Analyse kryptographischer Protokolle mit Hilfe asynchroner Produktautomaten

Michael Ueckerdt (156675)

2. Februar 2003

Inhaltsverzeichnis

1 Einführung

Die Sicherheit von Daten hängt heutzutage nicht mehr allein von den verwendeten kryptographischen Algorithmen ab. Einen ebenso großen Einfluss haben die kryptographischen Protokolle, welche bei der Übertragung dieser Daten bzw. zur Errichtung eines sicheren Übertragungskanals eingesetzt werden. Daraus leitet sich das Bedürfnis ab, die Sicherheit solcher Protokolle mit formalen Methoden zu überprüfen.

Das hier vorgestellte Verfahren modelliert das zu untersuchende Protokoll mit asynchronen Produktautomaten (kurz APA) und gehört damit in die Klasse der Modelle, welche auf Zustandsautomaten basieren. Wie auch bei anderen Methoden lassen sich hierbei lediglich Sicherheitslücken nicht jedoch die Sicherheit eines Protokolls nachweisen.

Ein erstes Problem bei der formalen Analyse von Protokollen liegt bereits in deren Spezifikation. Häufig liegen solche Protokollspezifikationen nicht vollständig formal vor, sondern enthalten bereits implizite Annahmen über die verwendeten Verschlüsselungsalgorithmen oder interne Aktionen der einzelnen Protokoll-Agenten. Die Sicherheit von Protokollen hängt jedoch auch von Informationen ab, die informal oder implizit vorliegen.

Viele Analysemodelle setzen voraus, daß die beteiligten Agenten die Typen von einzelnen Nachrichtenelementen überprüfen können, und so z.B. einfache Zufallszahlen von Schlüsseln unterscheiden können. In der Praxis gibt es jedoch Angriffe, die auf Typen-Verwirrung basieren. Mit dem hier vorgestellte Verfahren kann sehr genau modelliert werden, welche Typprüfungen vorgenommen werden und welche nicht. Zur Demonstration wird im folgenden ein Angriff auf das Needham-Schroeder Protokoll präsentiert, welcher nicht auf kompromittierten Sitzungsschlüsseln basiert.

Das Verfahren zur Modellierung kryptographischer Protokolle wurde von S. Gürgens, P. Ochsenschläger und C. Rudolph entwickelt und 2001 in einem GMD-Report (GMD - Gesellschaft für Mathematik und Datenverarbeitung) veröffentlicht. Zu Analyse der Protokolle (bzw. des entsprechenden APA) wird das Simple Homomorphism Verification Tool (kurz SHVT) eingesetzt, welches von P. Ochsenschläger, J. Repp, R. Rieke und U. Nitsche entwickelt wurde.

2 Ein APA für das Needham-Schroeder Protokoll

2.1 formale Definition eines APA

Ein APA kann als eine Menge elementarer Automaten betrachtet werden. Der Zustand eines APA ist in einzelne Zustandskomponenten untergliedert. Die Zustandsmenge eines APA ergibt sich dann aus dem Kreuzprodukt der zu den einzelnen Komponenten gehörenden Zustandsmengen. Die Zustandsmengen der elementaren Automaten sind dann Teilmenge des Kreuzprodukts einiger Zustandskomponenten des APA (bzw. der dazugehörigen Zustandsmengen). Die elementaren Automaten haben gewissermaßen Zugriff auf einige der Zustandskomponenten und können miteinander kommunizieren, indem sie den Inhalt ihrer gemeinsamen Komponenten verändern.

Formal besteht ein APA (A) aus einer Familie von Zustandsmengen (Z_S) (das entspricht den Zustandskomponenten), einer Familie elementarer Automaten (A_e) einer Nachbarschaftsrelation N und einem Startzustand z_0.

$$\begin{array}{rcl} (A) & = & [(Z_s)_{s\in\mathcal{S}}, (A_e)_{e\in\mathcal{E}}, N, z_0] \\ N & : & \mathcal{E} \to \mathcal{P}(\mathcal{S}) \\ A_e & = & (\Phi_e, T_e, \Delta_e) \\ T_e & \subseteq & \times_{S\in N(e)} Z_S \\ \Delta_e & \subseteq & T_e \times \Phi_e \times T_e \\ z_0 & \in & \times_{S\in(S)} Z_S \end{array}$$

$\mathcal{S}$ und $\mathcal{E}$ sind hierbei die Indexmengen für die Zustandsmengen bzw. die elementaren Automaten und $\mathcal{P}(\mathcal{S})$ entspricht der Potenzmenge von $\mathcal{S}$. Ein elementarer Automat ist ein Tupel aus einem Alphabet Φ_e, einer Zustandsmenge T_e und einer Zustandsüberführungsfunktion Δ_e. Die Zustandsmenge T_e eines elementaren Automaten ist dabei Teilmenge der Kreuzmenge aller benachbarten Zustandsmengen $Z_S, S \in N(e)$. Somit sind die elementaren Automaten über die Zustandskomponenten Z_S entsprechend der Nachbarschaftsrelation miteinander verbunden. Weiterhin wird festgelegt, daß jeder elementare Automat mindestens eine benachbarte Zustandskomponente hat und jede Zustandskomponente mit mindestens einem elementaren Automaten verbunden sein muss:

$$\begin{array}{l} \forall_{e\in\mathcal{E}} : N(e) \neq \emptyset \\ \mathcal{S} = \bigcup\limits_{e\in\mathcal{E}} N(e) \end{array}$$

Ein Zustandsübergang im APA entspricht einem Zustandsübergang eines der enthaltenen elementaren Automaten. Das Verhalten eines APA wird daher durch alle möglichen Folgen von solchen Zustandsübergängen repräsentiert.

2.2 Das Needham-Schroeder Protokoll

Zur Veranschaulichung soll nun ein APA für das Needham-Schroeder Protokoll konstruiert werden. Dazu zunächst kurz zur Arbeitsweise des Protokolls. Ziel des Needham-Schroeder Protokolls ist es, einen sicheren Kommunikationskanal zwischen zwei Agenten (üblicherweise A und B) zu schaffen. K_{XY} bezeichnet hierbei den gemeinsamen symmetrischen Schlüssel der Agenten X und Y. Es werden folgende Nachrichten ausgetauscht:

$$\begin{array}{lcl} 1.\ A \to S & : & A, B, R_A \\ 2.\ S \to A & : & \{R_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}} \end{array}$$

$$
\begin{aligned}
3.\ A \rightarrow B &: \{K_{AB}, A\}_{K_{BS}} \\
4.\ B \rightarrow A &: \{R_B\}_{K_{AB}} \\
5.\ A \rightarrow B &: \{R_B - 1\}_{K_{AB}}
\end{aligned}
$$

Zunächst sendet A eine Nachricht an den Server S, welche den Namen von A, den Namen des gewünschten Kommunikationspartners B sowie eine von A erzeugte Zufallszahl R_A enthält. Der Server antwortet an A mit einer durch K_{AS} verschlüsselten Nachricht, welche das vorher erzeugte R_A, den Namen von B, einen neu erzeugten gemeinsamen Schlüssel K_{AB} für A und B sowie einen mit K_{BS} verschlüsselten Teil enthält. In Schritt drei leitet Agent A den mit K_{BS} verschlüsselten Teil an B weiter. B schickt daraufhin eine neu erzeugte Zufallszahl R_B verschlüsselt mit K_{AB} an A, und A reagiert indem er $R_B - 1$ mit demselben Schlüssel verschlüsselt zurückschickt.

Nach diesen fünf Schritten sollen A und B einen gemeinsamen sicheren Schlüssel K_{AB} besitzen. Es wurde jedoch ein Angriff gefunden, welcher diese Schlussfolgerung widerlegt. Er beruht darauf, daß ein Sitzungsschlüssel nur eine begrenzte Zeit lang sicher bleibt. Hier soll jedoch ein anderer Angriff präsentiert werden, für den der Angreifer keinen alten Sitzungsschlüssel benötigt.

2.3 Spezifikation eines APA für das Needham-Schroeder Protokoll

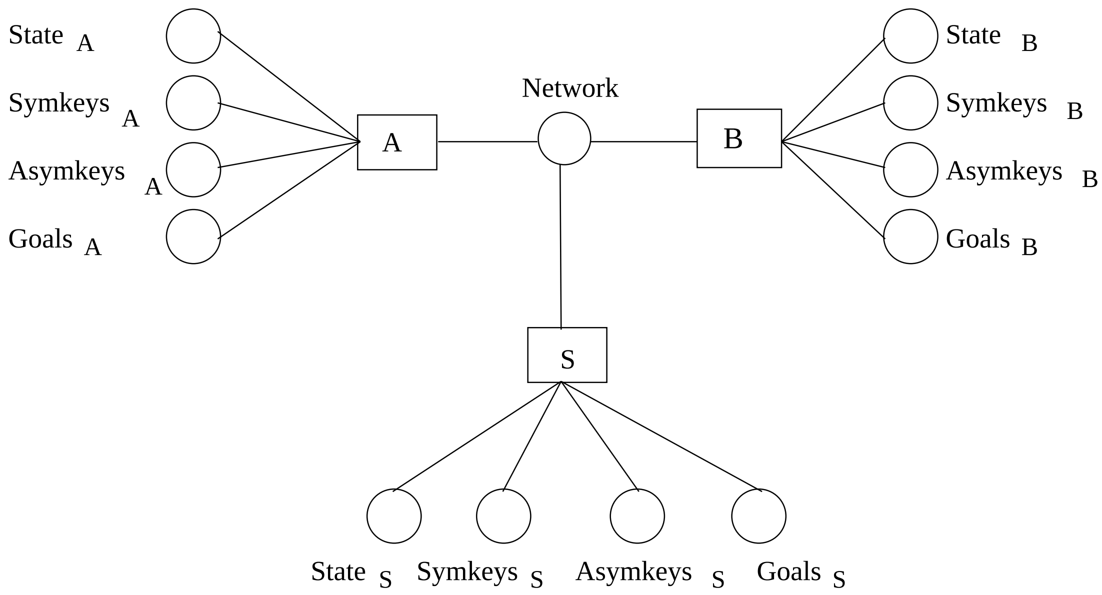
Zunächst einige grundlegende Mengen:

$$
\begin{aligned}
&Agents && \text{Menge der Namen von Agenten} \\
&Keywords &=& \{new\ session\ key, session\ key, start, respond, agent, server\} \\
&Keys &=& \{(P, Q, sym) | P, Q \in Agents\} \\
&Predicates && \text{Menge der Prädikate auf globalen Zuständen} \\
&I\!N && \text{die natürlichen Zahlen}
\end{aligned}
$$

Es wird davon ausgegangen, daß die Agenten keine Typprüfungen auf den Nachrichten vornehmen können und so die im Protokoll verwendeten Zufallszahlen einfach natürliche Zahlen sind. Soll die Möglichkeit einer Unterscheidung gegeben sein, so müssen einfach nur weitere Mengen definiert werden (z.B. *Random*, *Nonce*, etc.).

Die Grafik zeigt die Struktur des APA für das Needham-Schroeder Protokoll. Die Kreise entsprechen dabei den Zustandskomponenten, die Rechtecke repräsentieren elementare Automaten. Durch die Kanten ist angegeben, auf welche Zustandskomponenten der jeweilige elementare Automat Zugriff hat.

Der APA des Needham-Schroeder Protokolls enthält drei elementare Automaten A, B und S. Diese Automaten sind über die Zustandskomponente $Network$ miteinander verbunden. Zusätzlich besitzen A, B und S noch jeweils vier weitere private Zustandskomponenten (d.h. auf die jeweils nur ein einziger Automat Zugriff hat) $State_e$, $Symkeys_e$, $Asymkeys_e$ und $Goals_e$, wobei $e \in \{A, B, S\}$ ist.



Weiter muss die Menge der zugelassenen Nachrichten definiert werden. Die Vereinigung der Mengen $Agents$, $Keywords$, $Keys$ und IN bildet die Menge der atomaren Nachrichten. Darauf aufbauend kann die Menge aller Nachrichten M rekursiv definiert werden:

1. Jede atomare Nachricht ist Element von M.
2. Wenn $m_1, ..., m_n \in M$, dann ist auch $(m_1, ..., m_n) \in M$.
3. Wenn $k, m \in M$ dann sind auch $encrypt(k, m), decrypt(k, m) \in M$.
4. Wenn $m \in M$ dann ist auch $sub(m, 1) \in M$.

Zusätzlich werden noch die Standard-Funktionen $element(k, (m_1, ..., m_n))$ und $length(m_1, ..., m_n)$ eingeführt, welche das k-te Element eines Tupels bzw. die Anzahl der Elemente des Tupels liefern. Für die Funktionen $encrypt$ und $decrypt$ werden die folgenden symbolischen Eigenschaften definiert:

$$\begin{aligned} decrypt(k, encrypt(k, m)) &= m \\ encrypt(k, decrypt(k, m)) &= m \end{aligned}$$

Diese beiden Eigenschaften modellieren symmetrische Verschlüsselung. Für die Analyse wird in diesem Fall perfekte Verschlüsselung angenommen, d.h. Schlüssel können nicht erraten werden, für die Erzeugung von $encrypt(k, m)$ und $decrypt(k, m)$ müssen m und k bekannt sein und aus $encrypt(k, m) = encrypt(k', m')$ folgt, daß $m = m'$ und $k = k'$ sind. Für andere Analysen können diese Annahmen jedoch durchaus gelockert werden. So lässt sich beispielsweise durch die Annahme $blockcrypt(k, (m_1, m_2)) = (blockcrypt(k, m_1), blockcrypt(k, m2))$ ein Blockchiffre modellieren.

Für die Schlüssel wird festgelegt, daß $(P, Q, sym) = (Q, P, sym)$ ist. Durch die obigen Eigenschaften wird auf natürliche Weise eine kürzeste Normalform für Nachrichten festgelegt, in denen keine sich gegenseitig aufhebenden *encrypt*- und *decrypt*-Funktionen mehr enthalten sind. Die Menge dieser kürzesten Normalformen wird mit $Messages$ bezeichnet. Die Elemente von $Messages$ bilden nun den Inhalt der Zustandskomponenten des APA. Eine Ausnahme bildet die Komponente $Network$, welche Elemente von $Agents \times Messages$ enthält. Die symmetrischen Schlüssel eines Agenten P werden als Tupel (Q, sym, key) in die $Symkeys_P$-Komponente eingetragen (wobei key den symmetrischen Schlüssel und nicht das Element aus $Keywords$ meint).

Der Nachrichtenaustausch zwischen den Agenten läuft über die Komponente $Network$. Um eine Nachricht M an einen Agenten P zu schicken, wird das Tupel (P, M) der $Network$-Komponente hinzugefügt. Beim Empfangen der Nachricht wird das Tupel dann im dadurch ausgelösten Zustandsübergang des empfangenden Agenten aus $Network$ gelöscht. Mit Ausnahme des Angreifers können Agenten nur an sich selbst gerichtete Nachrichten empfangen.

Jetzt muss noch für jede der Zustandskomponenten $S \in \mathcal{S}$ die zugehörige Zustandsmenge Z_S definiert werden. Jede dieser Zustandsmengen ist ein Multiset. Ein Multiset über einer Menge A kann durch eine Funktion $f : A \to I\!N$ repräsentiert werden, wobei $f(a)$ angibt wie oft das Element $a \in A$ im Multiset f enthalten ist. Die Menge aller Multisets über einer Menge A wird im folgenden als $\overline{\mathcal{P}}(A)$ bezeichnet. Damit lassen sich die Zustandsmengen Z_S des APA definieren:

$$\begin{aligned}
Z_{Network} &= \overline{\mathcal{P}}(Agents \times Messages) \\
Z_{State_P} &= \overline{\mathcal{P}}(Messages) \\
Z_{Symkeys_P} &= \overline{\mathcal{P}}(Agents \times Symflags \times Keys) \\
Z_{Asymkeys_P} &= \overline{\mathcal{P}}(Agents \times Asymflags \times Keys) \\
Z_{Goals_P} &= \overline{\mathcal{P}}(Predicates)
\end{aligned}$$

Für $Symflags$ und $Asymflags$ werden dabei geeignete Mengen eingesetzt, für Needham-Schroeder ist $Symflags = \{sym\}$ und $Asymflags$ leer.

Als nächstes soll der Anfangszustand des APA spezifiziert werden. Er kann durch den Inhalt der Zustandskomponenten angegeben werden:

$$\begin{aligned}
State_A &= \{(B, agent), (S, server), (start, B)\} \\
Symkeys_A &= \{(S, sym, (A, S, sym))\} \\
State_B &= \{(A, agent), (S, server), (respond, A)\} \\
Symkeys_B &= \{(S, sym, (B, S, sym))\} \\
State_S &= \{(A, agent), (B, agent)\} \\
Symkeys_S &= \{(A, sym, (A, S, sym)), (B, sym, (B, S, sym))\}
\end{aligned}$$

Die Agenten A und B kennen sich also gegenseitig, kennen den Server S und haben beide jeweils einen Schlüssel für die Kommunikation mit S. S kennt A und B und besitzt ebenfalls die symmetrischen Schlüssel zur Kommunikation mit diesen. Das Token $(start, B)$ in $State_A$ ermöglicht Agent A das Protokoll mit B als gewünschtem Kommunikationspartner zu initiieren und $(respond, A)$ in $State_B$ gibt B die Möglichkeit darauf zu reagieren. Alle nicht aufgeführten Zustandskomponenten sind leer.

Im letzten Schritt der Spezifikation ist die Zustandsüberführungsfunktion für die einzelnen Agenten zu modellieren. Sie kann z.B. notiert werden, indem die Voraussetzungen für den Zustandsübergang eines Agenten und die damit verbundenen Aktionen aufgeschrieben werden. Als Beispiel nun nachfolgend der zu Schritt vier des Needham-Schroeder Protokolls gehörende Zustandsübergang. B erhält hierbei die Nachricht $\{K_{AB}, A\}_{K_{BS}}$ und schickt daraufhin $\{R_B\}_{K_{AB}}$ an A.

Voraussetzung	
$(B, M) \in Network$	Nachricht M an B liegt in $Network$
$(respond, A) \in State_B$	B kann auf das von A gestartete Protokoll reagieren
$(A, agent) \in State_B$	B kennt Agent A
$(S, server) \in State_B$	B kennt den Server S
$(S, sym, K_{BS}) \in Symkeys_B$	B teilt einen symmetrischen Schlüssel mit S
$element(2, decrypt(K_{BS}, M)) = A$	nach Entschlüsselung von M mit dem Schlüssel K_{BS} ist die zweite Komponente der Nachricht der Name von A
Aktionen von B	
$K_{AB} := element(1, decrypt(K_{BS}, M))$	die erste Komponente des Klartextes von M wird als K_{AB} gesichert
$R_B \in newRandom$	B generiert eine neue Zufallszahl R_B
$(B, M) \hookleftarrow Network$	B entfernt die Nachricht aus $Network$
$(new\ session\ key, A, K_{AB}, R_B) \hookrightarrow State_B$	B speichert K_{AB} als neuen Sitzungsschlüssel
$(A, encrypt(K_{AB}, R_B)) \hookrightarrow Network$	B schickt die nächste Nachricht an A

Die Spezifikation der anderen Schritte kann in [**?**] nachgelesen werden. Damit ist die Funktionalität des Protokolls vollständig modelliert. Durch diese Zustandsübergänge wird jedoch eigentlich beschrieben, was ein Agent in *Rolle* A oder B tun muss. Zu der formalen Beschreibung des Protokolls gehört also noch eine Festlegung, welche Kombination von Rollen verboten ist, und welche Rollen vom Angreifer übernommen werden können. Eine mögliche Festlegung für Needham-Schroeder wäre zu verbieten, daß ein Agent gleichzeitig zwei Rollen übernimmt (A und S, B und S oder A und B), und daß der Angreifer die Rolle von S einnehmen kann. Für die obige Spezifikation des Anfangszustands bedeutet daher das Tupel $(B, agent)$ in $State_A$, daß A alle Agenten kennt, die die Rolle B übernehmen können. Im Unterschied dazu müssen bei Tupeln der Form $(S, sym, (A, S, sym))$ in den $Symkeys$-Komponenten nun die Platzhalter A und S durch einzelne Agenten ersetzt werden, da ein Agent nur seinen symmetrischen Schlüssel für die Kommunikation mit dem Server kennt (und nicht die aller Agenten, die die Rolle A übernehmen können).

Die Sicherheit eines Protokolls kann nur relativ zu irgendwelchen Sicherheitsanforderungen bzw. Sicherheitszielen überprüft werden. Daher müssen die Sicherheitsziele mit spezifiziert werden. Sie werden in den $Goal$-Komponenten der einzelnen Agenten vermerkt. So kann ein Agent nach jeder Aktion nach der aus seiner Sicht ein bestimmtes Sicherheitsziel erfüllt sein soll, ein entsprechendes Prädikat in seiner $Goal$-Komponente vermerken. Damit ergibt sich dann eine Definition für die Sicherheit eines Protokolls.

Definition: *Ein Protokoll heißt sicher, wenn jedes Prädikat welches in einer der $Goal$-Komponenten steht wahr ist, solange es dort steht.*

Ein Sicherheitsziel nach dem Empfang der letzten Nachricht des Needham-Schroeder Protokolls wäre, daß der Sitzungsschlüssel K_{AB} nun keinem Agenten außer A und B bekannt ist. In der Zustandsüberführungsfunktion drückt sich das wie folgt aus:

Aktionen von B	
$(session\ key, A, K_{AB}) \hookrightarrow State_B$	B akzeptiert K_{AB} als Sitzungsschlüssel
$\forall_{P \in Agents \setminus \{A,B\}} : not\_knows(P, K_{AB}) \hookrightarrow Goals_B$	

3 Die Analyse des Protokolls

3.1 Der Analyse-APA

Die Analyse des Protokolls läuft auf einem speziell dafür zu schaffenden Analyse-APA. Dazu enthält er für jeden Agenten P, welcher sich gemäß der Protokollspezifikation verhält einen elementaren Automaten mit den benachbarten Zustandskomponenten $State_P$, $Symkeys_P$, $Asmykeys_P$, $Goals_P$ und $Network$. Zusätzlich existiert noch eine Zustandskomponente $Global$, welche von allen Agenten benutzt werden kann um z.B. Zufallszahlen zu generieren. Weiterhin gibt es einen elementaren Automaten G, welcher Zugriff auf $Network$, $Global$ und sämtliche $State$-Komponenten der anderen Agenten (inklusive des Angreifers) hat. Durch G können Zustandsübergänge beeinflusst werden, für das Needham-Schroeder Protokoll wird G allerdings nicht benötigt.

Zusätzlich wird im Analyse-APA auch ein Angreifer E spezifiziert. Also werden für E ein elementarer Automat und die zu E benachbarten Zustandskomponenten $State_E$, $Symkeys_E$, $Asymkeys_E$, und $Sent_E$ (um mehrfaches Abschicken derselben Nachricht zu vermeiden) in den APA eingefügt. Der Angreifer hat Zugriff auf $Network$ und $Global$. Anders als die restlichen Agenten kann der Angreifer jedes Tupel aus $Network$ entfernen, auch wenn er nicht der Adressat ist. Er kann die Bestandteile der Nachricht als Wissen in seine Zustandskomponente übernehmen, und er kann dem eigentlichen Adressaten eine eigene Nachricht zukommen lassen (d.h. der Angreifer weiß auch an wen die abgefangene Nachricht gehen sollte). Das Wissen des Angreifers kann rekursiv definiert werden:

1. Der Angreifer kennt alle Daten, die im Anfangszustand in seinen Komponenten stehen.

2. Der Angreifer kennt alle Agenten, die als Adressaten abgefangener Nachrichten auftraten, d.h. als erste Komponente der Paare aus $Network$.

3. Der Angreifer kennt alle von ihm gelesenen Nachrichten, d.h. die zweiten Komponenten der Paare aus $Network$.

4. Der Angreifer kennt alle Bestandteile der von ihm gelesenen Nachrichten (verschlüsselter Text wird dabei als ein atomarer Bestandteil betrachtet).

5. Der Angreifer kennt alle Nachrichten, die er durch Konkatenation generieren kann.

6. Der Angreifer kann neue Nachrichten durch Anwenden von *encrypt* generieren (hierbei kann jede ihm bekannte Nachricht als Schlüssel dienen).

7. Der Angreifer kann Zufallszahlen erzeugen.

8. Der Angreifer kennt jeden Klartext, den er durch Anwendung von *decrypt* auf eine Nachricht erzeugen kann, wenn er den passenden Schlüssel kennt.

9. Der Angreifer kann die symbolische Funktion *sub* auf Nachrichten anwenden.

3.2 Das Analyse-Szenario

Für ein konkretes zu analysierendes Szenario müssen jetzt noch die teilnehmenden Agenten (inklusive der Rollen die sie übernehmen können), sowie der Startzustand (d.h. der Inhalt sämtlicher Zustandskomponenten am Anfang) spezifiziert werden. In diesem Fall gibt es zwei Agenten $Alice$ (übernimmt Rolle A) und Bob (Rolle B) und einen Server (Rolle S). Des weiteren gibt es einen Angreifer E, der nicht an dem Protokoll teilnimmt, d.h. keine der Rollen übernehmen kann und auch keinen gemeinsamen Schlüssel mit dem Server besitzt. Er kennt jedoch die Namen der Agenten $Alice$, Bob und des Servers (d.h. er kann Nachrichten an diese schicken). Insgesamt sieht der Startzustand wie folgt aus:

$$\begin{aligned}
State_{Alice} &= \{(Bob, agent), (Server, server), (start, Bob)\} \\
Symkeys_{Alice} &= \{(Server, sym, (Alice, Server, sym))\} \\
State_{Bob} &= \{(Alice, agent), (Server, server), (respond, Alice)\} \\
Symkeys_{Bob} &= \{(Server, sym, (Bob, Server, sym))\} \\
State_{Server} &= \{(Alice, agent), (Bob, agent)\} \\
Symkeys_{Server} &= \{(Alice, sym, (Alice, Server, sym)), (Bob, sym, (Bob, Server, sym))\} \\
State_{E} &= \{(Alice, agent), (Bob, agent), (Server, server)\}
\end{aligned}$$

Alle nicht aufgeführten Zustandskomponenten sind zu Anfang leer. Es soll nun überprüft werden, ob alle in den *Goal*-Komponenten enthaltenen Prädikate wahr sind, solange sie dort enthalten sind. Insbesondere also das von B nach dem letzten Schritt eingefügte Prädikat $(\forall_{P \in Agents \setminus \{Alice, Bob\}} : not\_knows(P, K_{Alice,Bob}))$. Die Analyse ist eine Zustandsraumsuche auf den erreichbaren Zuständen, durchgeführt vom SH Verification Tool. Wird ein Zustand gefunden in dem ein aktuelles Ziel-Prädikat verletzt ist, kann mit demselben Tool der Weg der dahin führt ausgegeben werden. Eine Analyse auf dem soeben definierten Analyse-APA liefert einen Zustand, welcher das angegebene Prädikat verletzt. Nachfolgend nun der Weg zu diesem Zustand.

Zunächst gibt sich der Angreifer als *Bob* aus und startet einen Protokoll-Lauf mit *Alice*. Dazu generiert er eine Zufallszahl $random_E$ und schickt die entsprechende Nachricht an den *Server*.

$$E \quad \rightarrow \quad Server: \ (Bob, Alice, random_E)$$

Der Server generiert daraufhin einen neuen Sitzungsschlüssel K und eine Nachricht, welche er an *Bob* 'zurück'schickt.

$$\begin{aligned} Server \quad \rightarrow \quad Bob: & \ (encrypt((Bob, Server, sym), (random_E, Alice, K, M_{Alice}))) \\ wobei: & \ \ M_{Alice} = encrypt((Alice, Server, sym), (K, Bob)) \end{aligned}$$

Bob empfängt dann diese Nachricht, entschlüsselt sie und stellt fest, daß die zweite Komponente dem Namen *Alice* entspricht. Außerdem kann *Bob* auf einen von *Alice* gestarteten Protokoll-Lauf reagieren. D.h. *Bob* interpretiert diese Nachricht als dritte Nachricht eines von *Alice* gestarteten Protokoll-Laufs obwohl sie als zweite Nachricht des Protokolls gedacht ist. Entsprechend des oben aufgeführten Zustandsübergangs betrachtet *Bob* nun $random_E$ (die erste Komponente dieser Nachricht) als neuen Sitzungsschlüssel, generiert eine Zufallszahl $random_B$ und schickt diese verschlüsselt mit $random_E$ an *Alice*.

$$Bob \quad \rightarrow \quad Alice: \ (encrypt(random_E, random_B))$$

Der Angreifer fängt die Nachricht ab, kann sie entschlüsseln (da er $random_E$ kennt), und schickt die Antwort die *Bob* erwartet.

$$E \quad \rightarrow \quad Bob: \ (encrypt(random_E, sub(random_B, 1)))$$

Schließlich erhält *Bob* diese Nachricht, entschlüsselt sie und stellt fest, daß sie den Klartext $sub(random_B, 1)$ enthält. Also betrachtet *Bob* von nun an $random_E$ als gültigen Sitzungsschlüssel für die Kommunikation mit *Alice*. Dadurch wird das Prädikat, welches *Bob* im letzten Schritt in seine *Goals*-Komponente einfügt falsch, da es einen Agenten gibt (nämlich E), welcher ebenfalls diesen Schlüssel kennt. Der Angreifer könnte sich jetzt gegenüber *Bob* als *Alice* ausgeben.

Dieser Angriff wird als *arity-attack* bezeichnet (arity heißt Stelligkeit). Er kann verhindert werden, indem *Bob* beim Erhalt der dritten Nachricht des Protokolls prüft, wieviele Komponenten die Nachricht enthält. Der Angriff ist ebenfalls unmöglich, wenn die Agenten in der Lage sind zwischen Zufallszahlen und

Schlüsseln zu unterscheiden. In diesem Fall würde es genügen, wenn bei *Bob* im dritten Schritt die Voraussetzung $K_{AB} \in Keys$ hinzugefügt wird.

4 Das SH-Verification Tool

Das SH Verification-Tool (SH steht für Simple Homomorphism) hat zum Ziel, kooperierende Systeme auf bestimmte Eigenschaften zu überprüfen. APA sind eine Möglichkeit kooperierende Systeme zu spezifizieren. Wenn in einem gegebenen Zustand eines APA mehrere elementare Automaten einen Zustandsübergang durchführen können (d.h. es sind Voraussetzungen für verschiedene Zustandsübergänge erfüllt) ist nicht definiert, welcher der Agenten einen Zustandsübergang macht. Das ist eine Form von Nicht-Determinismus, welcher typisch für kooperierende Systeme ist. In solchen Nicht-Deterministischen Systemen ist das Verifizieren von bestimmten Eigenschaften sehr schwierig, da der Zustandsraum (die Menge der möglichen Zustände) sehr schnell mit der Anzahl der beteiligten Komponenten wächst. Um dennoch eine Analyse durchführen zu können, betrachtet das SHVT Abstraktionen des eigentlichen Systems. Unter bestimmten Bedingungen lassen sich aus Eigenschaften der Abstraktionen Aussagen über entsprechende Eigenschaften des Originalsystems ableiten.

Das SHVT ist nicht an eine formale Spezifikationssprache gebunden. Für den praktischen Einsatz muss es daher mit Tools kombiniert werden, welche 'labeled transition systems' (kurz LTS) generieren. Derzeit werden vom SHVT Produktnetze (eine spezielle Form von Petrinetzen) und APA als Spezifikationssprachen unterstützt. Ein LTS ist gegeben durch einen Zustandsraum, einer Menge von Transitionen (Zustandsübergängen) darauf und einen Startzustand. Es kann daher mit einem gerichteten Graphen visualisiert werden. Für Systeme aus der Praxis sind solche Darstellungen jedoch extrem umfangreich und daher nicht wirklich zur Veranschaulichung geeignet. Ein LTS definiert eine formale Sprache - die lokale Sprache des LTS. Die Transitionen des LTS bilden das Alphabet Σ dieser Sprache, d.h ein Buchstabe entspricht einem Tripel aus Zustand, Beschriftung der Transition, und Folgezustand. Ein Wort der lokalen Sprache eines LTS ist also eine mögliche Folge von Zustandsübergängen ausgehend vom Startzustand. Mit Σ^* wird die Menge aller endlichen Folgen von Buchstaben aus Σ notiert.

Auf solchen formalen Sprachen lassen sich Abstraktionen durch Homomorphismen repräsentieren. Eine Abbildung $h : \Sigma^* \to \Sigma'^*$) heißt Homomorphismus, wenn $h(\epsilon) = \epsilon$ und $h(uv) = h(u)h(v)$ gilt, wobei ϵ das leere Wort und u, v beliebige Wörter aus Σ^* sind. Ein Homomorphismus heißt alphabetisch, wenn er einzelne Buchstaben aus Σ auf einzelne Buchstaben in Σ' oder das leere Wort abbildet ($h(\Sigma) \subseteq \Sigma' \cup \{\epsilon\}$). Alle vom SHVT betrachteten Abstraktionen sind alphabetische Homomorphismen.

Eine Abstraktion entspricht einer Vereinfachung des Systems. Werden bestimmte Buchstaben auf das leere Wort abgebildet, so werden die damit verbundenen Zustandsübergänge ignoriert und das System wird einfacher (Verbergen von Aktionen). Werden verschiedene Buchstaben auf denselben Buchstaben abgebildet, so können die damit verbundenen Aktionen in der Abstraktion nicht mehr unterschieden werden (Identifikation von Aktionen), was das System ebenfalls vereinfacht.

Also entspricht eine solche Abstraktion einem Verlust von Information über das System. Interessant sind nun solche Abstraktionen, aus deren Analyse Rückschlüsse auf das Originalsystem gezogen werden können. Man kann zeigen, daß schlichte Homomorphismen diese Schlussfolgerung zulassen.

Definition: *Ein Homomorphismus heißt schlicht auf der Sprache L, wenn gilt:*

$$\begin{aligned} \forall_{x \in L}\, \exists_{w \in h(x)^{-1}(h(L))} : \quad & w^{-1}(h(x^{-1}(L))) = (h(x)w)^{-1}(h(L)) \\ \textit{wobei}: \quad & w^{-1}(L) = \{y \in \Sigma^* | wy \in L\} \end{aligned}$$

Dabei bezeichnet $w^{-1}(L)$ die Menge aller möglichen Fortsetzungen von w um ein Wort der Sprache L zu erhalten.

Für reguläre Sprachen ist Schlichtheit von Homomorphismen entscheidbar, die Algorithmen dazu sind jedoch sehr komplex. Es gibt jedoch notwendige und hinreichende Bedingungen für Schlichtheit, die sich auf einem LTS-Graphen effizient testen lassen. Für größere Systeme ist es allerdings sehr aufwendig einen Graphen zum entsprechenden LTS zu konstruieren (da diese Graphen sehr schnell sehr groß werden). Für 'günstig' strukturierte Systeme lässt sich die Schlichtheit von Homomorphismen überprüfen, ohne das komplexe LTS des Originalsystems zu berechnen. Dazu bettet man die einzelnen Komponenten des Systems in einfachere Umgebungen ein, die dasselbe Schnittstellenverhalten wie die Umgebungen der Komponenten im Originalsystem aufweisen. Für die so erhaltenen Systeme (Komponenten in jeweils einfacher Umgebung) kann die Schlichtheit von Homomorphismen vergleichsweise leicht geprüft werden. Die so erhaltenen Homomorphismen für verschiedene disjunkte Komponenten lassen sich nun zu einem Homomorphismus für das Gesamtsystem kombinieren. Unter bestimmten Bedingungen für die einzelnen Homomorphismen (unter anderem: Schlichtheit) kann man zeigen, daß der bei Kombination entstehende Homomorphismus ebenfalls schlicht ist. Für 'geeignet' strukturierte Systeme lässt sich so eine beträchtliche Reduktion des Zustandsraumes erreichen. APA sind in diesem Sinne geeignet strukturiert. Sie bestehen bereits aus einzelnen Komponenten (den elementaren Automaten), die so kombiniert sind, daß sie direkt Kandidaten für die oben erwähnte Einbettung in einfache Umgebungen darstellen.

Das SHVT arbeitet halbautomatisch. Es benötigt Hilfe vom Benutzer, um geeignete Homomorphismen zu generieren. Dabei stellt es die Möglichkeit zur schnellen Überprüfung der Schlichtheit zur Verfügung. Ist Schlichtheit nicht erfüllt oder nicht nachzuweisen, kann mit Hilfe der Ausgaben des SHVT die problematische Stelle gefunden werden, bei der eine Verfeinerung des Homomorphismus nötig ist.

Das SHVT wurde in Allegro Common Lisp implementiert und ist für nichtkommerzielle Anwendungen frei erhältlich. Ein anonymer FTP-Download ist jedoch nicht möglich aufgrund von lizenzrechtlichen Einschränkungen der Laufzeitbibliothek des Lisp-Systems.

5 Zusammenfassung

Die Sicherheit kryptographischer Protokolle hängt oft von impliziten Annahmen oder informal gegebenen Informationen ab. In dieser Ausarbeitung wurde ein Angriff auf das Needham-Schroeder Protokoll präsentiert, welcher nur möglich ist, wenn die Agenten Zufallszahlen und Schlüssel nicht unterscheiden können und einer der Agenten die ankommenden Nachrichten nicht auf die Anzahl ihrer Komponenten überprüft. Der Angriff steht und fällt also mit Implementationsdetails, die nicht unbedingt in der Protokollspezifikation behandelt werden.

Bei der Modellierung kryptographischer Protokolle mit asynchronen Produktautomaten werden keine implizite Annahmen verwendet. Alle Annahmen auf denen die Sicherheit das Protokolls basiert müssen explizit spezifiziert werden. Am Beispiel des Needham-Schroeder Protokolls wurde die Modellierung durch APA demonstriert.

Zur Analyse des APA wird vom Simple Homomorphism Verification Tool eine Suche auf dem endlichen Zustandsraum durchgeführt. Das SHVT arbeitet dabei auf speziellen Abstraktionen des Originalsystems, um mit den im allgemeinen sehr großen Zustandsräumen fertig zu werden. Die dabei erreichbare Reduktion des Zustandsraumes ermöglicht dann die Analyse größerer praxisrelevanter Systeme bzw. komplexerer Protokolle. Wurde bei der Analyse des Protokolls ein unsicherer Zustand gefunden, so kann ein dorthin führender Pfad berechnet werden, welcher das genaue Problem aufzeigt.

Literatur

[1] S. Gürgens, P. Ochsenschläger, C. Rudolph: 'Role based specification and security analysis of cryptographic protocols using asynchronous product automata', GMD-Report 151, 2001

[2] P. Ochsenschläger, J. Repp, R. Rieke, U. Nitsche: 'The SH-Verification Tool - Abstraction-Based Verififcation of co-operating systems', 1998

[3] P.Ochsenschläger, J.Repp, R. Rieke 'The SH-Verification Tool - A Tutorial', SIT - Fraunhofer-Institut für sichere Telekooperation, 2002