Modellierung kryptographischer Protokolle durch Induktion

Lukas Bulwahn

22. Juli 2006

In diesem Vortrag wird gezeigt, wie man kryptographische Protokolle durch induktive Definitionen der Protokolle modelliert und mechanisch verifizieren kann. Es wird dies anhand des Otway-Rees Protokolls gezeigt. Das Verfahren kann jedoch ohne große Änderungen auch auf weitere Protokolle, wie das Needham-Schroeder oder das Kerberos Protokoll, übertragen werden. Das hierbei beschriebene Verfahren wurde von Lawrence C. Paulson[1][2] entwickelt und in dem Theorembeweiser Isabelle[1] umgesetzt.

1 Einführung

Kryptographische Protokolle werden dazu verwendet, um geheime Nachrichten über ein unsicheres Netzwerk, wie zum Beispiel das Internet, auszutauschen.
Dabei sind zwei Eigenschaften offensichtlich: die Geheimhaltung und die Authentizität (Echtheit) des anderen Teilnehmers.
Das Protokoll soll sicherstellen, dass ein Dritter nicht die geheime Kommunikation „mithören“ kann und wenn Bob glaubt sich mit Alice zu unterhalten, so soll das Protokoll sicherstellen, dass dies auch Alice ist und vice versa.
Um nun Protokolle zu analysieren, sind die zwei folgenden Verfahren sehr beliebt:

- State exploration (Überprüfen des Zustandsraums): Das Protokoll wird als System von endlichen Zuständen modelliert und mit Hilfe eines Model Checkers überprüft. Fehler im Protokoll können dadurch einfach und sehr automatisiert gefunden werden. Das Nicht-Auffinden von Fehlern bedeutet aber nicht die Korrektheit des Protokolls. Desweiteren muss das Protokoll sehr stark vereinfacht werden, damit dies überhaupt von einem Model Checker überprüft werden kann.

[1] http://isabelle.in.tum.de

- Belief logics: Anhand von gegebenen Hypothesen gibt es eine Entscheidungsprozedur, ob eine Aussage in der Logik aus den Hypothesen folgt oder nicht. Falls nicht, kann dadurch ein Fehler aufgedeckt werden.

Das hier beschriebene Verfahren verwendet Elemente aus beiden Verfahren.
Es wird nun im Weiteren das formale Modell für Nachrichten, Ereignisse und Ereignisfolgen beschrieben.

2 Beschreibung des formalen Systems

Die folgenden Definitionen werden in Isabelle-ähnlicher Notation gegeben. Weitere Information zur Syntax findet man bei [3].

2.1 Agenten und Nachrichten

Es gibt drei Arten von Agenten: den Server, die freundlichgesinnten Agenten, und den Angreifer.
Agenten werden wie folgt formalisiert:
`datatype agent = Server | Friend nat | Spy`
Es folgt bereits aus der Definition, dass alle Agenten unterscheidbar sind, das heißt z.B. der Server ist nicht der Angreifer, der Friend i $\neq$ Friend j für i $\neq$ j.
Nachrichten werden entsprechend definiert:

```
datatype
msg = Agent agent
| Nonce nat
| Key key
| {| msg, msg |}
| msg_key
```

Dabei ist diese Definition aus folgender Überlegung entstanden. Die Nachrichten bestehen aus Agentennamen (wie z.B. der Absender) und Nonces, `Nonce nat`, die nicht erraten werden können.
Diese Nachrichtenteile können beliebig zusammengesetzt werden, dies wird durch `{| msg, msg |}` ausgedrückt. Es ist klar, das durch Verschachtelung von `{| msg, msg |}` beliebige zusammengesetzte Nachrichten erzeugt werden. `Key key` steht für einen beliebigen Schlüssel. $\texttt{msg}_{key}$ drückt aus, dass eine Nachricht `msg` mit dem Schlüssel `key` verschlüsselt worden ist.
Damit kann man alle notwendigen Nachrichten aufbauen.

2.2 Operatoren auf Nachrichten

Um nun Aussagen über Nachrichten und Protokolle machen zu können, werden drei Operatoren benötigt. Die Operatoren sind induktiv über Mengen von Nachrichten definiert.

- Der erste Operator ist der parts Operator. Der parts Operator repräsentiert die Menge aller Komponenten (parts) einer Nachricht, die aus einer Menge von Nachrichten möglicherweise wiederhergestellt werden können. Betrachtet man nun die Definition von Nachrichten, so ist klar die Nachricht selbst ist natürlich eine Komponente. Damit sind bereits die möglichen Fälle Agent und Nonce klar. Bei einem Paar von Nachrichten ist der erste und zweite Teil in der Menge der Komponenten.
  Bei verschlüsselten Nachrichten der Form $\mathrm{H} = \{\ \mathrm{X}_K\}$ ist $\mathrm{X} \in \mathrm{parts\ H}$, da X durch Entschlüsseln mit dem korrekten Schlüssel wiederhergestellt werden kann. Jedoch ist der Schlüssel $\mathrm{K} \notin \mathrm{parts\ H}$, da dieser nicht in der Nachricht mitgesendet wird, sondern lediglich beim Erstellen der Nachricht verwendet wird.
  Diese Überlegungen finden sich in folgender Definition wieder:
  parts : msg set $\rightarrow$ msg set

$$\frac{\mathrm{X} \in \mathrm{H}}{\mathrm{X} \in \mathrm{parts\ H}}$$

$$\frac{\{|\mathrm{X},\mathrm{Y}|\} \in \mathrm{parts\ H}}{\mathrm{X} \in \mathrm{parts\ H}} \qquad \frac{\{|\mathrm{X},\mathrm{Y}|\} \in \mathrm{parts\ H}}{\mathrm{Y} \in \mathrm{parts\ H}}$$

$$\frac{\mathrm{X}_K \in \mathrm{parts\ H}}{\mathrm{X} \in \mathrm{parts\ H}}$$

- Der zweite Operator ist der analz Operator. Dieser definiert alle möglichen Komponenten von Nachrichten, die wirklich wiederhergestellt werden können, d.h. für verschlüsselte Nachrichten muss der Decrypt-Schlüssel sich in der Menge der bereits analysierten Nachrichten befinden. Der analz Operator ist also dem parts Operator sehr ähnlich, dies spiegelt sich auch in der Definition von analz wieder.
  analz : msg set $\rightarrow$ msg set

$$\frac{\mathrm{X} \in \mathrm{H}}{\mathrm{X} \in \mathrm{analz\ H}}$$

$$\frac{\{|\mathrm{X},\mathrm{Y}|\} \in \mathrm{analz\ H}}{\mathrm{X} \in \mathrm{analz\ H}} \qquad \frac{\{|\mathrm{X},\mathrm{Y}|\} \in \mathrm{analz\ H}}{\mathrm{Y} \in \mathrm{analz\ H}}$$

$$\frac{\begin{array}{c}\mathrm{X}_K \in \mathrm{analz\ H} \\ \mathrm{Key(invKey\ K)} \in \mathrm{analz\ H}\end{array}}{\mathrm{X} \in \mathrm{analz\ H}}$$

- Der dritte notwendige Operator ist der synth Operator. Er definiert, welche Nachrichten aus einer Menge von Nachrichten erzeugt werden können.

Die Agenten sind überall bekannt, daher können diese Nachrichten von selbst erzeugt werden. Schlüssel und unerratbare Nonces können nicht erzeugt werden. Nachrichten können verschlüsselt werden, wenn der entsprechende Schlüssel bekannt ist.
synth : msg set → msg set

$$\frac{X \in H}{X \in \text{synth } H} \qquad \frac{}{\text{Agent agt} \in \text{synth } H}$$

$$\frac{\begin{array}{c} X \in \text{synth } H \\ Y \in \text{synth } H \end{array}}{\{|X,Y|\} \in \text{synth } H} \qquad \frac{\begin{array}{c} X \in \text{synth } H \\ \text{Key (K)} \in H \end{array}}{X_K \in \text{synth } H}$$

2.3 Events und Traces

In einer Kommunikation treten im wesentlichen die folgenden drei Ereignisse auf. Zum einen, das Versenden von Nachrichten, dies wird mit 'Says A B X' formalisiert, welches bedeutet, dass der Agent A dem Agent B die Nachricht X sendet.
Zum anderen, das Empfangen einer Nachricht, dies wird mit 'Gets A X' formalisiert. Dies bedeutet, dass der Agent A die Nachricht X empfangen hat. Der Agent A kann also beim Empfangen nicht die Herkunft bzw. den Absender der Nachricht feststellen.
Desweiteren kann sich jeder Agent bestimmte Nachrichten notieren. Dies wird mit 'Notes A X' bezeichnet, und bedeutet A merkt sich X. Tatsächlich wird dies im betrachteten Protokoll nur zur weiteren Unterscheidung von beabsichtigten und unbeabsichtigten Nachrichten verwendet. Beabsichtigte Nachrichten im Rahmen des Protokolls werden mit Says erzeugt und Notes wird verwendet wenn der Spion unbeabsichtigt eine Nachricht erhält.
Ereignisse (Events) haben dann folgende Form:

```
datatype
event = Says agent agent msg
| Gets agent msg
| Notes agent msg
```

Traces (Ereignisfolgen) sind nun einfach Listen von Ereignissen.

2.4 Wissen des Spions

Das Wissen, welches ein Agent besitzt, wird nun als Menge von Nachrichten repräsentiert. Zu Beginn besitzt der Server alle sharedKeys der Agenten, also auch seinen eigenen als auch den des Angreifers.
Die Agenten besitzen nur ihren eigenen sharedKey.
Der Angreifer besitzt neben seinem eigenen sharedKey auch alle sharedKeys

von Agenten, die er infiltriert hat - dies wird durch die Menge bad repräsentiert. Formal wird dies folgendermaßen ausgedrückt:

```
initState : agent → msg set
initState (Friend i) = {Key (shrK(Friend i))}
initState Server = {Key (shrK a) | a ist Agent}
initState Spy = {(Key (shrK a) | a ∈ bad)}
```

Die Funktion shrK ist eine Abbildung von dem jeweiligen Agenten auf seinen zugehörigen symmetrischen Schlüssel.
Hierzu werden noch weitere Eigenschaften von shrK definiert, wie z.B. die Injektivität, oder die Eigenschaft aller Schlüssel symmetrisch zu sein. Darauf wird hier aber nicht genauer eingegangen. Für das grundlegende Verständnis reicht daher die Signatur von shrK aus:

```
shrK : agent → key
```

Für die Menge bad gilt, dass der Angreifer stets in dieser Menge enthalten ist und der Server niemals enthalten ist. Alle anderen Beschränkungen für bad müssen sich in den Lemma befinden.

```
bad : agent set
```

Nachdem das Wissen des Angreifer zu Beginn erläutert ist, ergibt sich die Frage, welche Fähigkeiten der Angreifer besitzt, um neues Wissen zu erlangen.
Der Angreifer kann jede beliebige Nachricht abfangen und er kann auf das Wissen der infiltrierten Agenten zugreifen. Dieses Wissen wird durch spies repräsentiert.

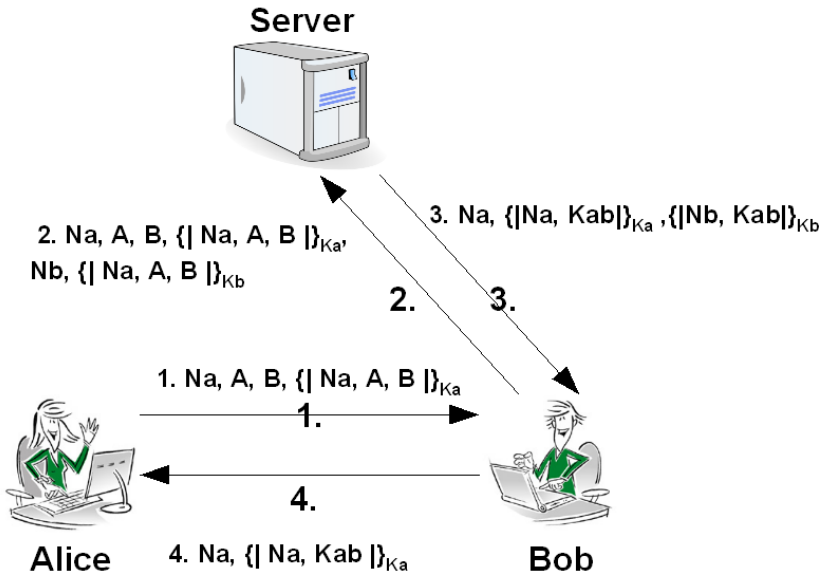
```
spies : event list → msg set
spies [] = initState Spy
spies ev#evs = case ev of
Says A B X ⇒ insert X (spies evs)
| Gets A X ⇒ spies evs
| Notes A X ⇒
if A ∈ bad then insert X (spies evs) else spies evs
```

2.5 Protokolle

Ein Protokoll wird nun als Menge von Ereignisfolgen formalisiert.
Das Protokoll beschreibt dabei, wie aus einer bereits bestehenden Ereignisfolge neue Ereignisfolgen erzeugt werden können beziehungsweise welche Ereignisse im nächsten Schritt hinzugefügt werden könnten. Isabelle bietet eine einfache Möglichkeit, induktiv aufgebaute Mengen zu definieren und bietet dafür ebenfalls reichhaltige Werkzeuge für Beweise an.
Damit ist nun das formale System erläutert. Im Weiteren betrachten wir nun, wie ein bestehendes Protokoll formalisiert wird.

3 Das Otway-Rees Protokoll

Wir betrachten nun das Beispiel eines vereinfachten Otway-Rees Protokolls:



Das Protokoll besteht aus den folgenden fünf Schritten:

1. A $\rightarrow$ B : Na, A, B, $\{|$Na, A, B $|\}_{Ka}$
   A sendet B die von A erzeugte Nonce Na unverschlüsselt und verschlüsselt.
   Implizit ist hier klar, dass die Nonce Na natürlich unverbraucht ist und der Empfänger B weder A selbst noch der Server S ist.

2. B $\rightarrow$ S : Na, A, B, $\{|$Na, A, B $|\}_{Ka}$, Nb, $\{|$Na, A, B$|\}_{Kb}$
   B empfängt die Nachricht von A, erzeugt die Nonce Nb, hängt diese verschlüsselt und unverschlüsselt an die Nachricht von A an, und sendet diese an den Server S.

3. S $\rightarrow$ B : Na, $\{|$Na, Kab$|\}_{Ka}$, $\{|$Nb, Kab$|\}_{Kb}$
   Der Server S empfängt die Nachricht von B, überprüft die Gleichheit von verschlüsselten und unverschlüsselten Nonces und sendet daraufhin an B eine Nachricht mit einem neuen Session Key Kab jeweils verschlüsselt für A und für B.

4. S $\rightarrow$ B : Na, $\{|$Na, Kab$|\}_{Ka}$
   B empfängt die Nachricht von S, entschlüsselt sich den Session Key, überprüft die Nonces Nb und sendet den Rest, also den Session Key verschlüs-

selt für A, weiter an A.

5. A empfängt die Nachricht von B, überprüft die Nonces Na, und erhält auch den Session Key.

Nun können die beiden Teilnehmer A und B mit dem Session Key weitere Nachrichten austauschen.

3.1 Formalisierung in Isabelle

Diese Protokollschritte in folgende logische Schlussregel transformiert werden. Dadurch definiert man eine Menge aller möglichen Traces für das Otway-Rees Protokoll.

Nil:

$$\frac{}{[] \in \text{otway}}$$

Fake:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{X} \in \text{synth (analz (spies evs))}\end{array}}{\text{Says Spy B X} \# \text{evs} \in \text{otway}}$$

Reception:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{Says A B X} \in \text{set evs}\end{array}}{\text{Gets B X} \# \text{evs} \in \text{otway}}$$

OR1:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{Na} \notin \text{used evs}\end{array}}{\text{Says A B } \{|\text{Na, A, B, } \{|\text{Na, A, B}|\}_{Ka}|\} \# \text{evs} \in \text{otway}}$$

OR2:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{Nb} \notin \text{used evs} \\ \text{Gets B } \{|\text{Na, A, B, X}|\} \in \text{set evs}\end{array}}{\text{Says B Server } \{|\text{Na, A, B, X, Nb, } \{|\text{Na, A, B}|\}_{Kb}|\} \# \text{evs} \in \text{otway}}$$

OR3:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{Kab} \notin \text{used evs} \\ \text{Gets Server } \{|\text{ Na, A, B, } \{|\text{Na, A, B}|\}_{Ka}\text{, Nb, } \{|\text{Na, A, B}|\}_{Kb}|\} \in \text{set evs}\end{array}}{\text{Says Server B } \{|\text{ Na, } \{|\text{Na, Kab}|\}_{Ka}\text{, } \{|\text{Nb, Kab}|\}_{Kb}|\} \# \text{evs} \in \text{otway}}$$

OR4:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{B} \neq \text{Server;} \\ \text{Says B Server } \{|\text{Na, A, B, X', Nb, } \{|\text{Na, A, B}|\}_{Ka}|\} \in \text{set evs} \\ \text{Gets B } \{|\text{Na, X, } \{|\text{Nb, Kab}|\}_{Kb}|\} \in \text{set evs}\end{array}}{\text{Says B A } \{|\text{Na, X}|\} \# \text{evs} \in \text{otway}}$$

Oops:

$$\frac{\begin{array}{c}\text{evs} \in \text{otway} \\ \text{Says Server B } \{|\text{Na, X, } \{|\text{Nb, Kab}|\}_{Kb}|\} \in \text{set evs}\end{array}}{\text{Notes Spy } \{|\text{Na, Nb, Kab}|\} \# \text{evs} \in \text{otway}}$$

Dies wird nun in Isabelle folgendermaßen formalisiert:

```
consts otway :: "event list set"
inductive "otway"
intros
Nil: "[] ∈ otway"

   Fake: "[| evsf ∈ otway; X ∈ synth (analz (spies evsf)) |]
==> Says Spy B X # evsf ∈ otway"
Reception: "[| evsr ∈ otway; Says A B X ∈ set evsr |]
==> Gets B X # evsr ∈ otway"

   OR1: "[| evs1 ∈ otway; Nonce NA ∉ used evs1 |]
==> Says A B {|Nonce NA, Agent A, Agent B,
Crypt (shrK A) {|Nonce NA, Agent A, Agent B|} |}
# evs1 ∈ otway"

   OR2: "[| evs2 ∈ otway; Nonce NB ∉ used evs2;
Gets B {|Nonce NA, Agent A, Agent B, X|} ∈ set evs2 |]
==> Says B Server
{|Nonce NA, Agent A, Agent B, X, Nonce NB,
Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
# evs2 ∈ otway"

   OR3: "[| evs3 ∈ otway; Key KAB ∉ used evs3;
Gets Server
{|Nonce NA, Agent A, Agent B,
Crypt (shrK A) {|Nonce NA, Agent A, Agent B|},
Nonce NB,
Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
∈ set evs3 |]
==> Says Server B
{|Nonce NA,
Crypt (shrK A) {|Nonce NA, Key KAB|},
Crypt (shrK B) {|Nonce NB, Key KAB|}|}
# evs3 ∈ otway"

   OR4: "[| evs4 ∈ otway; B ≠ Server;
Says B Server {|Nonce NA, Agent A, Agent B, X', Nonce NB,
Crypt (shrK B) {|Nonce NA, Agent A, Agent B|}|}
∈ set evs4;
Gets B {|Nonce NA, X, Crypt (shrK B) {|Nonce NB, Key K|}|}
∈ set evs4 |]
==> Says B A {|Nonce NA, X|} # evs4 ∈ otway"

   Oops: "[| evso ∈ otway;
Says Server B {|Nonce NA, X, Crypt (shrK B) {|Nonce NB, Key K|}|}
```

```
∈ set evso |]
==> Notes Spy {|Nonce NA, Nonce NB, Key K|} # evso ∈ otway"
```

Die Regel Nil steht für die leere Ereignisfolge. OR1-4 sind die bereits erwähnten Schritte des Otway-Rees Protokolls. Die Oops Regel steht für den unbeabsichtigten Verlust des Session Keys, und Fake steht für die betrügerischen Nachrichten, die der Angreifer erzeugen kann.
Im Detail findet man nun in der Formalisierung viele Punkte, die bereits implizit klar waren, wie z.B. Nachrichten werden nicht an sich selbst gesendet und die Noncen Na und Nb sind unverbraucht.
Die Überprüfung von Gleichheit von Noncen, die vorhin bei Schritt 3 und Schritt 4 explizit angegeben wurde, ist hier nun durch das Protokoll selbst gegeben.
Ebenfalls ist in dem Protokoll nicht angegeben, dass ein Agent nicht mehr als einmal auf eine Nachricht antwortet. Für die Korrektheit des Protokolls ist diese Einschränkung nicht notwendig, dies könnte aber auch das Aufdecken von Replay-Attacken verhindern.

4 Zu beweisende Eigenschaften eines korrekten Protokolls

Um die Korrektheit eines Protokolls zu beweisen, müssen folgende Eigenschaften gezeigt werden:

- possibility property
- forwarding lemmas
- regularity lemmas
- unicity theorems
- secrecy theorems
- authenticity guarantees

4.1 possibility property

Die possibility Eigenschaft stellt sicher, dass die Einzelschritte des Protokolls zusammenpassen und das gesamte Protokoll ausgeführt werden kann. Dies bedeutet nicht, dass wenn A das Protokoll beginnt, das Protokoll vollständig ausgeführt wird, denn keiner der Agenten muss auf die Nachrichten reagieren, noch ist gegeben, dass jede Nachricht auch beim Empfänger ankommt. Falls die possibility-Eigenschaft nicht gelten würde, gäbe es keinerlei Möglichkeit, also keine Ereignisfolge, bei der die vier Schritte des Otway-Rees Protokolls ausgeführt werden.

Konkret besagt die possibility-Eigenschaft, dass ein unverbrauchter Session-Key von B an A versendet wird.
Der Beweis ist klar und einfach. Man sieht leicht ein, dass es eine solche Ereignisfolge gibt und dass diese Schritte zusammenpassen.
possibility property:

- B $\neq$ Server
- Kab $\notin$ used []

$\Longrightarrow$ $\exists$ Na. $\exists$ evs $\in$ otway. Says B A $\{|$Na, $\{|$Na, Kab$|\}_{Ka}|\}$ $\in$ set evs

4.2 regularity lemmas

Diese Lemma machen Aussagen darüber, welche Eigenschaften für alle Ereignisfolgen gelten, trotz aller Möglichkeiten, die der Spion hat, Nachrichten zu senden und zu verändern. Bei Otway-Rees wird gezeigt, dass der Spion unter allen Umständen nicht an die Schlüssel der guten Agenten kommt. Keiner der Agenten außer dem Spion sendet seinen Schlüssel. Auch wenn der Spion Nachrichten mit Schlüsseln von infiltrierten Agenten sendet, wird er dadurch keinen neuen Schlüssel eines Agenten erhalten und sein Wissen vergrößern können.
Spy see shrK:

- evs $\in$ otway

$\Longrightarrow$ Key (shrK A) $\in$ parts (spies evs) $\Longleftrightarrow$ A $\in$ bad

4.3 unicity theorems

Es wird gezeigt, dass ein Session Key nur einmal vom Server vergeben wird und dieser daher dann die Nachricht (OR3) an B eindeutig identifiziert. Ein weiteres solches Resultat ist, dass die Nonce Na der Nachricht OR1 von A den gedachten Empfänger eindeutig bestimmt.
unique session keys:

- evs $\in$ otway
- Says Server B $\{|$ Na, X, $\{|$ Nb, Kab $|\}_{Kb}|\}$ $\in$ set evs
- Says Server B' $\{|$ Na', X', $\{|$ Nb', Kab $|\}_{Kb'}|\}$ $\in$ set evs

$\Longrightarrow$ X=X' $\wedge$ B=B' $\wedge$ Na=Na' $\wedge$ Nb=Nb'

4.4 secrecy theorem

Dieses Theorem besagt, dass bestimmte Nachrichtenteile von bestimmten Agenten gesehen werden können, und von anderen Agenten nicht. Während die vorherigen regularity lemmas schlicht dadurch bewiesen worden sind, dass bestimmte Nachrichten nicht versendet worden sind, ist dies bei den secrecy theorems nicht möglich. Stattdessen muss man zeigen, dass bestimmte Nachrichtenteile nie vom Spion entschlüsselt werden können. Beim Otway-Rees Protokoll ist man daran interessiert, dass der Spion nicht an den vom Server vergebenen Session Key kommen kann. Um diese Aussage jedoch beweisen zu können, müssen einige Annahmen gelten: zum einen darf keiner der beiden Agenten infiltriert worden sein, zum anderen darf der Spion nicht per Oops-Nachricht an den Schlüssel gelangt sein.
secrecy theorem:

- evs $\in$ otway
- A $\notin$ bad
- B $\notin$ bad
- Says Server B $\{|\text{Na}, \{|\text{Na}, \text{Kab}|\}_{Ka}, \{|\text{Nb}, \text{Kab}|\}_{Kb}|\} \in$ set evs
- Notes Spy $\{|\text{Na}, \text{Nb}, \text{Kab}|\} \notin$ set evs

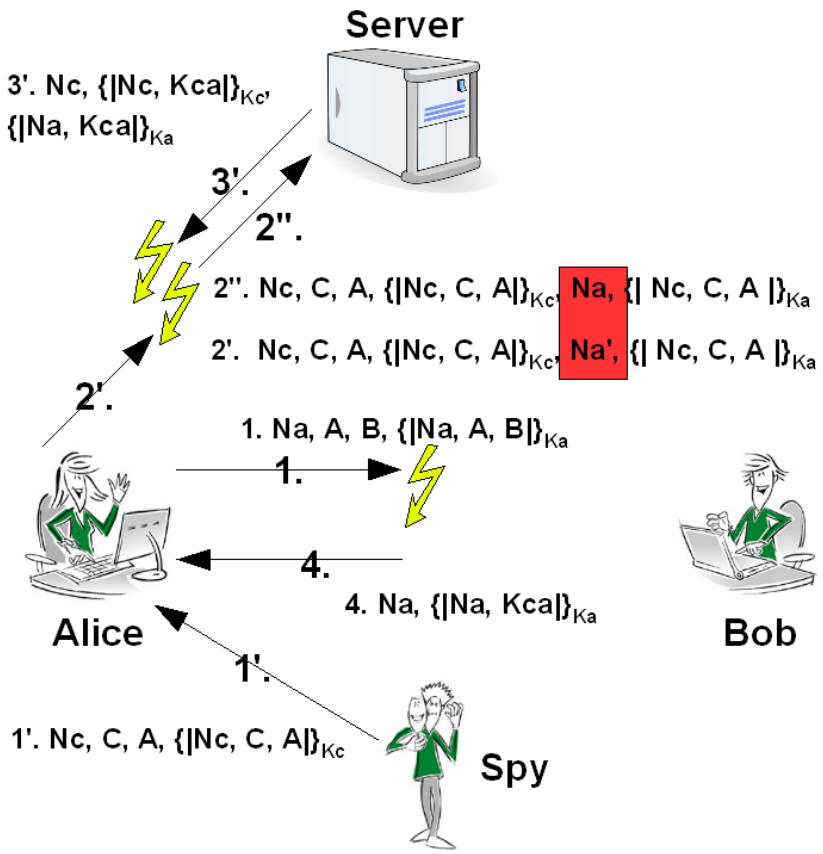
$\Longrightarrow$ Kab $\notin$ analz (spies evs)

4.5 authenticity guarantees

Das secrecy lemma ist ohne weitere Garantien wertlos, denn auch wenn das secrecy lemma gilt, können natürlich die Annahmen falsch sein. Wenn A eine Nachricht an B sendet, und daraufhin eine Antwort erhält, kann dann A sichergehen, dass diese Nachricht wirklich von B kommt, der diese Nachricht vom Server weitergeleitet hat?
Formal wird die folgende Eigenschaft wie folgt ausgedrückt:
Authentitätsgarantie für A:

- evs $\in$ otway
- A $\notin$ bad
- Says A B $\{|\text{Na}, \text{A}, \text{B}, \{|\text{Na}, \text{A}, \text{B}|\}_{Ka}|\} \in$ set evs
- Says B' A $\{|\text{Na}, \{|\text{Na}, \text{Kab}|\}_{Ka}|\} \in$ set evs

$\Longrightarrow \exists$ Nb. Says Server B
$\{|\text{Na}, \{|\text{Na}, \text{Kab}|\}_{Ka}, \{|\text{Nb}, \text{Kab}|\}_{Kb}|\} \in$ set evs
Für das vorgestellte Protokoll kann man diese Eigenschaft jedoch nicht zeigen! Aus dem Beweis lässt sich die folgende man-in-the-middle Attacke erkennen:



Diese Attacke ist möglich, da die Nonce Nb in Nachricht OR2 unverschlüsselt and den Server gesendet wird und die beiden Teile der Nachricht OR3 das gleiche Format besitzen. Modifiziert man das Protokoll nun folgendermaßen erhält man das korrekte Otway-Rees Protokoll:

1. A $\rightarrow$ B : Na, A, B, $\{|$Na, A, B $|\}_{Ka}$
2. B $\rightarrow$ S : Na, A, B, $\{|$Na, A, B $|\}_{Ka}$, $\{|$Nb, Na, A, B$|\}_{Kb}$
3. S $\rightarrow$ B : Na, $\{|$Na, Kab$|\}_{Ka}$, $\{|$Nb, Kab$|\}_{Kb}$
4. S $\rightarrow$ B : Na, $\{|$Na, Kab$|\}_{Ka}$

Nun ist das Ändern der Nonce Nb nicht mehr möglich, da diese jetzt im Schritt 2 verschlüsselt übertragen wird. Für dieses Protokoll kann alle besprochenen Eigenschaften mit wenigen Änderungen beweisen. Desweiteren lässt sich nun beweisen, dass, wenn A die Nachricht OR1 versendet hat und die Nachricht OR4 erhält, so muss die Nachricht OR3 vom Server an B gesendet worden sein. Einfach gesagt, damit wurde gezeigt, dass die einzelnen Nachrichten im Protokoll auch den gedachten Weg genommen haben.

Damit ist das Otway-Rees Protokoll verifiziert.

5 Zusammenfassung und Ausblick

Mit der vorgestellten Methode lässt sich also einfach ein Sicherheitsprotokoll verifizieren. Die Formalisierung und die vorgestellten Beweise befinden sich unter HOL/Auth/ in den Isabelle Sources.
Man könnte nun vermuten, dass dadurch jedes verifizierte Sicherheitsprotokoll auch sicher ist. Dies ist aber leider nicht richtig. Zum einen wird die Verschlüsselung als sicher angenommen, was nicht immer der Fall ist. So ist das Recursive Authentification Protocol verifiziert worden, jedoch ist ebenfalls eine mögliche XOR-Attacke auf dieses Protokoll aufgedeckt worden. Zum anderen können natürlich implementierungsabhängige Fehler auftreten, und damit ebenfalls die Sicherheit beeinträchtigen.

Literatur

[1] Lawrence C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. J. Computer Security 6 (1998), 85–128. (http://www.cl.cam.ac.uk/users/lcp/papers/Auth/jcs.pdf)

[2] Lawrence C. Paulson. Proving security protocols correct. IEEE Symposium on Logic in Computer Science. Trento, Italy (1999). (http://www.cl.cam.ac.uk/users/lcp/papers/Auth/lics.pdf)

[3] Tobias Nipkow, Lawrence C. Paulson, Markus Wenzel. A Proof Assistant for Higher-Order Logic. (http://isabelle.in.tum.de/dist/Isabelle/doc/tutorial.pdf)