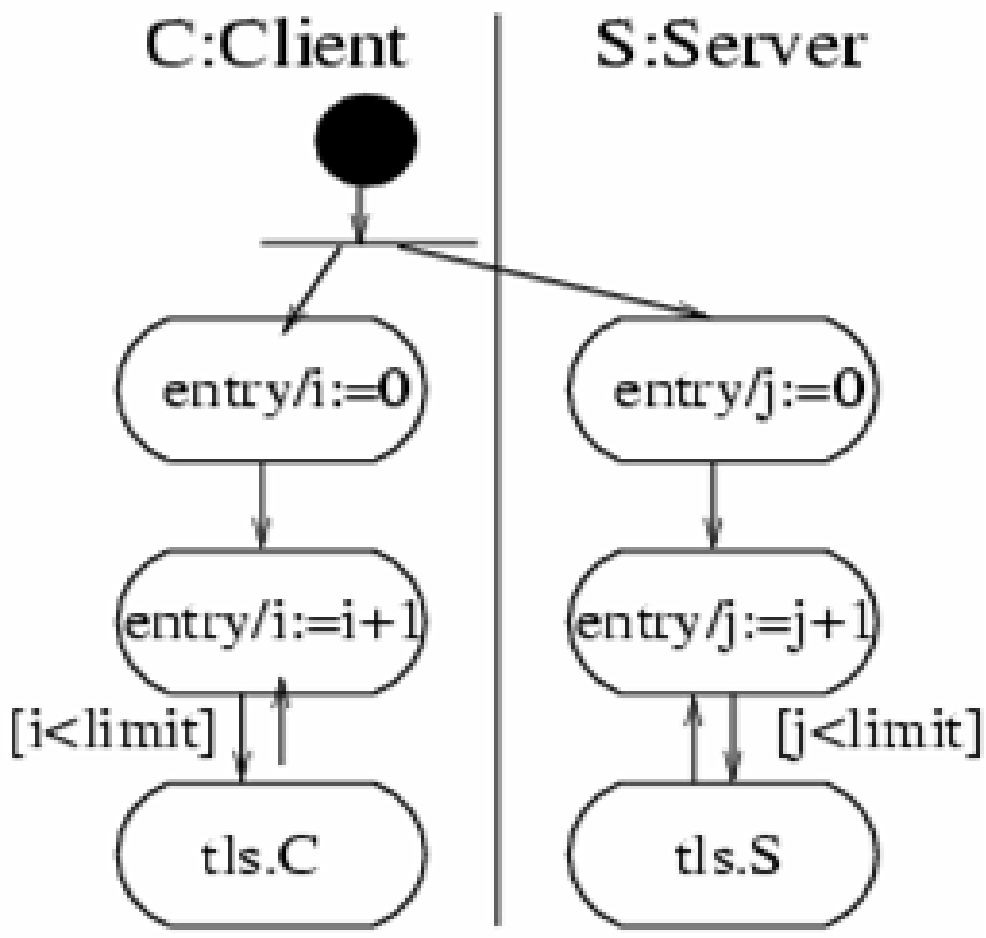
$\wedge \forall init_1, init_2, init_3.[knows(init_1) \wedge knows(init_2) \wedge knows(init_3) \wedge snd(Ext_{init_2}(init_3)) = init_2$

$\Rightarrow knows(\{Sign_{K_S^{-1}}(\ldots)\}_{\ldots}) \wedge [knows(Sign\ldots)] \wedge$

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

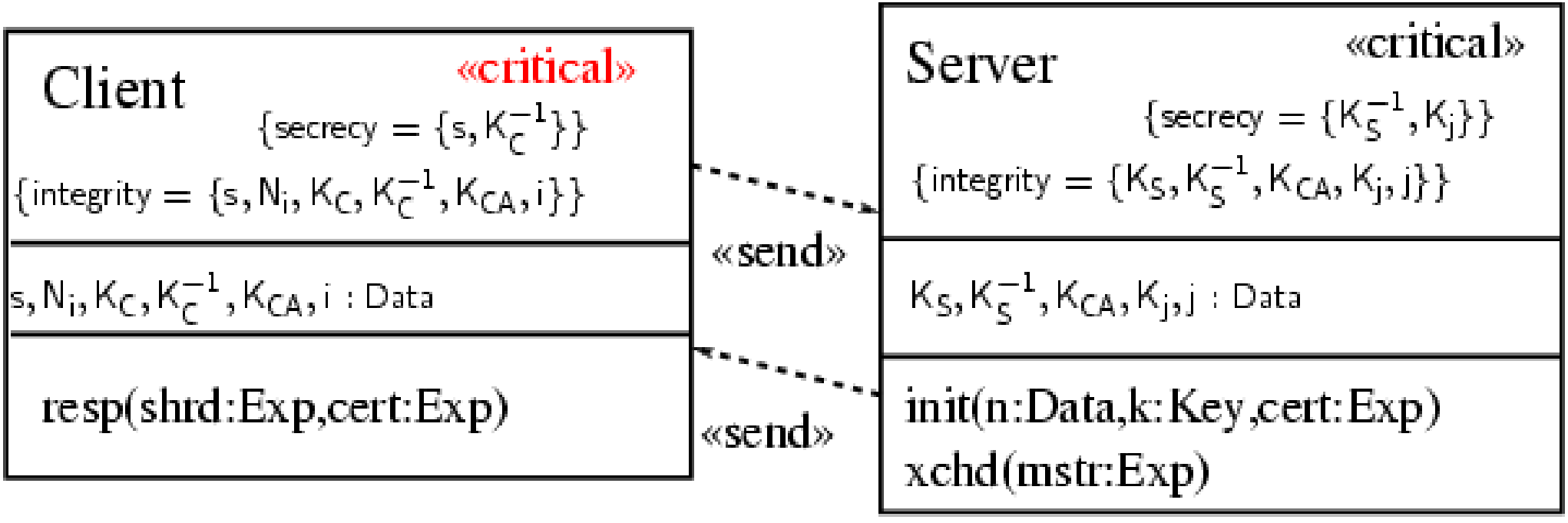
# Ausführung im Systemkontext




**Aktivitätsdiagramm**

# Sicherheitsanforderungen an Daten




**Klassendiagramm**

Frage: Vertraulichkeit von Geheimnis $s_i$ bewahrt, d.h. *knows($s_i$)* ableitbar ?

# Überblick Variante von TLS: Gesamtes Modell

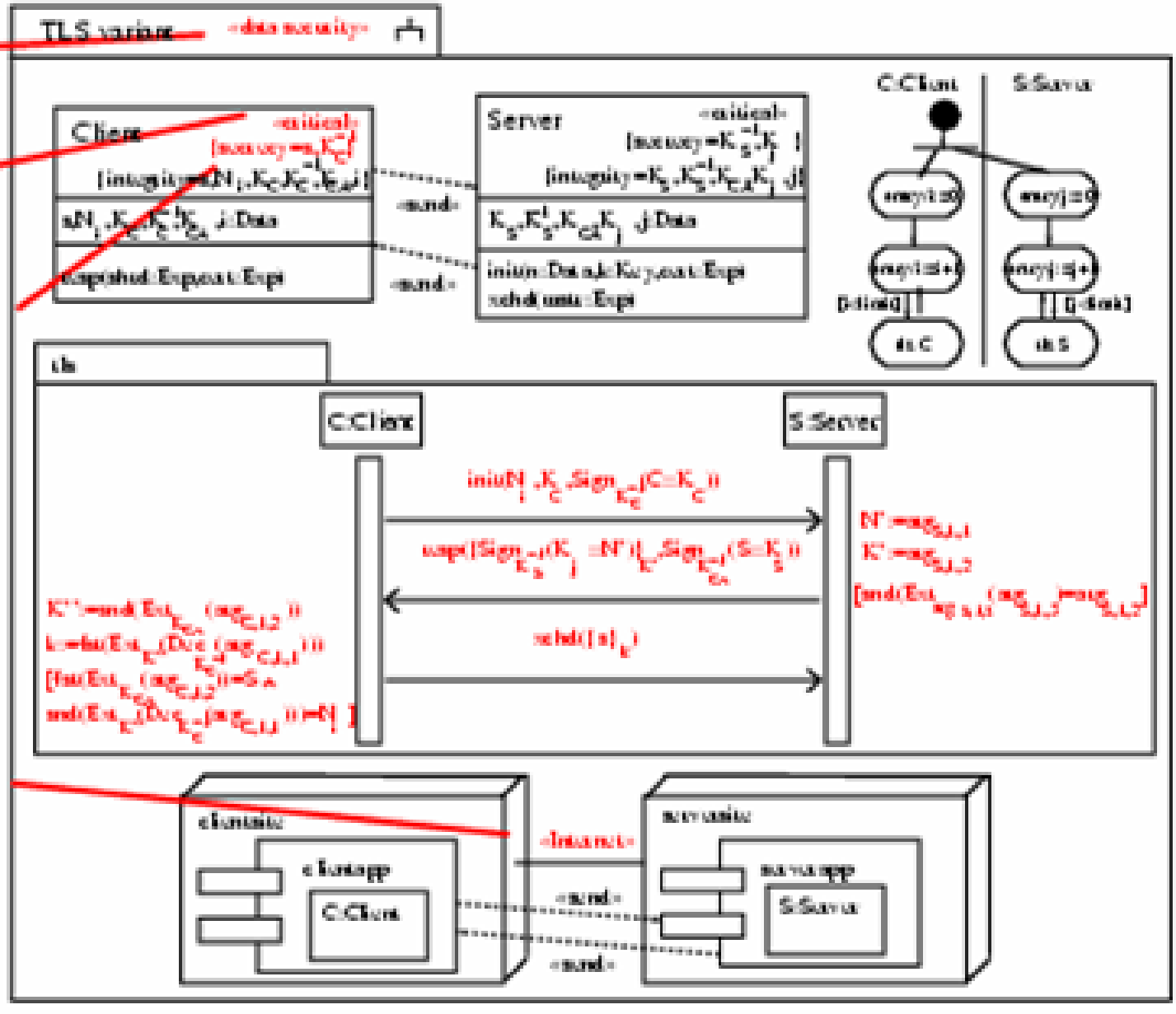


«data security»

«critical»

$\{secrecy = \{s, K_C^{-1}\}\}$

«Internet»



# Übersetzung in Logik in Theorembeweiser-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) ) ) )  )
```

# Übersetzung in Logik in Theorembeweiser-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)) )  )
) )  )).
```

# Überraschung ...



```
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
```

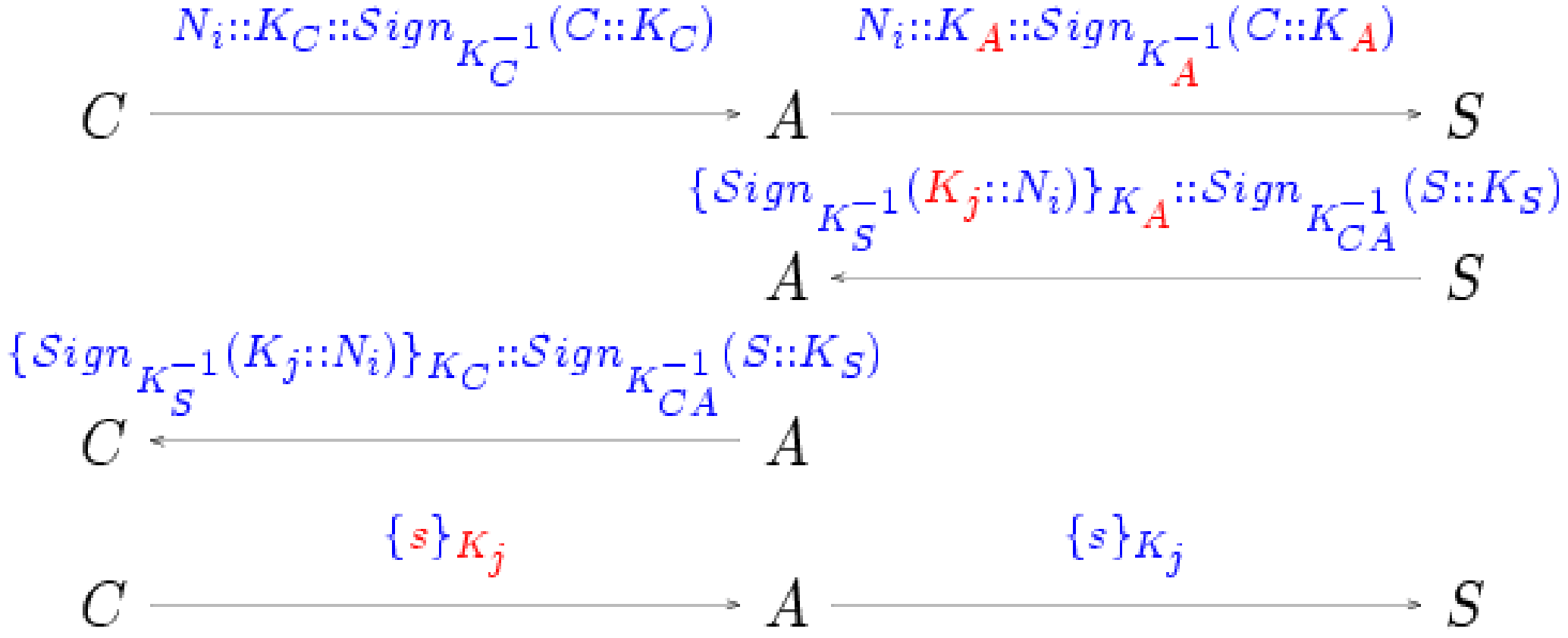
Attack

# … Was bedeutet das ?

- Kann *knows($s_i$)* ableiten (!).
- Daher: Protokoll bewahrt **nicht** Vertraulichkeit des Geheimnisses $s_i$ gegen Angreifer.

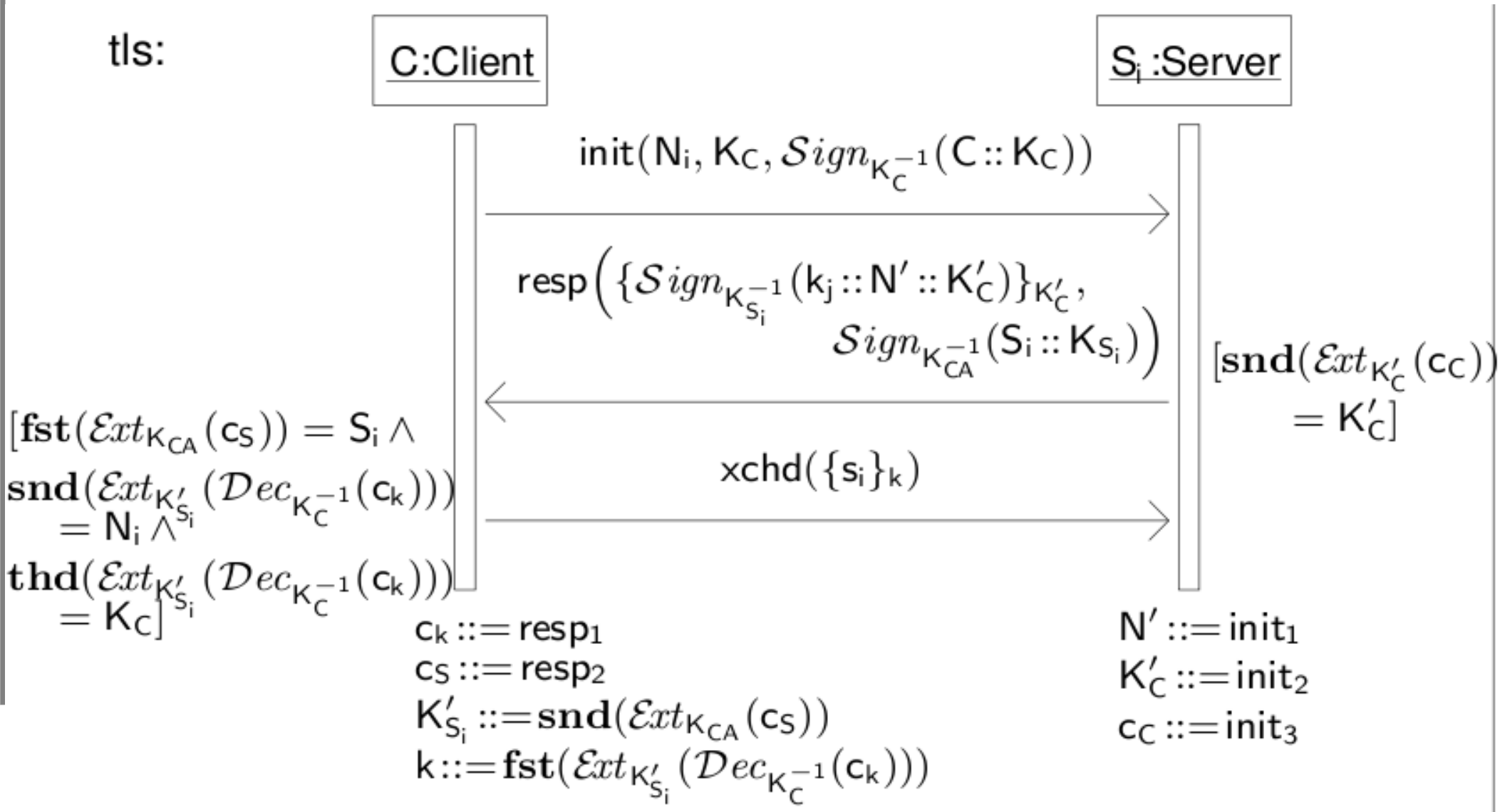
  → Unsicher in Bezug auf festgesetzte Ziele.
- Aber wieso ?
- Angriffszenario mittels prologbasierten Angriffsgenerator erzeugbar.

# Man-in-the-Middle Angriff



# Korrigiertes Protokoll

Analyse: *knows($s_i$)* nicht ableitbar, d.h. **Vertraulichkeit** von $s_i$ **bewahrt**.



# Zusammenfassung

- TLS-Variante
- Protokollmodellierung
- Sicherheitsanalyse
- Korrigierte Version

# Anhang (weitere Informationen zum selbständigen Nacharbeiten)

# Sicherheitsanalyse

Modulare UML Semantik: natürliches Modell potentieller Verhaltensweisen eines Angreifers erstellen.

- Verschiedene Angreifertypen modellieren. → Verschiedene Teile des Systems in bestimmter Weise angreifbar.
  - z.B. Angreifer vom Typ *insider* kann Kommunikationsverbindungen eines Firmen-LANs abhören.
- Angreiferverhalten modellieren: Klasse von UML Maschinen definieren, die auf Kommunikationsverbindungen auf bestimmte Art zugreifen.
- Sicherheit eines Systems bzgl. gegebenen Angreifertypen beurteilen: gemeinsame Ausführung eines Systems mit jeder UML Maschine in der Klasse betrachten.

→ Intuitive Formulierung von vielen Sicherheitseigenschaften.

# Angreifer

Modellklasse von Angreifern.

Je nach Bedrohungsszenario verschiedene Teile des System angreifbar.

Beispiel: Insider Angreifer können Kommunikationsverbindungen im LAN abhören.

Sicherheitsspezifikationen auswerten: Mit versch. Angreifermodellen simulieren.

# Konkrete Bedrohungen (1/2)

- Idee: $Threats_A(s)$ spezifiziert ein Bedrohungsszenario eines Angreifers des Typs A auf eine mit s stereotypisierte Komponente / Verbindung.
  - Festlegung der vom Angreifer erhaltbaren Daten bei Zugriff auf Komponenten.
  - Festlegung der vom Angreifer ausführbaren Aktionen, um auf betroffene Verbindung Zugriff zu erhalten.
- Von abstrakten Bedrohungen auf konkrete Bedrohungen schließen.
  → Relevant für Modellierungs- und Analysezwecke möglichen Angreifers.
- Definition der Menge $threats^S_A(x)$ konkreter Bedrohungen. → Analyse von UML Subsystem Spezifikation S gegenüber eines Angreifers vom Typ A.

# Konkrete Bedrohungen (2/2)

$threats^S_A(x)$: kleinste Menge genügt folgenden Bedingungen:

- *x:* Verbindung / Knoten im Deployment-Diagramm.
- Wenn jeder Knoten *n*, der *x* enthält*, ein Stereotype $s_n$ mit *access* $\in Threats_A(s_n)$ hat, dann:
  - Für jeden Stereotype *s* angefügt an *x*, ist

    $Threats_A(s) \subseteq threats^S_A(x)$
  - *x :* Verbindung verbunden mit einem Knoten, der Stereotype *t* mit *access* $\in Threats_A(t)$ hat.

    $\rightarrow \{delete,read,insert\} \subseteq threats^S_A(x)$

*(* Knoten und Subsysteme können untereinander verschachtelt sein)*

# Wichtige Sicherheitseigenschaften

- Nach Standardmethode von Dolev, Yao (1983) spezifiziert:
  - secrecy
  - integrity
  - authenticity
  - freshness
- Methode definiert Sicherheitsanforderungen unter Berücksichtigung des Angreifermodells auf intuitive Weise.
- Demnächst: Definition der Anforderungen eines sicheren Informationsflusses.

# Vertraulichkeit (1/4)

Definition von Vertraulichkeit (secrecy):

- Idee: System sendet nie Informationen aus denen Angreifer d re-konstruierbar. → Systemspezifikation stellt Vertraulichkeit (secrecy) eines Datenstücks d sicher.
- Angreifer eines gegeben Typs kennt d initial nicht, aber nach System-ausführung mit gewisser Eingabesequenz. → *d* nicht mehr geheim.
- Andernfalls d geheim.

# Vertraulichkeit (2/4)

Formalisierung von Vertraulichkeit (secrecy):

- Während Ausführung von $[[S]]_A$ E nicht in Wissensmenge K von A.

  → UML Subsystem S bewahrt Vertraulichkeit eines Ausdrucks E eines Angreifers vom Typ A.

- Für jeden Ausdruck E (an jedem Punkt Wert der Variable v) bewahrt S Geheimhaltung von E eines Angreifers vom Typ A .

  → *S* bewahrt Vertraulichkeit einer Variable v eines Angreifertyps A.

# Vertraulichkeit (3/4)

- *Beachte:* Durch Konstruktion des Angreiferwissens: Angreifer kann Ausdrücke aufschlüsseln. → Zugriff auf geheime Unterausdrücke.
- Definition zur Verifizierung: Angabe der oberen Schranke für Wissensmenge K ?

  → Möglich wenn sicherheitsrelevante Teile einer Systemspezifikation S als Sequenz von Kommandos folgender Form gegeben:

  - *await event e*
  - *check condition g*
  - *output event e'*

# Vertraulichkeit (4/4)

Beispiele:

- System bewahrt nicht Vertraulichkeit von *m* / *K* gegenüber das Internet abhörende Angreifer, falls:
  - Ausdruck $\{m\}_K :: K \in Exp$ über ungesicherte Internetverbindung gesendet.
  - nur $\{m\}_K$ und nichts anderes gesendet, umgekehrt.
- System S erhält mit öffentlichem Schlüssel von S verschlüsselten Schlüssel K über dedizierte Kommunikationsverbindung & sendet $\{m\}_K$ hierüber.
  - Bewahrt nicht Vertraulichkeit von m gegenüber Angreifern, die die Verbindung abhören und Nachrichten einschleusen.
  - Geheimhaltung bewahrt, falls Angreifer keine Nachrichten einschleusen.

# Integrität (1/2)

Definition von Integrität:

- Während Ausführung betrachteten Systems: Systemvariable bekommt anderen Wert als Soll-Wert.

  → Angreifer hat Wert dieser Variable verändert.

  → Integrität der Variable verletzt.

- Angreifer während Ausführung anwesend, aber Variable hat Soll-Wert.

  → System stellt Integrität der Variable sicher.

# Integrität (2/2)

Formalisierung von Integrität:

- Gegeben: Menge $E \subseteq Exp$ von akzeptierten Ausdrücken:
  - Während Ausführung von $[[S]]_A$ an jedem Punkt: Attribut a undefiniert oder vom Element von E ausgewertet.

    → Subsystem *S* stellt bzgl. E des Angreifertyps A mit initialem Wissen $K^0$ Integrität des Attributs a sicher.
  - Wenn $E = Exp \setminus K^0$, dann stellt S Integrität eines Attributs a bzgl. eines Angreifers vom Typ A mit initialem Wissen $K^0$ sicher.

# Anmerkung zu Vertraulichkeit & Integrität



Beachte:

- Formalisierung von Geheimhaltung (secrecy) bzw. Integrität (integrity) relativ *„grob/simpel“*.
- Impliziter Informationsfluss nicht verifiziert
- aber vergleichsweise leicht verifizierbar
- in Praxis ausreichend.

# Authentizität (1/3)

Definition von Authentizität:

- Während Systemausführung: Nachricht erscheint in diesem Teil zuerst.

  → Nachricht hat seinen Ursprung im Teil des Systems.

- Authentizität sicherstellen: Information am Ursprung der Nachricht schützten.

# Authentizität (2/3)

Formalisierung von Authentizität:

- Angenommen: a und o Attribute im Subsystem S, in dem o Ursprung der Nachricht speichern soll, welcher in a gespeichert ist.
- Während Ausführung von $[[S]]_A$: an jedem Punkt kommt Wert des Attributes a als erstes innerhalb der Ausführung in $outQu_o$ von allen Ausgabe- und Verbindungswarteschlangen in S als Unterausdruck vor.
  → S stellt bzgl. des Ursprungs o eines Angreifertyps A mit initialem Wissen $K^0$ (Nachrichten-)Authentizität sicher.

# Authentizität (3/3)

Verbindung zu Integrität:*

- Identität des Senders einer Nachricht Teil dieser Nachricht.
  → Möglichkeit Sender zu authentifizieren.
- Hier: Aus Datenintegrität folgt Nachrichtenauthentizität.

* A. Menezes, P. van Oorschot, and S. Vanstone. Handbook of Applied Cryptography, FL, 1996.
D. Gollmann. Facets of security. In C. Priami, editor, Global Computing. Programming Environments, Languages, Security, and Analysis of Systems, IST/FET International Workshop, (GC 2003)

# Freshness (1/2)

Frische (freshness) eines Wertes hat 2 Eigenschaften*:

- Unberechenbarkeit (unpredictability): Angreifer kann Wert nicht erraten.
- Frische (newness): Wert erschien nie zuvor während Ausführung des Systems.

Frische (freshness) bzw. Unberechenbarkeit (unpredictability) von *Daten* unter Berücksichtigung eines Angreifertyps *A* angegeben, der *Daten* nicht zur Menge seines vorherigen Wissens $K^p_A$ zählen kann.

*(* laut Gavin Lowe)*

# Freshness (2/2)

Definition von freshness (im Sinne von newness):

- Atomarer Wert *data ∈ (Data ∪ Keys)* im Subsystem *S* innerhalb einer Subsysteminstanz oder Objekts *D* ,
- welches in *S*, frisch (fresh),
- wenn Wert *data* in Spezifikation S nur in Teilen des Diagramms, welches D spezifiziert, vorkommt.

→ Bereich der Daten in S (scope of data in S).

# Notation (1/2)

Annahme: Übliche Definitionen Theorie und Logik aus grundlegender Menge, z.B. aus *"Handbook of logic in computer science"** , & folgende Definitionen:

- $\mathbb{N}$ : Menge von nicht-negativen Integers.
- $\mathbb{N}_n$ : Menge von nicht-negativen Integers bis zu & inkl. $n$, $\forall\ n \in \mathbb{N}$.
- $P(X)$ : Menge von Teilmengen der Folge $X$.

Gegeben: Folge (oder Liste) $l = (l_1, l_2, l_3, ...)$:

- *head(l)* für head element $l_1$
- *tail(l)* for tail $(l_2, l_3, ...)$
- *[ ]* für empty list (insbesondere für leeren String)

* S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors. Handbook of logic in computer science, volume 1-5, pages xii+827. The Clarendon Press, New York, 1992-2000.

# Notation (2/2)

- Multimenge (or bag): Menge mit mehreren Kopien von einem Element, mit Notation *{{ }}* anstelle *[ ]* .
  - e.g. *{{1,1,1,1,1}}:* Teilmenge bestehend aus 5 Kopien des Elements *1*.
- Für Multimenge *M* & Menge *X*:
  - *M* ↘ *X* filtert alle Elemente aus *M*, die Elemente von *X* .
- Für 2 Multimengen *M* und *N*:
  - *M* ∪ *N* ist ihre union.
  - *M* \ *N* ist subtraction der N von M.
  - *M* ⊆ *N* if *M* ↘ *N* = *M*
- Für Multimenge *M*
  - *⌊M⌋* : Menge der Elemente in *M*.
  - *#M* : Anzahl der Elemente in *M*.

# Überblick formaler Semantiken Nachrichten

In UML: Beide Objekte & Systemkomponenten durch exchanging messages aus gegebener Menge *Events* kommunizierbar.

- Eintreffen solcher Nachricht: *event*
- Bestehen aus:
  - Nachrichtenname aus gegebener Menge *MsgNm*.

    (Nachrichtennamen mit Objekt oder Instanznamen aus gegebener Menge *UMLNames* vorsetzbar)
  - Argumente an Nachricht, die Elemente gegebener Menge von *Exp* (see this slide).

$$msg = messagename(exp_1, \ldots, exp_n)$$

# Überblick formaler Semantiken Input/Output Queues

Jedes Objekt / Komponent *O* kann

- Nachrichten in Multimenge $inQu_O$ (input queue genannt) erhalten.
- Teilt Nachrichten an Multimenge $outQu_O$ (output queue genannt) mit.

→ Eher Multimengen als Mengen, weil mehrere Kopien gleicher Nachricht gleichzeitig empfangbar.

# Überblick formaler Semantiken Nachrichtenverteilung

Senden einer Nachricht *msg* von einem Objekt / Teilsysteminstanz *S* zu einem Objekt / Teilsysteminstanz *R* :

- *S* ordnet Nachricht *R.msg* zu ihre Multimenge $outQu_S$ zu.
- Scheduler verteilt Nachrichten aus output queues in vorgesehene input queues.
  - Nachrichtenkopf wird entfernt
  - *R.msg* von $outQu_S$ entfernt und *msg* in $inQu_R$ hinzugefügt.
- *R* entfernt *msg* von ihrer input queue & verarbeitet dessen Inhalt.

# Überblick formaler Semantiken
# Weitere Details

- Operationsaufruf: Sender nachverfolgen um Senden von Rücksignal zu erlauben.
- Diese Kommunikationsmodellierung ermöglicht flexible Bearbeitung.
  - z.B. Verhalten des Schedulers modifizierbar → Kenntnisse zugrundeliegender Kommunikationsschicht entnehmen.
- Prüfen von Sicherheitsfragen / anderen Aspekten, wie Ordnung / Verzögerung von Nachrichten.

# Überblick formaler Semantiken Einzelobjekte



- Auf Ebene der Einzelobjekte: Verhalten mit Statecharts / sequence diagrams modellieren.
- Interne Aktivitäten, enthalten als Zustände dieser Statecharts, z.B. als Statecharts / Sequenzdiagramme definierbar.
- Verwendung von Teilsystemen: Verhalten eines Systemkomponenten *C* durch einschließen eines Aktivitätsdiagramms definierbar.

  → Koordiniert jeweilige Aktivitäten versch. Komponenten & Objekten.

# Überblick formaler Semantiken UML machine (1/3)

- Für jedes Objekt *C* gegebenen Systems definieren Semantiken UML machine *[[C]]*, mit
  - Ist State machine.
  - kommuniziert mit Umgebung durch Nachrichten.
- Verhaltenssemantiken *[[D]]* eines Statechart-diagrammes *D* modellieren run-to-completion Semantiken.
- Jedes Sequenzdiagramm *S* gibt Verhalten *[[S.C]]* einzelner enthaltenen Komponenten *C*.
- Teilsysteme gruppieren Diagramme zusammen, die verschiedene Teile des System beschreiben.

  → Komponentesystem *C,* gegeben durch Teilsystem *S,* enthält Teilkomponenten $C_1,...,C_n$.
- Kommunikation: Über Kommunikationsverbindungen im geeigneten Verteilungsdiagramm.

# Überblick formaler Semantiken
# UML machine (2/3)

Verhaltensinterpretation *[[S]]* einer UML Teilsystemspezifikation *S*:

1. Multimenge von input events *(incoming messages)* benötigt.
2. Ereignisse durch Eingabe Multimengen & link queues verteilt.
   → Verbinden Teilkomponenten & als Argumente für Funktion gegeben
   → Definiert Verhalten beabsichtigter Empfänger in *S*.
3. Ausgabenachrichten zu link queues der Links verteilt.
   → Verbindet Sender einer Nachricht zum Empfänger oder
   → Als Ausgabe von *[[S]]* gegeben, falls Empfänger kein Teil von *S*.

*[[S]]* : UML machine.

# Überblick formaler Semantiken UML machine (3/3)



- Ausführung eines UML Teilsystems *S*: Folge von States & Multi-mengen der Ein- & Ausgabenachrichten von *[[S]]*.
- UML Spezifikation: nicht-deterministisch, z.B. weil mehrere Transitionen im Statechartdiagramm gleichzeitig ausführbar.
- Teilsystem *T* : *Black-box Verbesserung* von *S* , falls jedes Eingabe / Ausgabe Verhalten von *T* = Eingabe / Ausgabe Verhalten von *S*.
- *T* : *verspätete Black-box Verbesserung* von *S*, falls jedes Eingabe / Ausgabe Verhalten von *T* sich von Eingabe/Ausgabe Verhalten von *S* unterscheidet. → Nur in dieser Verzögerung einführbar.

# Adversary machine (1/5)

- Tatsächliches Verhalten eines Gegnertyps *A* als "type A adversary machine" modellieren.
- UML machine* mit folgenden Daten: (* see this slide)
  - Menge von States *State* mit control state *control ∈ State*.
  - Menge von aktuellem Gegenerwissen $K_A \subseteq Exp$.
  - Für jede mögliche control state *c ∈ State* & Menge von Wissen $K \subseteq Exp$:
    - Menge $Delete_{c,K}$ enthält Namen von jedem Link *l* mit $delete \in threats^S_A(l)$.
    - Menge $Insert_{c,K}$ enthält jedes Paar *(l,E)*, wo *l* Name eines Links mit $insert \in threats_A^S(l)$, und $E \in K$.
    - Menge $newState_{c,k} \subseteq State$ von States.

# Adversary machine (2/5)

Machine: iterativ durchführen.

- Vom spezifizierten Startzustand $control := control^0$.
- Mit aktuellem Gegnerwissen $K := K^0_A$.

Jede Iteration verläuft mit folgenden Schritten:

- Aufnahme der Inhalte aller link queues in $K$, die zum link $l$ mit $read \in threats^S_A(l)$ gehören .
- Abbildung des Inhalts von jedem link queue zu $\varnothing$, der zum link $l \in Delete_{control,K}$ gehört.
- Inhalt von jedem link queue, der zum link $l$ gehört, mit allen Ausdrücken $E$ erweitert, wo $(l,E) \in Insert_{control,K}$ gilt.
- Nächster control state: Nicht deterministisch von Menge $newState_{control,K}$ wählen.

# Adversary machine (3/5)

- Menge $K^0_A$ von Startwissen: Algebra der Ausdrücke durch Mengen $K^a_A$ und $K^p_A$ erzeugt.
- $K^a_A$: Menge von verfügbaren Wissen:
  - Datenwerte $v$ in UML Spezifikation unter Berücksichtigung gegeben.
  - jeder Knoten $n$ enthält $v$, der $s_n$ mit $access \in Threats_A(s_n)$ trägt.
- $K^p_A$: Menge von Vorkenntnissen:
  - Angreifer Zugriff auf weitere Daten geben.
  - Vor Ausführungsbeginn des Systems bekannt, wie öffentlicher Schlüssel.

# Adversary machine (4/5)

- Angreifer *A* kann alle über link *l* gesendete Werte löschen, repräsentiert durch $delete_l \in threats^S_A(l)$.
- *A* kann nicht selektiv Wert *e* mit bekannter Bedeutung von *l* löschen.
- Beispiel:
  - Über Internet im virtuellen privaten Netzwerk gesendete Nachrichten: verschlüsselt.
  - Können alle Nachrichten willkürlich löschen.
  - Können deren Bedeutung dem Angreifer bekannte Nachrichten nicht löschen.

# Adversary machine (5/5)

Bzgl. Folie UML machine (3/3):

- Teilsystem *T* : Blackbox Verfeinerung in Gegenwart eines Gegnertyps *A* eines Teilsystems *S*, falls

  jedes beobachtbares Ein-/Ausgabe Verhalten einer Ausführung von *T* in Gegenwart eines Gegnertyps *A*

  = Ein-/Ausgabe Verhalten einer Ausführung von *S* in Gegenwart eines Gegneryps *A*.

- *T* : verzögerte Blackbox Verfeinerung in Gegenwart eines Gegnertyps *A* eines Teilsystems *S*, außer dass Verzögerung in *T* einführbar.

# Verhaltensinterpretation: Security Evaluation

Definition der “execution of subsystem S in presence of an adversary of type A” als UML Machine $[[S]]_A$ durch Erweiterung der Definition von *[[S]]* auf Folie UML machine (2/3).

→ Sicherheit des Systems in Bezug auf jeweilige Art des Gegners beurteilen.

1. Multimenge empfangener input events *(eingehende Nachrichten).*
2. Ereignisse auf Teilkomponenten verteilen.
3. Ausgabenachrichten von Teilkomponenten verteilt.
4. Allgemeinster Typ A adversary machine auf link queues anwenden.