Analyse kryptographischer Protokolle mittels Produktnetzen, basierend auf Modellannahmen der BAN-Logik

Diplomarbeit

vorgelegt beim Fachbereich Informatik (Fb. 20)
der Johann Wolfgang Goethe–Universität
in Frankfurt am Main

von
Carsten Rudolph

Betreuer: Prof. Dr. E. Raubold

Frankfurt 1996

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Frankfurt, den 17. Dezember 1996

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei der Ausarbeitung meiner Diplomarbeit unterstützt haben. Insbesondere bei Prof. Dr. E. Raubold für Aufgeschlossenheit und Diskussionsbereitschaft und bei Dr. Sigrid Gürgens für viele hilfreiche Gespräche über die Analyse kryptographischer Protokolle.

Zusammenfassung

In vielen Bereichen der Telekooperation ist es notwendig, daß zwischen verschiedenen Paaren von Teilnehmern (Menschen, Computern, Diensten, Chipkarten) Authentizität und Vertraulichkeit gesichert werden. Im allgemeinen begegnet man diesem Problem mit dem Einsatz kryptographischer Protokolle. Einige Protokolle dienen zum Beispiel zur Etablierung eines neuen Schlüssels zwischen zwei Partnern, die sich vorher jeweils von der Identität des anderen überzeugen. Obwohl die verwendeten kryptographischen Verfahren große Sicherheit bieten, gibt es noch viele Möglichkeiten, solche Protokolle zu attackieren. Dazu zählen zum Beispiel replay-attacks (Wiederholung alter Nachrichten) oder parallel-session-attacks (Starten eines parallelen Protokollaufs mit Austausch von Nachrichten). Daher ist es wichtig, kryptographische Protokolle hinsichtlich ihrer Sicherheit zu analysieren. Ziel dieser Arbeit ist es, die Möglichkeiten zur Verwendung der Produktnetzmaschine als Analysewerkzeug für Authentifikationsprotokolle zu untersuchen.
Um diese komplexe Problemstellung einzuschränken, wird die Wirkungsweise von Protokollen unter den Modellannahmen der BAN-Logik betrachtet. Diese von Burrows, Abadi und Needham eingeführte Logik bietet einen guten Ansatz zum Verständnis der Problematik und deckt einen Teil der möglichen Angriffe ab. Die als Produktnetze implementierten Methoden sollen mit geringem Aufwand an erweiterte Modelle angepaßt werden können.
Es werden zwei Möglichkeiten vorgestellt, mit der Produktnetzmaschine Aussagen über Authentifikationsprotokolle zu treffen, erstens durch direkte Modellierung eines möglichen Angreifers und zweitens durch Implementation einer “Bottom-Up” Beweismethode für die BAN-Logik.

Inhaltsverzeichnis

1 Einleitung

Durch die fortschreitende Entwicklung leistungsfähiger verteilter Computersysteme und Telekooperationsanwendung entstehen immer höhere Anforderungen an die Informationssicherheit. Zu sichern ist dabei die Identität der Partner (Authentizität), Schutz der Daten gegen unbefugten Zugriff (Vertraulichkeit) und Schutz der Daten gegen unbemerkte Veränderung (Integrität). Außer Computernetzen sind besonders Chipkarten und Anwendungen der Telekommunikation, wie Mobiltelefone, betroffen. Das gebräuchlichste Mittel um diese Sicherheitsproblematik zu behandeln, ist die Kryptographie. Durch Verschlüsselung wird Vertraulichkeit gesichert und durch digitale Unterschriften wird authentifizierter Datenaustausch möglich. Es stehen verschiedenen Algorithmen zur Verfügung, die aus heutiger Sicht den Sicherheitsanforderungen genügen. Aber bei der Verwendung dieser Algorithmen in Kommunikationsprotokollen können gravierende Fehler gemacht werden, die trotz der sicheren Verschlüsselung Angriffe ermöglichen. Das grundlegende Problem, mit dem sich Protokollentwickler auseinandersetzen müssen, ist die Existenz von Betrügern, die beliebig in die Kommunikation eingreifen können. Die Vielfalt möglicher Angriffe ist außerordentlich groß. Aus diesem Grund ist es notwendig, leistungsfähige formale Methoden zur Verifikation von Authentifikationsprotokollen zu besitzen. Eine Methode, mit der einige Protokollfehler gefunden wurden, ist die von Burrows, Needham und Abadi in [4] eingeführte modale Logik, üblicherweise BAN-Logik genannt. Ausgehend von den Modellvorstellungen, die der BAN-Logik zu Grunde liegen, werden in dieser Arbeit Möglichkeiten der maschinellen Unterstützung der Verifikation von kryptographischen Protokollen vorgestellt. Verwendet wird dafür das in der GMD[1] entwickelte Werkzeug Produktnetzmaschine.
Nach einer kurzen Beschreibung der Problemstellung gibt Kapitel 3 eine Einführung in die BAN-Logik. Dabei werden auch die Grenzen dieser Methode erläutert.
In Kapitel 4 werden zwei Ansätze zur Verwendung von Produktnetzen in der Analyse von kryptographischen Protokollen vorgestellt. Der erste Ansatz macht keinen Gebrauch von der BAN-Logik ansich, bezieht sich aber auf das eingeschränkte Modell. Ein als Produktnetz modellierter Angreifer führt seine Attacken an einem Modell des zu analysierenden Protokolls durch. Mit Hilfe der Produktnetzmaschine wird der Erreichbarkeitsgraph des Systems aus Angreifer und regulären Teilnehmern berechnet. Ein möglicher Angriff kann durch Prüfung aller möglichen Systemzustände gefunden werden.
Der zweite Ansatz ist die Implementation einer Beweismethode für die BAN-Logik. Dafür wurde die Tableaumethode aus [5] derart abgeändert, daß sie deterministisch für die BAN-Logik anwendbar wird.

[1] GMD – Forschungszentrum Informationstechnik GmbH

2 Aspekte der Analyse kryptographischer Protokolle

Protokolle, in denen kryptographische Verfahren angewandt werden, bestehen im allgemeinen aus wenigen Nachrichten. Die üblicherweise zur Verschlüsselung verwendeten Algorithmen sind mittlerweile ausreichend analysiert, um beim Protokolldesign die Stärken und Schwächen berücksichtigen zu können. Fragen nach der Sicherheit der Verschlüsselungsalgorithmen werden deshalb hier nicht betrachtet. Es wird angenommen, daß die verwendeten Algorithmen ausreichend sicher sind. Über die Qualität der verwendeten Schlüssel müssen ebenfalls einige Annahmen getroffen werden. Zum Beispiel:

- Schlüssel können nicht "erraten" oder berechnet werden.
- Mit einem Schlüssel K verschlüsselter Text kann mit keinem Schlüssel $\bar{K}$ mit $\bar{K} \neq K$ (bzw. $\bar{K} \neq K^{-1}$ im unsymmetrischen Fall) entschlüsselt werden.

Diese Annahmen müssen selbstverständlich im Einzelfall bei der Analyse realer Protokolle verifiziert werden. Doch auch bei Verwendung sehr starker Algorithmen werden immer wieder Angriffe gegen scheinbar sichere Protokolle gefunden. Das grundlegende Problem ist die Existenz von Betrügern, die beliebig in die Kommunikation eingreifen können. Ein gegen Manipulation gesicherter Kanal ist in den meisten Anwendungen nicht vorhanden. Um Aussagen zur Sicherheit von Protokollen zu treffen, muß man vorher die Möglichkeiten eines Angreifers analysieren. Es ist aber außerordentlich schwierig, das Verhalten eines Betrügers umfassend zu simulieren. Einen Überblick, wie vielfältig die Fehlermöglichkeiten in kryptographischen Protokollen sind, bietet [2]. Alte Nachrichten können wiederholt, oder Nachrichtenteile können ausgetauscht werden. Teilnehmer werden veranlaßt, Dinge zu unterschreiben, die sie nicht kennen. Über parallel gestartete Protokolläufe können Teilnehmer als Orakel benutzt werden.
Ein weiterer Aspekt der Protokollanalyse ist der Aufwand, der zum Erreichen eines Sicherheitszieles notwendig ist. In der Praxis muß oft die Sicherheit mit der Schnelligkeit und dem Rechenaufwand abgewogen werden. Dies gilt insbesondere dann, wenn nur beschränkte Kapazitäten zur Verfügung stehen, wie zum Beispiel bei Chipkarten. Die Fragen, die die Analyse kryptographischer Protokolle beantworten sollte, sind zum Beispiel folgende:

Welches Sicherheitsziel erreicht ein Protokoll?

Werden Nachrichten generiert, die weggelassen werden können, ohne das Protokoll zu schwächen?

Benötigt ein Protokoll stärkere Voraussetzungen (z.B. synchrone Uhren) als ein anderes?

Werden Daten verschlüsselt, die ohne Schwächung des Protokolls auch unverschlüsselt bleiben können?

2.1 Modell des Kommunikationssystems

Authentifizierungsprotokolle werden hier in einem System aus einzelnen Teilnehmern und Kommunikationskanälen zwischen den Teilnehmern betrachtet. Diese Kanäle bieten die einzige Möglichkeit der Kommunikation. Um möglichst alle bekannten Angriffsarten im Modell zu berücksichtigen, sollte der Angreifer sehr mächtig sein, während die regulären Teilnehmer wenig Kontrolle über die Übertragungskanäle besitzen. Auf seinem Kanal kann jeder Teilnehmer beliebig senden oder empfangen, aber ein Angreifer kann den gesamten Verkehr auf dem Kommunikationssystem umfassend kontrollieren. Dabei bedeutet umfassende Kontrolle, daß der Angreifer beliebig Nachrichten lesen, entfernen oder einspeisen kann.

Ein Protokoll ist eine Vorschrift zur Kommunikation. Es schreibt die Aktionen und den Nachrichtenaustausch in Abhängigkeit von den inneren Zuständen der Teilnehmer und der empfangenen Nachrichten vor. Mit einem Protokollauf wird eine bestimmte Ausführung des Protokolls bezeichnet.
Das Ziel eines Protokolls könnte zum Beispiel der Aufbau eines sicheren Kanals sein, der authentische und vertrauliche Datenübertragung zwischen zwei Teilnehmern ermöglicht. Um dieses Ziel zu erreichen, stehen den Teilnehmern ausschließlich die unsicheren, vom Angreifer manipulierbaren Übertragungswege zur Verfügung.
Manche Protokolle setzen die Existenz eines ausgezeichneten Teilnehmers voraus, wie zum Beispiel eine certification-authority bei public-key Verschlüsselung oder einen keyserver für die Vergabe neuer Schlüssel. Abbildung 1 zeigt ein System aus zwei Teilnehmern A und B, einem keyserver S und dem Angreifer C, der beliebig auf den Kanal Einfluß nehmen kann.
Um Aussagen über kryptographische Protokolle treffen zu können, ist es in jedem Modell notwendig, gegenseitiges Vertrauen vorauszusetzen. Kein Teilnehmer darf seine eigenen Geheimnisse (z.B. private Schlüssel für Unterschriften) für andere zugänglich machen. Könnten sich in diesem Punkt die Teilnehmer nicht vertrauen, wären die Ziele kryptographischer Protokolle nicht erreichbar.

3 Einführung in die BAN-Logik

Bei der BAN-Logik handelt es sich um eine *logic of belief*, die von M. Burrows, M. Abadi und R. Needham entwickelt und in [4] beschrieben wurde. Als Ziel von Authentifikationsprotokollen wird die Erlangung neuen “Glaubens” durch die Teilnehmer gesehen. “Glauben” bedeutet dabei, etwas auf Grund bestimmter Informationen und Annahmen für wahr zu halten. Die Protokollteilnehmer können eventuell nach einem Protokollauf an die Sicherheit eines neuen Schlüssel glauben oder sicher bezüglich der Identität ihres Partners sein. In der Logik werden die Konzepte, die zu einem Zugewinn an Glauben führen, formalisiert. Der Informationsgehalt eines Protokollschrittes wird aus Sicht des jeweiligen Adressaten bewertet. Diese Sicht auf die Wirkungsweise von Authentifikationsprotokollen führt zu einem guten Verständnis der häufigsten Feh-

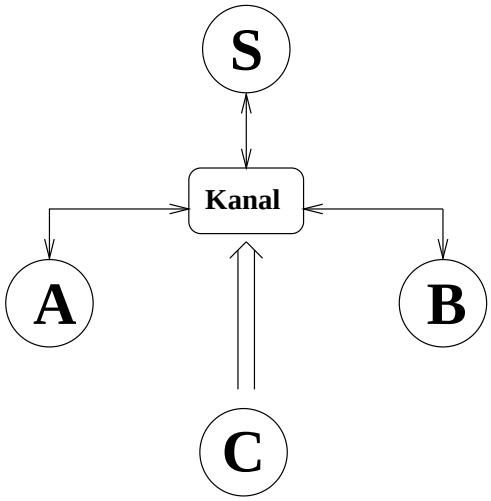


Abbildung 1: Modell des Kommunikationssystems

lerquellen.
In diesem Kapitel werden informal die Syntax, Semantik und Ableitungsregeln der BAN-Logik beschrieben und die Grenzen und Schwierigkeiten dieser Methode erläutert.

3.1 Zusätzliche Modellannahmen

Durch kleine Einschränkungen des in Kapitel 2.1 beschriebenen Modells erhält man eine sehr effektive formale Methode.

Eine grundlegende Einschränkung bildet das in Abbildung 2 dargestellte Zeitmodell der BAN-Logik. Ein neuer Protokollauf wird erst gestartet, wenn der vorherige abgeschlossen ist. Es werden also nur zwei Zeitebenen unterschieden: der aktuelle Lauf und die Vergangenheit. Ein Angreifer kann Nachrichten aus der Vergangenheit verwenden, während die Teilnehmer des aktuellen Laufes keine Kenntnisse über die vorherigen Läufe besitzen. In der Realität gilt, daß jeder Teilnehmer nur ein beschränktes Gedächtnis besitzt, das vom Angreifer übertroffen werden kann. Deshalb muß man davon ausgehen, daß der Angreifer noch Protokolläufe aus der Vergangenheit gespeichert hat, für die im Speicher des regulären Teilnehmers kein Platz mehr ist. Dieser Sachverhalt wird durch die Maximalforderung, daß die Teilnehmer schon Nachrichten aus dem letzten Lauf nicht mehr erkennen, simuliert.
Weiter wird angenommen, daß jeder Teilnehmer Nachrichten die er selbst generiert hat, von fremden Nachrichten unterscheiden kann. Selbst generierte Nachrichten werden nicht akzeptiert, wenn sie während eines Angriffs wiederholt werden. Dies gilt auch für Nachrichten aus alten Protokolläufen.

Zu Beginn eines Protokollaufs gelten bestimmte Initialaussagen. Teilnehmer glauben an die Gültigkeit einer Zeitmarke oder an die Sicherheit eines verwendeten

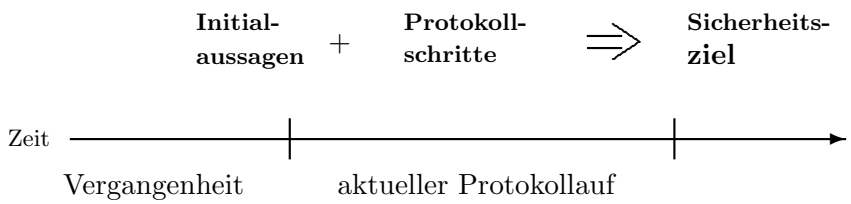


Abbildung 2: Zeitmodell der BAN-Logik

Schlüssels. Mit den Regeln und den Protokollschritten können die Teilnehmer neuen Glauben gewinnen, jedoch während eines Protokollaufs keinen Glauben verwerfen. Mit Hilfe der BAN-Logik Regeln können die erreichten Sicherheitsziele aus den Initialaussagen und dem idealisierten Protokoll abgeleitet werden. Ein Beweis der BAN-Logik kann auch als eine Reduktion des Sicherheitsziels auf die Initialaussagen verstanden werden.

Um auf ein Protokoll eine formale Methode anwenden zu können, ist es notwendig, das Protokoll in einer geeigneten Sprache darzustellen. Bei BAN werden Nachrichten in Aussagen der Logik transformiert. Diese Aussagen stellen ein idealisiertes Protokoll dar. Dieser Vorgang erfolgt zwar nach bestimmten Regeln, bleibt aber an sich informal. Klartext wird ganz weggelassen, da er nach der Vorstellung von BAN keinen Einfluß auf die Sicherheit hat. Diese Annahme liegt im Modell begründet, denn kein Teilnehmer verrät seine Geheimnisse. Da Protokolle im allgemeinen nicht einheitlich beschrieben werden, muß die Idealisierung informal bleiben. Allerdings liegt hier eine Fehlerquelle in der Beweisführung mit der BAN-Logik. Es besteht die Gefahr, daß bei der Transformation des Protokolls zusätzliche sicherheitsrelevante Informationen in die Protokollbeschreibung aufgenommen werden, die in den realen Protokollschritten nicht enthalten sind. Dazu wird in Kapitel 3.4 ein Beispiel angegeben.

3.2 Notation

In der BAN-Logik wird zwischen verschiedenen Objekten unterschieden: Teilnehmern, Schlüsseln und Formeln. Nachrichten des Protokolls werden wie Aussagen der Logik behandelt.
Zur Beschreibung der Aussagen stehen folgende Konstrukte zur Verfügung:

$\boldsymbol{A \mid\equiv X}$	"A glaubt X", bzw. A hat ein begründetes Vertrauen in X und handelt so, als wäre X wahr.
$\boldsymbol{S \Longrightarrow K}$	S hat Jurisdiktion über K. S ist eine Authorität bezüglich K. Teilnehmer können mittels authentischer Kommunikation mit S Vertrauen in neue Schlüssel gewinnen, oder ein Zertifikat akzeptieren.
$\boldsymbol{A \mid\sim X}$	A hat irgendwann (entweder im aktuellen Lauf, oder in der Vergangenheit) X gesagt.

$\sharp X$	X ist "frisch", d.h. X ist vor dem aktuellen Lauf nie verwendet worden.
$A \overset{K_{AB}}{\longleftrightarrow} B$	K_{AB} ist ein sicherer symmetrischer Schüssel für die Kommunikation von A und B.
$\overset{K}{\longmapsto} A$	A besitzt K als öffentlichen Schlüssel. Der korrespondierende geheime Schlüssel K^{-1} ist (außer bestimmten authorisierten Teilnehmern) nur A bekannt.
$A \triangleleft X$	A "sieht" X. Diese scheinbar problemlose Aussage kann doch verschiedene Interpretationen besitzen. Bei der BAN-Logik wird nämlich nicht berücksichtigt, ob X von A erkannt wird, oder ob X nur ein nicht überprüfbarer Bitstring ist, dem die Bedeutung entsprechend der Protokollvorschrift zugeordnet wird.
$A \overset{X}{\rightleftharpoons} B$	X ist ein von A und B geteiltes Geheimnis und kann in verschlüsselten Nachrichten zur Authentifizierung dienen.
$\{X; K\}$	Die Formel X ist mit dem Schlüssel K verschlüsselt.
$X.Y$	Konkatenation von Nachrichtenteilen

Zur Beschreibung der Protokolle werden außerdem folgende Bezeichnungen verwendet:

A, B	Reguläre Protokollteilnehmer
P, Q	Teilnehmer in allgemeinen Beschreibungen
S	authentication server
C	Angreifer
N, N_A, N_B	Im aktuellen Protokollauf erstmals verwendete Daten, z. B. Zufallszahlen, die jeweils von einem Teilnehmer für "frisch" gehalten werden können. In Anlehnung an die englischsprachige Literatur werden solche Konstrukte mit *nonce* bezeichnet.

3.3 Ableitungsregeln

Das Konzept der BAN-Logik baut auf einigen grundlegenden Regeln auf. In diesem Abschnitt werden die wichtigsten Ableitungsregeln für den Fall symmetrischer Verschlüsselungsalgorithmen angegeben. Für asymmetrische Verfahren oder geteilte Geheimnisse gibt es entsprechend abgeänderte Regeln.

- message-meaning rule

$$\frac{\boldsymbol{P} \mid\equiv \boldsymbol{Q} \overset{K}{\longleftrightarrow} \boldsymbol{P}, \boldsymbol{P} \triangleleft \{\boldsymbol{X}; \boldsymbol{K}\}}{\boldsymbol{P} \mid\equiv \boldsymbol{Q} \mid\sim \boldsymbol{X}}$$

  Die message-meaning rule ermöglicht Aussagen über den Erzeuger einer verschlüsselten Nachricht abzuleiten. Glaubt P an die Sicherheit des Schlüssels K und empfängt er eine Nachricht verschlüsselt mit K, kann die Nachricht nur von ihm selbst oder dem Partner generiert worden sein. Da in der BAN-Logik angenommen wird, daß selbst generierte Nachrichten immer wiedererkannt werden, kann P schließen, daß Q die Nachricht irgendwann gesagt hat. Im nichtsymmetrischen Fall gilt die Regel entsprechend mit dem privaten Schlüssel. Die Aussage $Q \mid\sim X$ sagt nichts über den Zeitpunkt, zu dem X von Q generiert und gesagt wurde. Um sicher zu sein, daß es sich nicht um eine wiederholte Nachricht aus alten Protokolläufen handelt, ist die folgende Regel nötig:

- nonce-verification rule

$$\frac{\boldsymbol{P} \mid\equiv \sharp(\boldsymbol{X}), \boldsymbol{P} \mid\equiv \boldsymbol{Q} \mid\sim \boldsymbol{X}}{\boldsymbol{P} \mid\equiv \boldsymbol{Q} \mid\equiv \boldsymbol{X}}$$

  Glaubt P, daß Q irgendwann X gesagt hat, und glaubt P außerdem, daß X vor dem aktuellen Lauf nie verwendet wurde, kann P glauben, daß Q die Aussage X nicht irgendwann, sondern im aktuellen Lauf gesagt hat. Basierend auf der Annahme gegenseitigen Vertrauens kann geschlossen werden, daß Q an X glaubt.

- jurisdiction rule

$$\frac{\boldsymbol{P} \mid\equiv \boldsymbol{Q} \Longrightarrow \boldsymbol{X}, \boldsymbol{P} \mid\equiv \boldsymbol{Q} \mid\equiv \boldsymbol{X}}{\boldsymbol{P} \mid\equiv \boldsymbol{X}}$$

  Glaubt P, daß Q bezüglich X Jurisdiktion besitzt, also eine vertrauenswürdige Autorität darstellt, und glaubt P, daß Q im Moment selbst an die Gültigkeit von X glaubt, dann hat P Grund, auch an die Gültigkeit von X zu glauben. X kann dabei z.B. eine Aussage über die Sicherheit eines Schlüssels sein.

Neben diesen drei Regeln gibt es noch eine Reihe weiterer Regeln in der Art von:

- belief rules

$$\frac{\boldsymbol{P} \mid\equiv (\boldsymbol{X}.\boldsymbol{Y})}{\boldsymbol{P} \mid\equiv \boldsymbol{X}} \quad \text{und} \quad \frac{\boldsymbol{P} \mid\equiv \boldsymbol{X} \wedge \boldsymbol{P} \mid\equiv \boldsymbol{Y}}{\boldsymbol{P} \mid\equiv (\boldsymbol{X}.\boldsymbol{Y})}$$

Die Konkatenation der Aussagen X und Y impliziert bezüglich des Wahrheitsgehalts keine semantische Verknüpfung. Sie können deshalb sowohl einzeln als auch getrennt betrachtet werden.

- freshness rule

$$\frac{\boldsymbol{P} \mid\equiv \sharp(\boldsymbol{X})}{\boldsymbol{P} \mid\equiv \sharp(\boldsymbol{X}.\boldsymbol{Y})}$$

Ist in einer Nachricht $(X.Y)$ ein Teil X enthalten, von dem P glaubt, daß er in alten Protokolläufen noch nicht verwendet worden ist, dann kann die konkatenierte Aussage nur im aktuellen Protokollauf zum ersten Mal gesagt worden sein. Über die Frische von Y läßt sich keine Aussage ableiten.

3.4 Bewertung der BAN-Logik

Durch die eingeschränkten Modellannahmen wird die BAN-Logik zu einem übersichtlichen und leicht anzuwendenden Werkzeug. Prinzipien, auf denen viele Protokolle beruhen, werden formalisiert. So können einige grundlegende Fehler bei der Konstruktion neuer Protokolle vermieden werden. Die Pionierarbeit, die Burrows, Needham und Abadi mit ihrer BAN-Logik geleistet haben, ist sehr hoch zu bewerten. Allerdings zeigte sich schnell, daß die BAN-Logik als alleinige Analysemethode nicht ausreicht. Es werden viele Schwächen gefunden, andere bleiben aber unentdeckt. Findet man mit Hilfe der Logik eine Schwachstelle, dann ist das Protokoll mit Sicherheit angreifbar. Kann man das gewünschte Sicherheitsziel ableiten, bedeutet das nicht, daß keine Angriffe möglich sind. Dafür gibt es mehrere Gründe, die im folgenden kurz dargestellt werden.
Offensichtlich entsprechen einige der Modellannahmen nicht den Eigenschaften realer Systeme. Zum Beispiel beruhen einige bekannte Angriffe auf parallelen Protokolläufen. Diese können im Zeitmodell der BAN-Logik nicht erfaßt werden. Ein Beispiel dafür ist die in [12] vorgestellte Attacke auf das Neumann-Stubblebine Authentifikationsprotokoll. Dabei benutzt der Angreifer einen regulären Teilnehmer während einer parallelen Session als Orakel, um sich im eigentlichen Lauf als Partner maskieren zu können. Die Annahme, daß selbstgenerierte Nachrichten immer wiedererkannt werden, trifft für reale Systemen oft nicht zu. Auch bei Verwendung digitaler Unterschriften sind zusätzliche Maßnahmen notwendig, um die Erkennung selbstgenerierter Nachrichten zu gewährleisten. Unterschriebene Nachrichten müßten noch verschlüsselt werden, um eine Entfernung der Unterschrift zu verhindern. Manche Protokolle sind nur dann sicher, wenn ausschließlich von anderen Teilnehmern generierte Nachrichten akzeptiert werden. Der oben erwähnte Angriff auf das Neumann-Stubblebine Protokoll ist z.B. nur erfolgreich, wenn der attackierte Teilnehmer selbstgenerierte Nachrichten als fremde akzeptiert. Auch der zweite, “paradox-attack” genannte Angriff in [12] basiert auf dem Zurücksenden einer Nachricht an den Erzeuger.

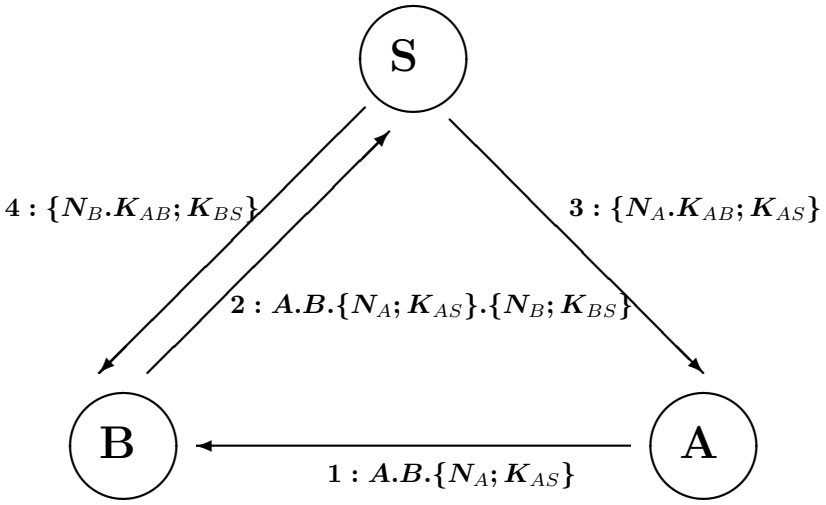


Abbildung 3: Protokoll zur Demonstration einer BAN-Logik Schwäche

Diese Schwächen zeigen die Grenzen des eingeschränkten Modells, und es existieren einige Ansätze für erweiterte Logiken. Zum Beispiel wurde in [6] eine Logik vorgestellt, in der Erkennbarkeit von Daten und Herkunft formalisiert werden.
Unabhängig von den Modellannahmen ist das schon oben erwähnte Problem der Erstellung eines idealisierten Protokolls.
Um diese Problematik zu verdeutlichen, habe ich ein kurzes Protokoll erstellt, das die Vereinbarung eines neuen Schlüssels zwischen zwei Teilnehmern mit Hilfe eines keyservers ermöglicht. Mit der BAN-Logik findet man keine Protokollschwäche, an der ein Angriff ansetzen könnte. Beide Teilnehmer glauben, daß der neue Schlüssel sicher ist und außer den regulären Teilnehmern niemandem bekannt ist. Dieses Ziel wird aber nicht erreicht, weil sich ein Angreifer als einer der Teilnehmer ausgeben kann. Das Protokoll besteht aus folgenden vier Schritten (siehe auch Abb.3):

1: $A \rightarrow B : A.B.\{N_A; K_{AS}\}$

2: $B \rightarrow S : A.B.\{N_A; K_{AS}\}.\{N_B; K_{BS}\}$

3: $S \rightarrow A : \{N_A.K_{AB}; K_{AS}\}$

4: $S \rightarrow B : \{N_B.K_{AB}; K_{BS}\}$

Damit sich ein betrügerischer Teilnehmer C als B ausgeben kann, muß er nur die zweite Nachricht gegen

2: $C \rightarrow S : A.C.\{N_A; K_{AS}\}.\{N_C; K_{CS}\}$

austauschen. Der keyserver S stellt einen Schlüssel für A und C bereit. Die Nachrichten 3 und 4 werden zu:

3: $S \rightarrow A : \{N_A.K_{AC}; K_{AS}\}$

4: $S \rightarrow C : \{N_B.K_{AC}; K_{BS}\}$

Für A ändert sich nichts. Die Schlüssel K_{AB} und K_{AC} können von A nicht unterschieden werden. A würde den neuen Schlüssel akzeptieren und zur Kommunikation mit B verwenden. C könnte sich gegenüber A als B ausgeben.
Warum findet man mit der BAN-Logik diesen Angriff nicht? Um diese Frage zu beantworten, muß man zuerst den Beweis betrachten, wie man ihn mit der Logik führen würde. Als erster Schritt wird das idealisierte Protokoll erstellt. Klartext wird ignoriert und für Schlüssel, die als Daten auftauchen, wird die Aussage "ist ein sicherer Schlüssel für..." eingesetzt. Damit erhält man folgendes idealisierte Protokoll:

1: $A \rightarrow B : \{N_A; K_{AS}\}$

2: $B \rightarrow S : \{N_A; K_{AS}\}.\{N_B; K_{BS}\}$

3: $S \rightarrow A : \{N_A.A \overset{K_{AB}}{\longleftrightarrow} B; K_{AS}\}$

4: $S \rightarrow B : \{N_B.A \overset{K_{AB}}{\longleftrightarrow} B; K_{BS}\}$

Nun wird versucht, die Aussage $A \models A \overset{K_{AB}}{\longleftrightarrow} B$ zu beweisen. Dazu benötigt man einige Aussagen, die vor Beginn des Protokollaufs erfüllt sein müssen:

(i)	$A \models A \overset{K_{AS}}{\longleftrightarrow} S$	A teilt den sicheren Schlüssel K_{AS} mit S
(ii)	$A \models S \Longrightarrow A \overset{K}{\longleftrightarrow} B$	A vertraut dem keyserver S bezüglich der Qualität von Schlüsseln
(iii)	$A \models \sharp N_A$	A vertraut der Frische seiner eigenen nonce

Diese Annahmen sind offensichtlich sinnvoll und können in einem realen System garantiert werden.
A sieht die dritte Nachricht des Protokollaufs. A kann nur diese Nachricht verwenden, um neue Erkenntnisse zu erlangen. Wir haben also

$$A \triangleleft \{N_A.A \overset{K_{AB}}{\longleftrightarrow} B; K_{AS}\}$$

Zusammen mit (i) ergibt die message-meaning rule $A \models S \mid\sim N_A.A \overset{K_{AB}}{\longleftrightarrow} B$. Anwendung der freshness- und nonce-verification rule führt mit Annahme (iii) zu
$A \models S \models N_A.A \overset{K_{AB}}{\longleftrightarrow} B$. Dies impliziert $A \models S \models A \overset{K_{AB}}{\longleftrightarrow} B$. Anwendung der jurisdiction-rule mit Annahme (ii) bringt das gewünschte Ergebnis:

$$A \models A \overset{K_{AB}}{\longleftrightarrow} B$$

Das Protokoll scheint sein Ziel zu erreichen. Aber der oben gezeigte Angriff beweist das Gegenteil. Der Grund für das Scheitern der BAN-Logik an diesem Beispiel ist in der Transformation des Protokolls zu suchen. In der dritten Nachricht wird durch Einfügen von $A \overset{K_{AB}}{\longleftrightarrow} B$ anstelle von K_{AB} ein Zusammenhang zwischen der zweiten und

dritten Nachricht hergestellt, der im Originalprotokoll nicht existiert. Man unterstellt dabei, A könnte an der Nachricht des keyservers erkennen, für welche zwei Teilnehmer der Schlüssel gedacht ist. Beschränkt man sich auf die Beschreibungsmöglichkeiten der BAN-Logik, lassen sich solche Fehler nicht ausschließen, denn man müßte ein Protokoll analysieren, bevor man eine korrekte idealisierte Version erstellen kann.
Um diese Probleme zu vermeiden, benötigt man eine Sprache, mit der man Protokolle genau spezifizieren kann, ohne Interpretationen der Nachrichten einfügen zu müssen. Dabei müßte auch das Verhalten der einzelnen Teilnehmer in die Beschreibung aufgenommen werden. Dazu gehört insbesondere die Information, welche Nachrichtenteile auf Gültigkeit geprüft werden. Würde man z.B. im obigen Protokoll die dritte Nachricht gegen

3: $S \rightarrow A : \{B.N_A.A \overset{K_{AB}}{\longleftrightarrow} B; K_{AS}\}$

austauschen, hinge die Sicherheit des Protokolles davon ab, ob die Identität des angegebenen Teilnehmers B von A überprüft wird. In diesem Beispiel scheint es selbstverständlich zu sein, daß A sowohl die nonce N_A als auch die Identität von B prüft, aber man findet oft Protokollbeschreibungen, in denen solche "selbstverständlichen" Informationen verschwiegen werden, obwohl die Sicherheit des Protokolles davon abhängt. Bisher gibt es noch keine Methode, die sich für den allgemeinen Gebrauch durchgesetzt hat. Wahrscheinlich wird aber für komplexe Beweise maschinelle Unterstützung notwendig werden. Am Beispiel der Produktnetzmaschine wird nun beschrieben, wie ein schon existierendes Verifikationswerkzeug zur Unterstützung der Protokollanalyse verwendet werden kann.

4 Einsatzmöglichkeiten für die Produktnetzmaschine

Die Produktnetzmaschine ist ein Werkzeug zur formalen Spezifikation und Verifikation von verteilten Systemen. Produktnetze sind beschriftete Petrinetze[2] mit individuellen Marken und verschiedenen Kantentypen. Neben den "normalen" Kanten, die das Schalten einer Transition abhängig vom Vorhandensein einer Marke ermöglichen, ist die Verwendung von Verbots- und Abräumkanten erlaubt. Ohne diese verschiedenen Kantentypen lassen sich viele einfache Vorgänge nur mit komplizierten Konstruktionen modellieren. Alle Kanten tragen eine Kantenanschrift. Dabei können Variablen, Konstanten und Funktionen verwendet werden. Transitionen können Transitionsinschriften enthalten, die zusätzliche Bedingungen für die Aktiviertheit darstellen. Jeder Stelle wird ein eigener Definitionsbereich zugeordnet. Die dabei verwendeten Mengen werden, wie die Funktionen für Kantenanschriften und Transitionsinschriften, im Vorspann definiert. Der Definitionsbereich einer Stelle kann das Produkt verschiedener Mengen sein. Daher der Name *Produktnetz*. Mit der Anfangsmarkierung wird der

[2]Petrinetze sind ein mathematisches Modell zur Beschreibung und Analyse von Abläufen mit nebenläufigen Prozessen und nichtdeterministischen Vorgängen. Dieses Modell wurde 1962 von C.A.Petri [10] vorgeschlagen.

Startzustand vorgegeben. Durch Schalten aktivierter Transitionen werden Markierungen in Folgemarkierungen überführt. Entsprechend der Kanten und der Interpretation der Variablen in den Kantenanschriften werden dazu Marken aus Stellen entfernt bzw. hineingelegt. Nebenläufiges Schalten mehrerer Transitionen wird durch deterministisches Berechnen aller möglichen Schaltfolgen simuliert. Echte Nebenläufigkeit wird nicht betrachtet (Interleaving Konzept).
Um diese Netze in einem Werkzeug zu verwenden, muß die Überprüfung der Aktiviertheit und das Schalten einer Transition algorithmisch durchführbar sein. Daher gibt es Einschränkung für die erlaubten Mengen und Funktionen. Die mächtigsten erlaubten Funktionen sind primitiv-rekursive Funktionen. Diese Einschränkungen haben keine Auswirkung auf die Ausdrucksstärke der Netze, denn schon die Verwendung von Verbotskanten macht Petri-Netze in Bezug auf die Ausdrucksstärke äquivalent zu Turingmaschinen, mit denen sich alle bekannten algorithmischen Vorgänge darstellen lassen [7].
Netze können mit einem Editor gezeichnet und beschriftet werden und komplett oder in Teilen auf syntaktische Korrektheit überprüft werden. In einer speziellen Vorspannsprache werden die oben erwähnten Mengen und primitiv-rekursiven Funktionen im Vorspann angegeben. Unter Anwendung der Schaltregel kann der Erreichbarkeitsgraph (Menge aller möglichen erreichbaren Systemzustände mit ihren Übergängen) als Basis für die Analyse des dynamischen Verhaltens eines spezifizierten Systems berechnet werden. Zur Auswertung sehr großer Erreichbarkeitsgraphen bietet die Produktnetzmaschine verschiedene Möglichkeiten. Neben Programmen zum Suchen von Markierungen mit bestimmten Eigenschaften oder zur Bestimmung von Schaltfolgen können mit dem Homomorphismuseditor Homomorphismen auf Schaltfolgensprachen definiert werden. Um neben Sicherheits- auch Lebendigkeitseigenschaften überprüfen zu können, stehen spezielle Graphenalgorithmen zur Bestimmung der Schlichtheit [3] von Homomorphismen zur Verfügung. Eine komplette Einführung in die Verifikation mit Produktnetzen findet man in [9].
In Kapitel 2.1 wurden die Fragen der Verifikation kryptographischer Protokolle erläutert. Lassen sich diese Fragen mit einem Verifikationstool für verteilte Systeme beantworten?
Eigenschaften von Systemen kann man klassifizieren in Lebendigkeitseigenschaften und Sicherheitseigenschaften [1]. Unter einer Sicherheitseigenschaft versteht man, daß ein "schlechtes" Ereignis zu keiner Zeit eintritt. Eine Lebendigkeitseigenschaft bedeutet, daß ein bestimmtes Verhalten oder ein bestimmter Zustand immer wieder erreicht werden kann. Die kritischen Eigenschaften kryptographischer Protokolle lassen sich als Invarianzargumente beschreiben. Ein solches Invarianzargument ist zum Beispiel:

Es gilt immer: "Der aktuelle session key ist außer den regulären Teilnehmern niemandem bekannt."

[3] Lebendigkeitseigenschaften beziehen sich auf die möglichen Fortsetzungen von Schaltfolgen. Um aus Lebendigkeitseigenschaften des homomorphen Bildes der Schaltfolgensprache Aussagen über die Eigenschaften der gesamten Sprache treffen zu können, ist die Schlichtheit des Homomorphismus nötig.

Nach der obigen Klassifizierung handelt es sich hier um eine Sicherheitseigenschaft. Lebendigkeitseigenschaften können bei der Verifikation kryptographischer Protokolle vernachlässigt werden bzw. müssen im System, in dem das Protokoll verwendet wird, abgesichert werden. Ein Angreifer, wie er in Kapitel 2.1 beschrieben wird, kann z.B. beliebig oft eine erfolgreiche Durchführen eines Protokolles sabotieren. In der Praxis muß deshalb gesichert werden, daß das Protokoll immer wieder neu gestartet werden kann. Solche Verwaltungsaufgaben werden aber nicht vom Protokoll selber durchgeführt, sondern von der Umgebung, in der das Protokoll verwendet wird.
Die wichtigsten Fragen an kryptographische Protokolle lassen sich in der für die Verifikation verteilter Systeme üblichen Weise ausdrücken. Es sollte also möglich sein, ein leistungsfähiges Verifikationstool wie die Produktnetzmaschine auf diesem Gebiet einzusetzen. Dafür werden nun zwei mögliche Ansätze vorgestellt.

Die bei einem Verifikationstool für verteilte Systeme naheliegende Vorgehensweise ist die direkte Modellierung des Systems aus regulären Teilnehmern und einem möglichst mächtigen Angreifer. Als problematisch erweist sich dabei die Komplexität des Modells.
Der zweite Ansatz verwendet die Produktnetzmaschine als Basis zur Implementation einer Beweismethode auf Basis der BAN-Logik.

4.1 Der universelle Angreifer

Die Idee ist, einen lernenden Angreifer zu modellieren, der totale Kontrolle über die Kommunikation zwischen den Protokollteilnehmern besitzt. Durch Protokollieren der im Netz auftretenden Nachrichten erlangt der Angreifer neue Kenntnisse. Außerdem kennt der Angreifer die Protokollvorschrift, um an der richtigen Stelle alte oder selbst generierte Nachrichten in die Kommunikation einzufügen. In Abbildung 4 ist ein solcher Angreifer schematisch dargestellt. Er besteht aus einem Zustandsautomat, der die Aktionen des Angreifers in Abhängigkeit von den erlangten Kenntnissen und den Ereignissen im Netz steuert, und den Speichern, in denen die Protokollvorschrift und die bisher aufgetretenen Nachrichten abgelegt sind. Der mit “NETZ” beschriftete Teil von Abbildung 4 enthält ein Modell der regulären Teilnehmer und des Kommunikationssystems. Die Teilnehmer starten Protokolläufe und reagieren entsprechend der Protokollvorschrift auf an sie adressierte Nachrichten. Mit der Produktnetzmaschine wird eine Erreichbarkeitsanalyse des gesamten Systems aus Teilnehmern und Angreifer durchgeführt. Der gesamte Erreichbarkeitsgraph wird auf Erfolg eines Angriffs durchsucht.

Die Modellierung eines solchen Systems wird am Beispiel des Needham-Schroeder Protokolls beschrieben. Dieses Protokoll ist durch eine Replay-Attacke angreifbar. In dem aus Angreifer und regulären Teilnehmern bestehenden Modell sollte daher ein Zustand erreichbar sein, in dem der aktuelle session-key dem Angreifer bekannt ist.

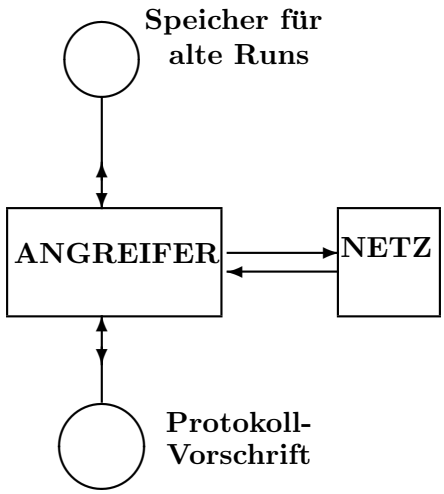


Abbildung 4: Angreifer (Konzept)

4.1.1 Das Needham-Schroeder Protokoll

Das Needham-Schroeder Protokoll ist ein Authentifikationsprotokoll, das durch Bereitstellung eines sicheren symmetrischen Schlüssels vertrauliche und authentische Kommunikation zwischen zwei Partnern ermöglichen soll. Mittels einer dritten Instanz,dem keyserver S, wird ein neuer symmetrischer Schlüssel zwischen den beiden Teilnehmern A und B vereinbart. Ein Sicherheitsziel ist, daß dieser Schlüssel keinem Unbeteiligten bekannt sein darf und für den aktuellen Protokollauf von S generiert wurde. Um dieses Ziel zu erreichen, wird folgendes vorrausgesetzt:

- Es existiert eine Authorität S, der bezüglich der Qualität neuer Schlüssel vertraut wird. In der Terminologie der BAN-Logik: "S hat Jurisdiktion über $A \stackrel{K}{\longleftrightarrow} B$".
- Jeder Teilnehmer teilt einen sicheren symmetrischen Schlüssel mit dem keyserver S.
- Die Teilnehmer sind in der Lage, nonces N_A und N_B zu generieren, die mit hinreichender Wahrscheinlichkeit in alten Protokolläufen nicht verwendet wurden.

Das Protokoll besteht aus 5 Schritten (siehe auch Abbildung 5):

1. $A \rightarrow S : A.B.N_A$
   Teilnehmer A startet das Protokoll durch eine Anfrage an den keyserver S.
2. $S \rightarrow A : \{N_A.B.K_{AB}; K_{AS}\}\{K_{AB}.A; K_{BS}\}$
   Als Antwort sieht er seine nonce N_A, die Adresse eines anderen Teilnehmers und einen neuen Schlüssel, zusammen verschlüsselt mit K_{AS}. Stimmt N_A mit der

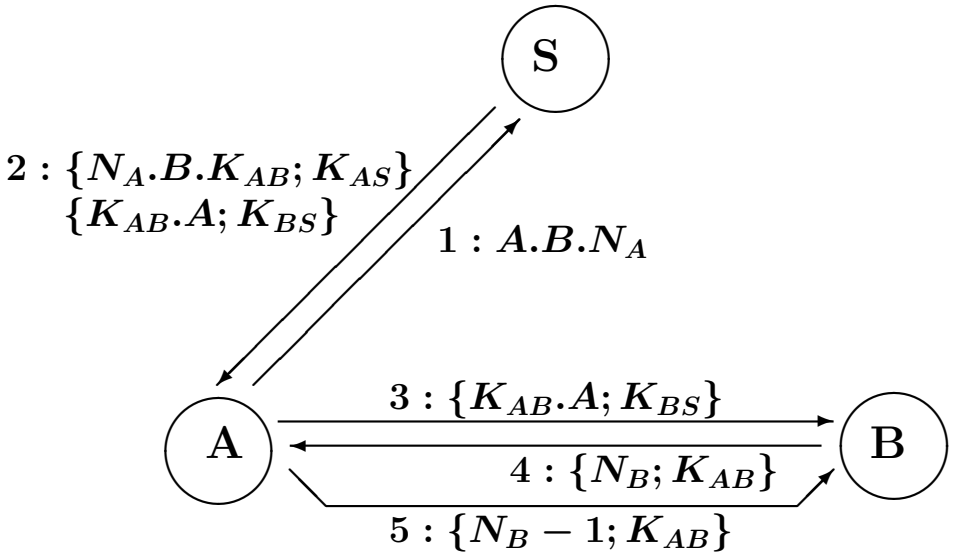


Abbildung 5: Das Needham-Schroeder Protokoll

nonce aus Nachricht 1 überein und steht an der zweiten Stelle der Nachricht die Adresse von B[4], akzeptiert A den neuen Schlüssel und verwendet ihn zur Kommunikation mit B. Außerdem gibt er den für ihn nicht lesbaren Nachrichtenteil $\{K_{AB}.A; K_{BS}\}$ an B weiter.

3. $A \rightarrow B : \{K_{AB}.A; K_{BS}\}$
   B kann diese Nachricht nicht von Nachrichten aus vorherigen Protokolläufen unterscheiden. Ein Angreifer kann durch Austausch gegen eine alte Nachricht einen nicht mehr sicheren Schlüssel einbringen.

4. $B \rightarrow A : \{N_B; K_{AB}\}$

5. $A \rightarrow B : \{N_B - 1; K_{AB}\}$
   Die Sicherheit des Protokolls wird durch die letzten beiden Nachrichten nicht verbessert. Eigentlich soll dieser Handshake beiden Teilnehmern zeigen, daß ihr Partner den neuen Schlüssel kennt und verwendet. Aber wenn K_{AB} dem Angreifer bekannt ist, kann er auch die Nachricht 5 generieren und damit die Anfrage aus Schritt 4 beantworten.

4.1.2 Modellierung der Teilnehmer

Das Modell der Teilnehmer basiert auf der funktionellen Beschreibung in der Spezifikation des Protokolls. Jedes Verhalten ist möglich, solange es mit den Vorschriften der Spezifikation übereinstimmt.

[4]Die Prüfung der Identität von B ist notwendig, sonst ergibt sich eine weiterer Angriffsmöglichkeit.

In diesem Beispiel genügt es, den Teilnehmern feste Rollen zuzuordnen. Im allgemeinen wird man für alle regulären Teilnehmer dasselbe Modul verwenden, so daß sie beide Rollen übernehmen können. Die einzelnen Teilnehmer werden als Teil des verteilten Algorithmus durch je ein Produktnetz modelliert (Abbildung 6). Die Kommunikation erfolgt über die Stelle M (Messages), indem die Netze über diese gemeinsame Stelle zu einem Netz verklebt werden. Auf M werden mit Adressen versehene Nachrichten gelegt. Diese Stelle ist der Einflußbereich des Angreifers. Liegt auf M eine Nachricht mit der passenden Adresse, kann sie von dem jeweiligen Teilnehmer weiterverarbeitet werden. Entsprechend der Protokollvorschrift prüft der Teilnehmer, ob er die Nachricht akzeptieren kann und legt die jeweilige Antwort auf M. Jeder Protokollschritt wird mit einer eigenen Transition modelliert. So kann der Ablauf des Protokolls in der Analyse leicht nachvollzogen werden. So startet z. B. T_REQ (request) ein neues Protokoll mit einer Anfrage an S, T_GIVE erzeugt die dritte Nachricht aus den Daten der zweiten, wenn S auf den request geantwortet hat, usw.[5] Über die Transitionsinschriften werden aus dem Topf der Nachrichten in M diejenigen mit der passenden Adressierung ausgewählt. Entsprechend den Modellannahmen kann A erst einen neuen Lauf starten, wenn der vorherige Lauf beendet wurde. Eine Marke auf der Stelle NEU bedeutet, daß aus Sicht des Teilnehmers A der letzte Protokollauf beendet wurde. Entgegen der allgemeinen Modellannahmen können aber im System noch nicht verarbeitete Nachrichten aus alten Protokolläufen existieren. In der Realität kann A aber nur aus seiner lokalen Sicht über die Beendigung des alten Laufs entscheiden. Eine Manipulation des Verhaltens von A durch Entscheidungen in Abhängigkeit vom globalen Systemzustand widerspricht der Vorstellung, daß sich die Teilnehmer möglichst wie in einer realistischen Implementation des Protokolls verhalten sollen.

Der keyserver S ist ein Teilnehmer mit besonderem Verhalten. Er besitzt einen Vorrat neuer symmetrischer Schlüssel. Dieser wird durch einen Zähler auf der Stelle KAB simuliert. Auf jeden request antwortet S mit der Bereitstellung eines neuen Schlüssels.

Der Angreifer (Abb. 7) kann alle Nachrichten lesen und speichern. Sobald er eine Nachricht besitzt, die syntaktisch der nächsten regulären Nachricht entspricht, kann er einen Angriffsversuch starten. Dazu müssen die Teile der Nachricht, die von dem angegriffenen Teilnehmer überprüft werden, exakt übereinstimmen, während Nachrichtenteile, die unbekannt sind oder nicht geprüft werden, beliebig sein können. Die Erreichbarkeitsanalyse der Produktnetzmaschine ermöglicht es, jeweils beide Fälle (Angriffsversuch und Weiterlaufen des Protokolls) zu betrachten.
Nonces und Schlüssel verändern sich teilweise von Protokollauf zu Protokollauf. Da der eigentliche Wert hier keine Rolle spielt, werden veränderliche Daten durch einfache Zähler modelliert. Durch die Wahl unterschiedlicher Anfangswerte und Zählschritte bleiben die einzelnen Zähler während der Analyse unterscheidbar.
Die einzelnen Protokolläufe werden durch den Zähler FC (freshcount) unterschieden.

[5] Zur Erläuterung des Modells siehe auch die Tabellen 1 und 2

Erläuterung der Stellen von A,B und S	
RUNS	Anzahl der von A zu startenden Läufe
N_A N_B	Zähler als Simulation für nonces
ABS	Adressen der Partner von A (Bei Erweiterung auf mehr Teilnehmer nötig)
FRESH_A, FRESH_B	Zwischenspeicher für die aktuelle nonce
K_TEMP	Zwischenspeicher für den neuen Schlüssel. Erst nach einer Antwort wird der Schlüssel als neuer session key freigegeben.
S_A_IN , S_B_IN, S_S_IN	Nachrichteneingang
KAB_A, KAB_B	Speicher für den, jeweils aus Sicht des Teilnehmers, aktuellen session key .
KAS, KBS	Speicher für die symmetrischen Schlüssel mit keyserver S
KAB	Zähler. Simuliert den Vorrat ungebrauchter Schlüssel, die S vergeben kann.
NEU	Ist ein Protokoll aus Sicht von A beendet, ermöglicht eine Marke auf NEU den Start eines neuen Laufes
FC	Zähler der Protokolläufe, um in der Analyse in globaler Betrachtungsweise die einzelnen Läufe unterscheiden zu können, und den Angreifer zu steuern. Der Angreifer soll bei seiner Replay-Attacke nur alte Nachrichten verwenden.
K_ALT	Speicher für alte Schlüssel. Durch Zugriff des Angreifers auf diese Stelle wird simuliert, daß der Angreifer durch irgendeine Methode in den Besitz alter Schlüssel gelangt.

Tabelle 1: Stellen von A,B und S im Modell des Needham-Schroeder Protokolls

Erläuterung der Stellen von C	
MEM	Speicher aller mitgelesenen Nachrichten
MEM_FC	Alte Nachrichten mit freshcount markiert. Hier wird die Mächtigkeit des Angreifers eingeschränkt, indem jede Nachricht nur einmal wiederholt werden kann.
REP_Ps	Adressen der zu attackierenden Teilnehmer
RESP	Speicher zur Unterscheidung von Anfrage und Antwort in Nachrichten 4 und 5
K_ALT, FC	Globale Stellen, siehe Tabelle 1

Tabelle 2: Stellen des Angreifers C im Modell des Needham-Schroeder Protokolls

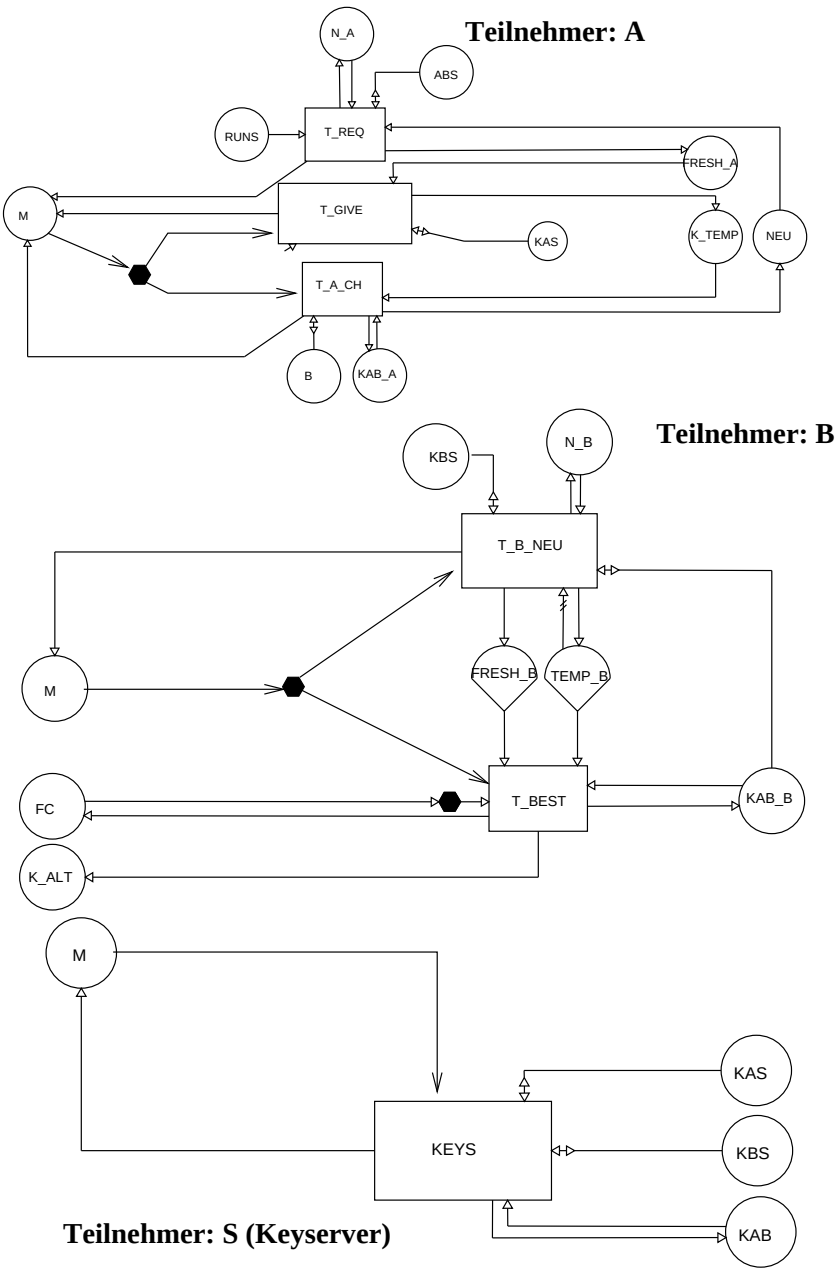


Abbildung 6: Reguläre Teilnehmer des Needham-Schroeder Protokolls

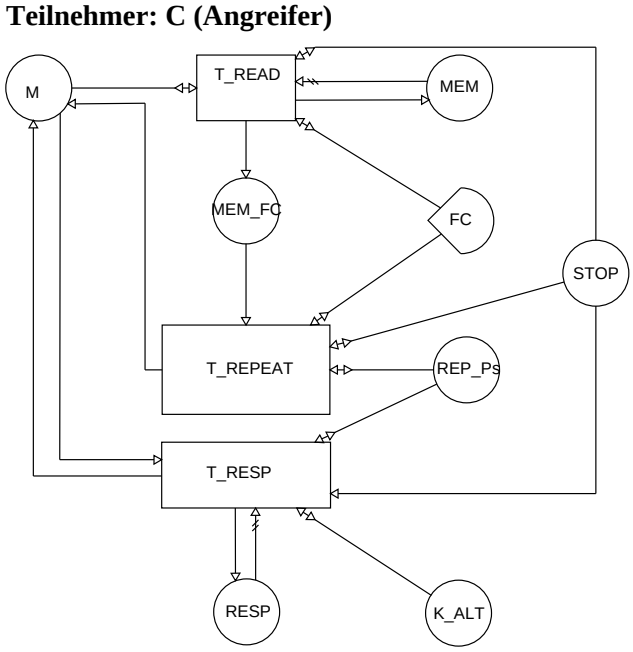


Abbildung 7: Angreifer für replay Attacken

Dabei wird die globale Sicht des Analysierenden ausgenutzt. Alte Session-Keys werden dem Angreifer über die Stelle K_ALT direkt zur Verfügung gestellt. Um einen endlichen Erreichbarkeitsgraphen zu erhalten, wird die Anzahl der Protokolläufe beschränkt. Im allgemeinen ist vor der Analyse nicht bekannt, wieviele Protokolläufe notwendig sind, bis sich die Möglichkeiten des Angreifers nicht mehr durch weitere Läufe verändern. Benutzt man ein gegenüber der BAN-Logik erweitertes Modell, ist nicht ohne weiteres zu erkennen, wieviele reguläre Teilnehmer im System für eine aussagekräftige Sicherheitsüberprüfung nötig sind. In diesem Beispiel genügt es, sich auf die Minimalzahl an Teilnehmern und zwei Protokolläufe zu beschränken.
In Anhang A sind der Vorspann und die beschrifteten Netze dargestellt.

4.1.3 Analyse

Zuerst wird das Modell aus A,B und S, ohne Einfluß des Angreifers, auf Übereinstimmung mit der Spezifikation überprüft. Danach wird mit aktiviertem Angreifer der gesamte Erreichbarkeitsgraph berechnet und auf Erfolg eines Angriffs durchsucht. Im Beispiel erhält der Angreifer die Vorgabe, Teilnehmer B soll angegriffen werden. Ein Versuch mit Angriffsziel A führt nicht zu einem erfolgreichen Angriff. Ein Erfolg ist gegeben, wenn der von B als aktuell anerkannte Schlüssel in der Menge der Schlüssel enthalten ist, die dem Angreifer C bekannt sind, d.h., wenn die Marke auf KAB_B in der Menge der Marken auf K_ALT enthalten ist.
Für einen erfolgreichen Angriff genügen zwei Protokolläufe, damit ein alter symmetrischer Schlüssel mit den passenden Nachrichten vorliegt. Der Angreifer benötigt die

von S erzeugte und mittels K_{BS} verschlüsselte Nachricht, die er nicht selbst generieren kann. Der für dieses Netz berechnete Erreichbarkeitsgraph besteht aus 240 verschiedenen Markierungen mit 24 möglichen Endzuständen (Totmarkierung). Zur Prüfung der gewünschten Sicherheitseigenschaft genügt es, nur die Stellen KAB_B und K_ALT auszuwerten. An der letzten der folgenden möglichen Markierungen erkennt man einen erfolgreichen Angriff:

```
KAB_B: 1<0>            (Anfangsmarkierung)

KAB_B: 1<101>
K_ALT: 1<0>            (nach dem 1. Lauf)

KAB_B: 1<201>          (nach dem 2. Lauf)
K_ALT: 1<0> + 1<101>

KAB_B: 1<101>          (nach erfolgreichem Angriff)
K_ALT: 1<0> + 1<101> + 1<201>
```

4.1.4 Vorteile der direkten Modellierung

Sowohl reguläre Teilnehmer als auch der Angreifer sind entsprechend ihrem realen Verhalten modelliert. So kann das dynamische Verhalten dieses Systems analysiert werden. Gegenüber einer auf der BAN-Logik basierenden Methode bietet dieser Ansatz einige Vorteile.

- Das Berechnungsmodell kann leicht geändert werden. Zum Beispiel können parallele Läufe zugelassen werden.
- Die Angriffe können klassifiziert werden und je nach Angriffsart einzeln betrachtet und bewertet werden.
- Die Problematik der Erstellung eines idealisierten Protokolls wird eingeschränkt, da das Verhalten der Teilnehmer genau modelliert werden muß. So wird zum Beispiel die Schwäche des Protokolls aus Kapitel 3.4 erkannt.

Trotz der Vorteile dieser Methode ist es fraglich, ob ein allgemeines Modell eines universellen Angreifers für beliebige kryptographische Protokolle analysiert werden kann. Das grundlegende Problem ist die Komplexität. Erweitert man das Modell oder gibt dem Angreifer mehr Möglichkeiten, explodiert die Menge der erreichbaren Zustände.

4.1.5 Komplexität

Das obige Beispiel ergibt mit 240 Zuständen einen kleinen Zustandsraum.[6] Dies liegt aber nur daran, daß dem Angreifer vorgegeben wird, welche Art von Attacke er durchzuführen hat. Weitere Beispiele führen zu explodierenden Zustandsräumen. Eine mögliche Variante ist, den oben beschriebene Angreifer mit etwas weniger "Wissen" auszustatten. Er wiederholt nur noch Nachrichten an den richtigen Adressaten in der vom Protokoll vorgegebenen Reihenfolge, ohne die Syntax zu überprüfen. Jede Nachricht wird höchstens einmal wiederholt. Diese Änderung ergibt eine realistische Situation, da der Angreifer die verschlüsselten Nachrichten zwar speichern und wiedergeben kann, jedoch den Inhalt nicht sieht. Durch diese Änderung ergibt sich ein Anwachsen des Zustandsraumes auf 34993 Zustände. Lässt man ausserdem noch parallele Protokolläufe zu, wächst der Zustandsraum auf mehr als 300.000 Zustände.
Ein universeller Angreifer hat so viele verschiedene Möglichkeiten, daß eine Erreichbarkeitsanalyse zu nicht mehr mit annehmbarem Aufwand berechenbaren Größenordnungen führt. Das Dilemma bei dieser Methode ist, daß zumindest ein Verdacht über eine Schwäche des Protokolls bestehen muß, um einen Angriff zu finden. Meistens liegen die interessanten Fehler beim Protokoll aber genau da, wo man sie nicht vermutet. Versuche zur Minimierung des Zustandsraumes mit konventionellen Methoden, wie in [3] vorgeschlagen, führten zu einer Verkleinerungen des Zustandsraumes, ändern aber nicht die Komplexität des Problems. In einigen Fällen könnten solche Methoden eine Berechnung ermöglichen. Verlässt man allerdings das eingeschränkte Modell der BAN-Logik, dürfte es meiner Meinung nach schwierig sein, das Anwachsen des Zustandsraums zu beherrschen.
Um eine allgemeine Methode zu erhalten, ist wohl ein anderer Ansatz notwendig. Eventuell könnte man einen Angreifer mit den Mitteln einer Logik "intelligenter" machen. Diese Überlegung führte zum nächsten Ansatz, ein automatisches Beweisverfahren für die BAN-Logik mit Produktnetzen zu implementieren.

4.2 Die Tableau-Methode für die BAN-Logik

Die Beweise der BAN-Logik werden in [4] derart dargestellt, daß die für das jeweilige Protokoll sinnvollen Initialaussage angegeben werden und versucht wird, daraus die erreichbaren Sicherheitsziele abzuleiten. Es ist aber vor der Analyse eines Protokolles keineswegs klar, welche Annahmen notwendig sind, um die Sicherheit des Protokolles zu gewährleisten. Das gewünschte Sicherheitsziel des Protokolles ist normalerweise schon beim Protokolldesign bekannt. Es ist daher naheliegend, vom Ziel ausgehend, die Initialaussagen herzuleiten. Es gibt im allgemeinen mehrere mögliche Mengen von Initialaussagen, die jeweils mit den Protokollnachrichten die gewünschte Sicherheitsaussage ermöglichen. Das Protokoll erreicht das gewünschte Ziel, wenn eine der Mengen von Initialaussagen in dem System, in dem das Protokoll läuft, erfüllbar ist. Eine geeignete Methode um diese Vorgehensweise zu automatisieren, ist die Tableau-

[6] Um die Zahlen einordnen zu können, sei erwähnt, daß mit der Produktnetzmaschine je nach Speichergrösse Automaten mit mehreren hunderttausend Zuständen analysiert werden können.

Methode. Gegenüber der üblichen Tableau-Methode, wie sie zum Beispiel in [5] beschrieben wird, erfordert die BAN-Logik einige Änderungen. Normalerweise wird in der Tableau-Methode eine Aussage bewiesen, indem die verneinte Aussage zum Widerspruch gebracht wird. Beweise werden als Baum aufgeschrieben. An der Wurzel wird die Verneinung der zu beweisenden Aussage notiert. Der Baum wird dann mit Hilfe der Tableau-Regeln aufgebaut. Seien zum Beispiel X und Y Aussagen einer Logik, steht nun $X \wedge Y$ an einem Ast, können X und Y an den Ast angefügt werden. Ein anderes Beispiel: Steht $X \vee Y$ an einem Ast, kann dieser Ast in eine linke und rechte Fortsetzung aufgeteilt werden und der linke Sohn mit X und der rechte mit Y beschriftet werden. Die Aussage an der Wurzel ist widerlegt, wenn in jedem Ast ein Widerspruch (z.B X und $\neg X$) aufgetreten ist. Diese Methode ist nicht deterministisch. Je nachdem, welche Regeln verwendet werden entstehen unterschiedliche Tableaus.

Für die BAN-Logik ist eine andere Vorgehensweise nötig. Das Ziel des BAN-Logik Beweises unterscheidet sich von den in [5] vorgestellten Logiken. Es soll nicht über "wahr" oder "falsch" entschieden werden, sondern es sollen lediglich Vorbedingungen abgeleitet werden. Wegen der fehlenden Verneinung können keine Aussagen zum Widerspruch gebracht werden. Die Wurzel wird mit der zu beweisenden Aussage beschriftet. Die Tableau-Methode kann für die BAN-Logik deterministisch durchgeführt werden. Es steht nur eine endliche Menge an Regeln mit einer endlichen Anzahl möglicher Interpretationen durch Daten aus dem Protokoll, zur Verfügung. In jedem Ast werden jeweils solange alle möglichen Vorbedingungen abgeleitet, bis keine Regel mehr anwendbar ist, oder nur noch Vorbedingungen abgeleitet werden können, die im Ast schon vorhanden sind.

4.2.1 Umformung der BAN-Logik Regeln

Um die BAN-Logik Regeln "rückwärts" anwenden zu können, müssen sie entsprechend Abbildung 8 in sogenannte Tableau-Regeln umgeformt werden. Die message-meaning rule wird zum Beispiel zu:

$$\frac{\boldsymbol{P \mid\equiv Q \mid\sim X}}{\boldsymbol{(P \mid\equiv Q \overset{K}{\longleftrightarrow} P) \wedge (P \triangleleft \{X; K\})}}$$

Damit P glauben kann, daß Q irgendwann X gesagt hat, genügt es, daß P eine mit K verschlüsselte Nachricht gesehen hat und K für einen sicheren Schlüssel zur Kommunikation mit Q hält.

Die Vorbedingungen stehen bei Tableauregeln "unter dem Strich". Entsprechend den Regeln wird, beginnend mit der zu beweisenden Formel, sukzessiv das Tableau aufgebaut. Mit "und" verknüpfte Bedingungen stehen in demselben Ast, mit "oder" verknüpfte Bedingungen erzeugen eine Verzweigung und stehen in verschiedenen Ästen. Lassen sich auf eine Formel mehrere Regeln oder verschiedenen Interpretationen einer Regel (s.u.) anwenden, erzeugt jede Anwendung hinreichende Vorbedingungen, die jeweils in einem eigenen Ast des Tableaus stehen. Für jeden Pfad durch das Tableau müssen die Formeln markiert werden, auf die Regeln angewendet

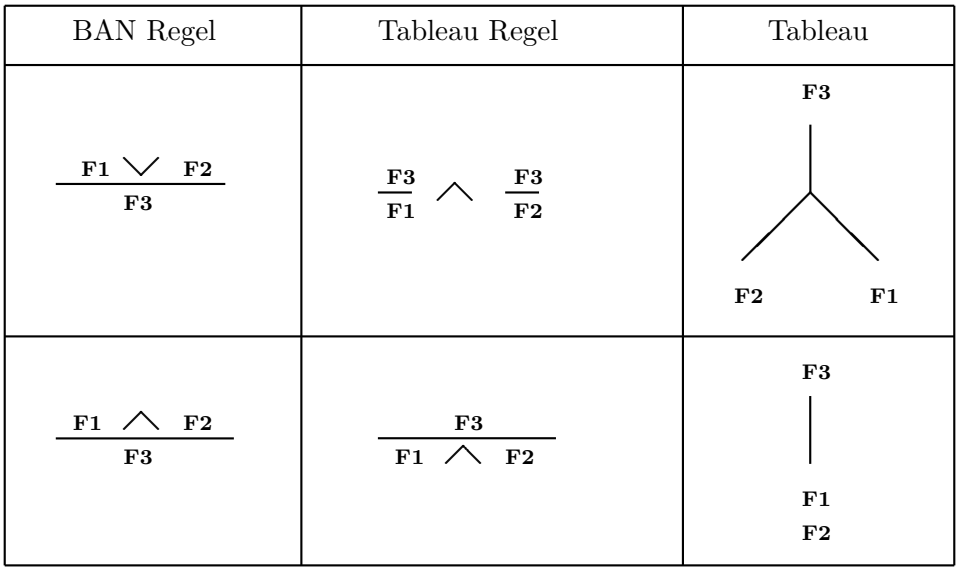


Abbildung 8: Prinzipien zur Umformung in Tableau-Regeln

wurden. Die nicht markierten Formeln auf einem Pfad von der Wurzel zu einem Blatt bilden eine hinreichende Menge von Vorbedingungen für die Formel an der Wurzel.

4.2.2 Einbindung des idealisierten Protokolls in den Beweis

In den Regeln ist keine Information über das Protokoll enthalten. Der Bezug von Protokollschritten zur Anwendung der Regeln wird über die Belegung von Variablen geregelt. In den Vorbedingungen stehen bei einigen Regeln Variablen, die mit Werten aus dem idealisierten Protokoll belegt werden müssen. Eine solche Regel ist zum Beispiel:

$$\frac{\boldsymbol{P} \mid\equiv \boldsymbol{X}}{\boldsymbol{P} \mid\equiv (\boldsymbol{X}.\boldsymbol{Y})}$$

Damit P glaubt, X sei wahr, genügt es, daß er X und Y glaubt. Diese Folgerung scheint trivial, aber Y muß aus dem idealisierten Protokoll gewählt werden. Will man die Vorbedingung von $P \mid\equiv X$ bezüglich dieser Regel ableiten, muß (X.Y) in einer Nachricht im Protokoll auftreten, die von P gesehen wird. Gibt es mehrere Möglichkeiten, die Variablen zu belegen, kann die Regel mit verschiedenen Interpretationen angewendet werden.
Informationen über Variablen sind zur Implementation der Tableaumethode wichtig. Deshalb werden die Regeln in zwei Klassen unterteilt:

protokollabhängige Regeln: In den zu erzeugenden Vorbedingungen stehen Variablen, die mit Objekten aus dem idealisierten Protokoll belegt werden müssen.

protokollunabhängige Regeln: Zur Anwendung dieser Regeln wird keine Information aus dem idealisierten Protokoll benötigt.

Im Anhang B.1 wird eine Aufstellung der wichtigsten Tableau-Regeln zur BAN-Logik, eingeteilt nach protokollabhängigen und -unabhängigen Regeln angegeben.

4.2.3 Tableau-Beweis zum Needham-Schroeder Protokoll

In diesem Abschnitt wird zu dem in Kapitel 4.1.1 dargestellten Needham-Schroeder Protokoll ein Beweis mit der Tableau-Methode vorgeführt und die Bewertung des Ergebnisses erläutert. Vor der Anwendung der Tableau-Methode muß das Sicherheitsziel, das das zu untersuchende Protokoll erreichen soll, bestimmt werden. Sinnvoll ist es, mit der minimalen Sicherheitsforderung zu beginnen. Wird diese nicht erreicht, müssen stärkere Forderungen nicht mehr betrachtet werden. Das Needham-Schroeder Protokoll hat primär die Zielsetzung $A \models A \overset{K_{AB}}{\longleftrightarrow} B$ und $B \models A \overset{K_{AB}}{\longleftrightarrow} B$. Auf diese Aussagen konzentriert sich der Beweis. Würden sie vom Protokoll mit annehmbaren Vorbedingungen erreicht, könnte man den Beweis mit $A \models B \models A \overset{K_{AB}}{\longleftrightarrow} B$ und $B \models A \models A \overset{K_{AB}}{\longleftrightarrow} B$ weiterführen.

Die Abbildungen 9 und 10 zeigen die graphischen Darstellungen der jeweiligen Tableaubeweise zu den ersten beiden Formeln. Erstellt wurden diese Tableaus mit maschineller Unterstützung durch die im nächsten Kapitel beschriebene Implementation der Methode.
An der Wurzel der Tableaus steht jeweils die zu beweisende Aussage. Die Zahlen an den Kanten geben die Regeln entsprechend der Implementation (siehe Abbildung 11 und Anhang B.1) an. Gleiche Knoten wurden zusammengefaßt und Pfade, die sich nur durch die Reihenfolge der Regelanwendungen unterscheiden, bis auf einen Repräsentanten ausgeblendet. Auf Formeln, die nicht mit eckigen Klammern markiert sind, können keine Regeln angewendet werden. Sie sind mögliche Initialaussagen zu Beginn des Protokollaufs (vergleiche Kapitel 3).

Ergebnis für $A \models A \overset{K_{AB}}{\longleftrightarrow} B$:
Im Tableau in Abbildung 9 gibt es 7 verschiedene Pfade von der Wurzel zum Blatt.
Im rechten Teilbaum enthält jeder Pfad die nicht markierte Formel
$A \models S \Longrightarrow N_A.B.A \overset{K_{AB}}{\longleftrightarrow} B.$
Die nonce N_A wird aber von A erzeugt und von S nur wiedergegeben. Daher kann S keine Jurisdiktion über die nonce N_A besitzen. Diese Vorraussetzung kann also nicht erfüllt werden.
Der linke Teilbaum enthält einen Pfad, der mit "sinnvollen" Annahmen beschriftet ist. Er ist durch doppelte Kanten markiert.Die Initialaussagen aus diesem Pfad sind:

$A \models S \Longrightarrow A \overset{K_{AB}}{\longleftrightarrow} B$	Vertrauen in den keyserver ist nötig.
$A \models \sharp N_A$	A glaubt seine eigene nonce ist frisch.
$A \models A \overset{K_{AS}}{\longleftrightarrow} S$	Der Schlüssel K_{AS}, den A und S teilen, sollte gut sein.

$A \triangleleft \{N_A.B.A \overset{K_{AB}}{\longleftrightarrow} B; K_{AS}\}$ Diese Nachricht ist im Protokoll an A adressiert und kann von A gesehen werden

Die Existenz eines solchen Pfades bedeutet, daß das Ziel $A \models A \overset{K_{AB}}{\longleftrightarrow} B$ des Protokolls im Modell der BAN-Logik erreicht wird. Die restlichen Pfade des linken Teilbaumes enthalten die Formeln $A \models \sharp B$ oder $A \models \sharp A \overset{K_{AB}}{\longleftrightarrow} B$. Beide können nicht als sinnvolle Vorraussetzungen betrachtet werden. A kann nicht darauf vertrauen, daß der neue Schlüssel noch nie verwendet wurde. Die Adresse von B kann auch nicht dazu dienen, zu garantieren, daß eine Nachricht in vorherigen Protokolläufen nicht verwendet wurde.
Ergebnis für $B \models A \overset{K_{AB}}{\longleftrightarrow} B$:
Im Tableau in Abbildung 10 gibt es 6 Pfade von der Wurzel zu einem Blatt. Keiner dieser Pfade ergibt eine sinnvolle Menge an Vorbedingungen. In jedem Pfad ist entweder $B \models \sharp A$ oder $B \models \sharp A \overset{K_{AB}}{\longleftrightarrow} B$ enthalten. B hat aber keinen Grund, die nonce von A oder den neuen Schlüssel für frisch zu halten. Das Sicherheitsziel $B \models A \overset{K_{AB}}{\longleftrightarrow} B$ wird nicht erreicht.

4.2.4 Implementation der Tableau-Methode

Obwohl die Produktnetzmaschine zur Eigenschaftsverifikation von Modellen entwickelt wurde, kann man sie auch sinnvoll zur Implementation der Tableau-Methode verwenden. Um sich die Möglichkeit zur Kombination verschiedener Verfahren offen zu halten, kann es vorteilhaft sein, sich bei der Implementation auf Verwendung einer Plattform zu beschränken.
Interpretiert man ein im Aufbau befindliches Tableau als Zustand und die Anwendung einer Regel als Zustandsübergang, kann man einen Beweis mit der Tableau-Methode direkt auf einen Zustandsautomaten übertragen. Der Erreichbarkeitsgraph der Produktnetzmaschine erweist sich dabei als geeignete Datenstruktur zur Darstellung des Tableaus. Es werden jeweils nur die Formeln in einer Markierung gespeichert, für die noch keine Vorbedingungen abgeleitet wurden. In den Totmarkierungen stehen daher nur noch Aussagen, die nicht mehr weiter abgeleitet werden können, die also jeweils eine nicht mehr abzuschwächende Menge hinreichender Vorbedingungen darstellen. Die speziellen Zeichen der BAN-Logik, wie $\models, \triangleleft, \mid\sim$ oder $\sharp$, sind im Zeichensatz für die Produktnetzmaschine nicht vorhanden. Um die Beweise gut lesbar zu halten, werden für die BAN-Logik-Zeichen jeweils an die Bedeutung angelehnte deutsche oder englische Wörter eingeführt. Die Aussage $A \models \sharp X$ wird z. B. zu `A.glaubt.frisch.X`. Das Protokoll wird als Menge von Nachrichten der Struktur `(P,M)` dargestellt. Dabei ist P der Adressat der Nachricht M.

Die Implementation besteht aus dem Vorspann und zwei Produktnetzen, die über gemeinsame Stellen verbunden sind. Ein Netz (Abb. 11) steuert den Ablauf des Beweises durch nichtdeterministische Anwendung der Tableau-Regeln. Das zweite Netz (Abb. 12) erzeugt zu einer Formel mit Hilfe des idealisierten Protokolls alle möglichen Variablenbelegungen zur Anwendung protokollabhängiger Regeln. Die Netze

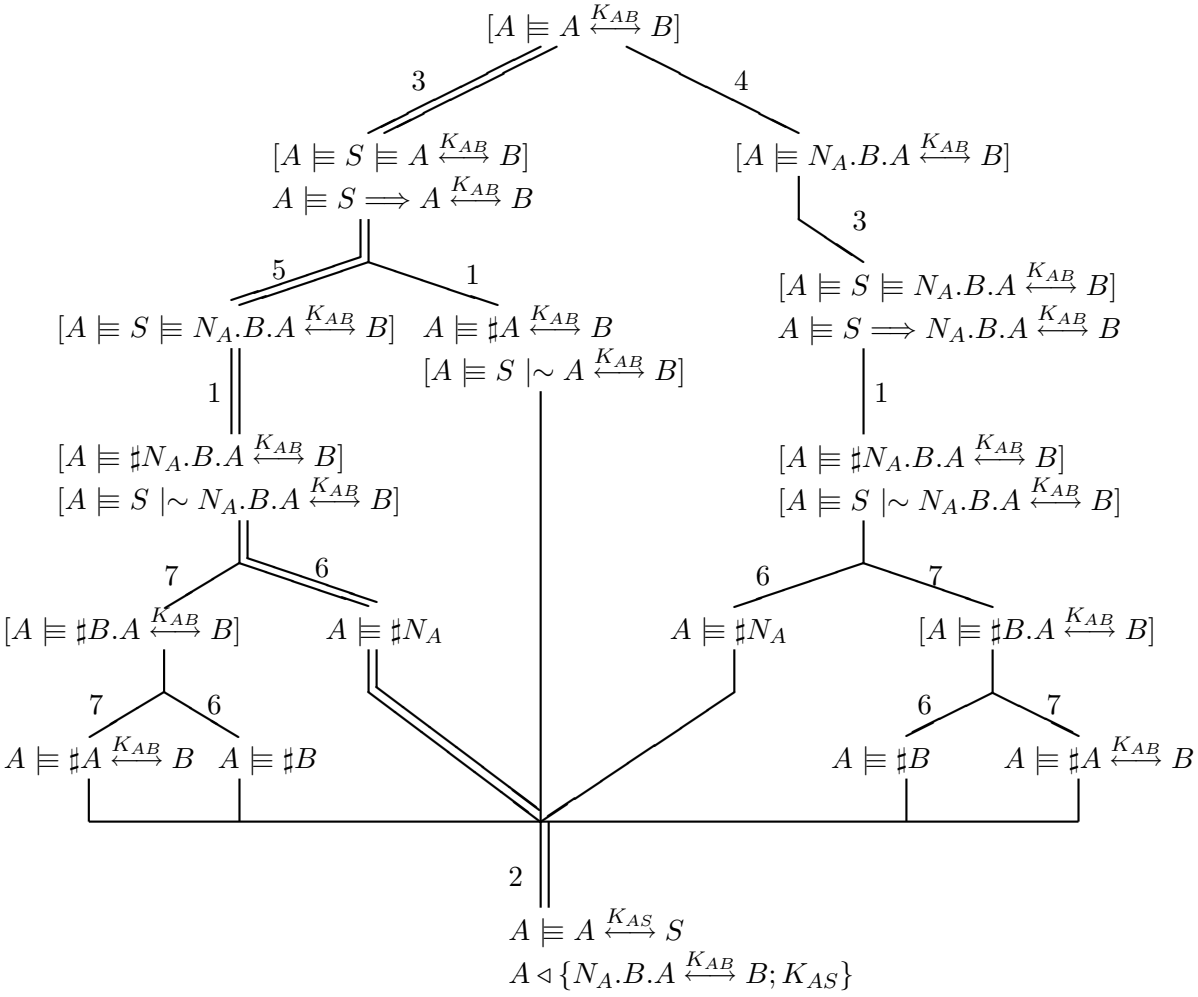


Abbildung 9: Tableau für $A \models A \overset{K_{AB}}{\longleftrightarrow} B$ im Needham-Schroeder Protokoll

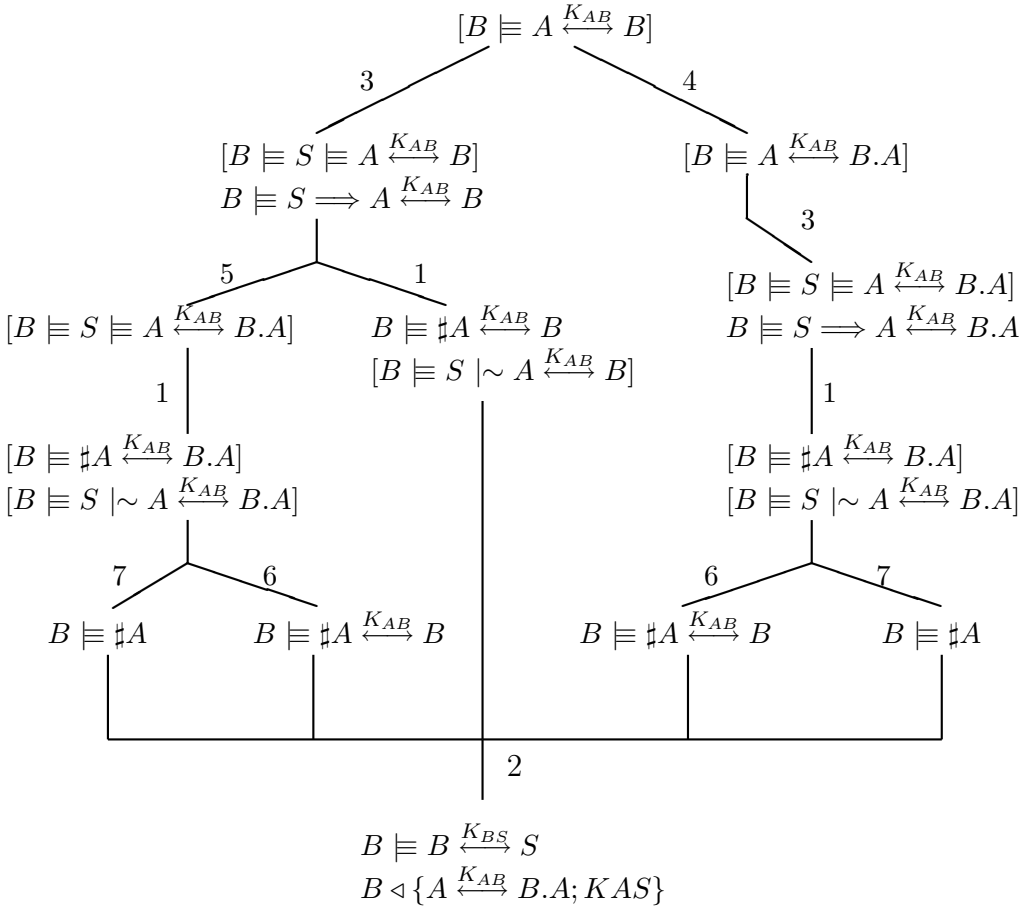


Abbildung 10: Tableau für $B \models A \overset{K_{AB}}{\longleftrightarrow} B$ im Needham-Schroeder Protokoll

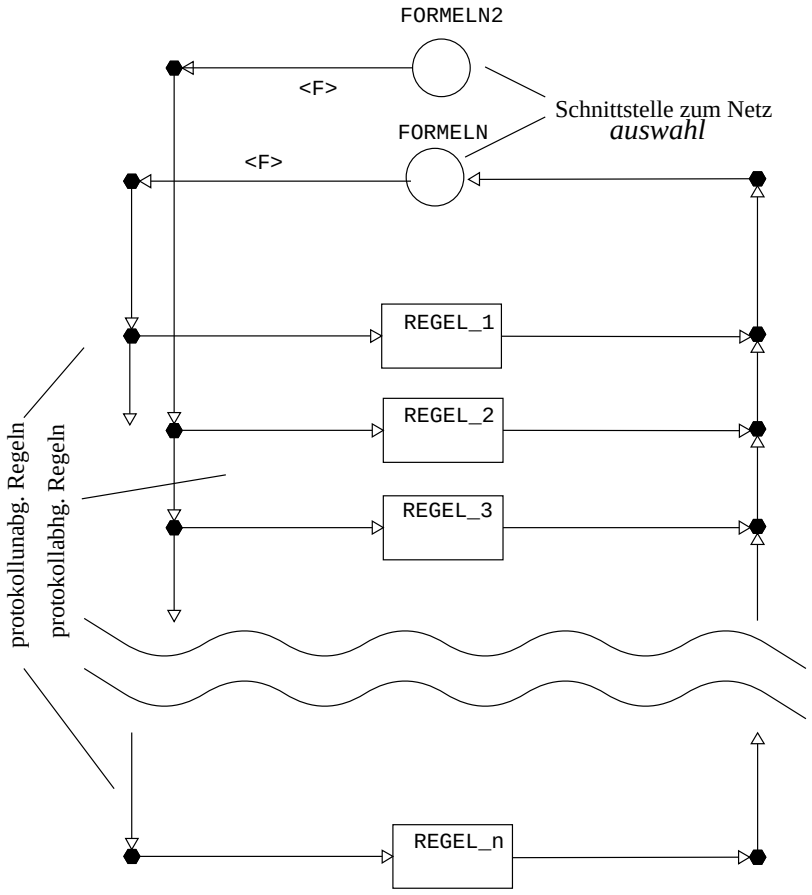


Abbildung 11: Skizze für das Netz *tableau*

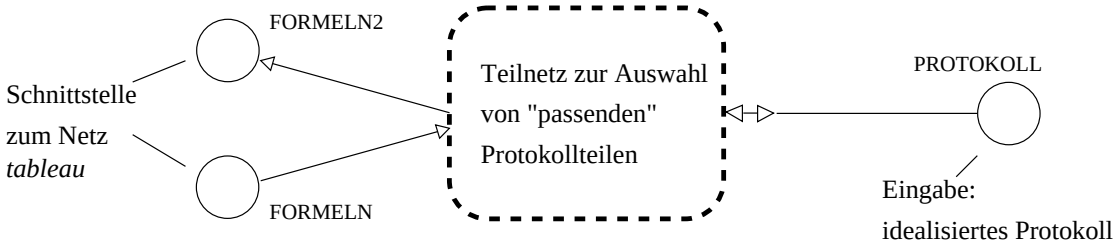


Abbildung 12: Skizze für das Netz *auswahl*

beschreiben den logischen Ablauf des Algorithmus, während im Vorspann alle "inhaltlich" wichtigen Funktionen definiert sind. Sie geben die Bedingungen für das Schalten einer Transition, also für den Fortgang des Algorithmus an, und werden zur Erzeugung der Folgemarkierungen aufgerufen. Die Erreichbarkeitsanalyse des gesamten Netzes [7] wird so zum Lauf eines Programms, das einen Tableaubeweis zur BAN-Logik berechnet. Als Eingabedaten werden über die Anfangsmarkierung das idealisierte Protokoll und die zu beweisende Aussage eingegeben. Änderungen im Vorspann oder an den Netzen werden nur notwendig, wenn die Logik geändert oder die zur Beschreibung der Protokolle definierte Sprache erweitert werden soll. Zu den in Anhang B.3 angegebenen Protokollen wurden Tableaubeweise maschinell durchgeführt und jeweils die Ergebnisse aus [4] bestätigt.

5 Schlußbemerkung

In dieser Arbeit wurden zwei grundsätzlich unterschiedliche Ansätze zur maschinellen Unterstützung der Verifikation kryptographischer Protokolle vorgestellt.

Die Konstruktion eines universellen Angreifers, quasi die Programmierung des "Computers des Bösen", scheitert an der Komplexität des Problems. Für bestimmte Klassen von Angriffen, die mit anderen Verfahren nicht abgedeckt werden, könnte dieser Ansatz trotzdem sinnvoll sein. Bei der Betrachtung von Protokollen realer Größe ist aber die Anwendung von Methoden zur Verkleinerung des Zustandsraumes notwendig. Ein großer Vorteil dieses Ansatzes ist, daß nur bekannte Verifikationsmethoden verwendet werden und das Analyseergebnis keiner weiteren Interpretation bedarf. Außerdem erfordert die Modellierung des Protokolls eine Reflektion der genauen Vorgänge beim Ablauf des Protokolls. Im Produktnetzmodell ist mehr Informationen über das Verhalten der einzelnen Teilnehmer enthalten als in der üblichen Spezifikation von Protokollen als Liste von Nachrichten.

Die Implementation einer Beweismethode auf Basis der BAN-Logik übernimmt die in Kapitel 3.4 dargestellten Schwächen der Logik. In ähnlicher Weise können aber auch Beweisverfahren für Weiterentwicklungen der BAN-Logik implementiert werden. Die Produktnetzmaschine erweist sich als geeignetes Werkzeug, die Methoden zu automatisieren. Die Trennung des logischen Programmablaufs von den Datenstrukturen und Funktionen erleichtert zudem die Anpassung an erweiterte Modelle und Logiken. Abhängig von der in der Produktnetzmaschine möglichen Datenstrukturen ergibt sich allerdings eine sehr komplexe Syntax zur Spezifikation der Protokolle. Für eine praktische Anwendung wäre es deshalb sinnvoll eine intuitiv verständliche konkrete formale Sprache zur Beschreibung von Protokollen entwickeln. Ein in einer solchen Sprache spezifiziertes Protokoll könnte dann automatisch in eine Anfangsmarkierung des Produktnetzes übersetzt werden kann.

[7] Die beiden Produktnetze werden über die Stellen FORMELN und FORMELN2 zu einem Netz verklebt.

A Modellierung eines Angreifers am Beispiel des Needham-Schroeder Protokolls

A.1 Vorspann und Netze

```
\* Mengendefinitionen *\


defset BOOLE = {1,0} ;
defset PRINCIPALS = { A,B,C,S } ;
defset DATEN = nat_0 || PRINCIPALS ;
defset DATENSEQ = seq(DATEN) ;
defset CRYPT1 =  pro(DATENSEQ,nat_0) ;
defset CRYPT2 =  CRYPT1 || DATEN ;
defset CRYPT3 =  seq(CRYPT2) ;

\* Datenstruktur verschluesselten Texts (Daten,nonces und Schluessel
werden durch natuerliche Zahlen modelliert)  *\
defset CRYPT = pro(CRYPT3,nat_0) ;

\* An jeweiligen Teilnehmer adressierte Nachrichten *\
defset MESSAGES = pro(CRYPT,PRINCIPALS) ;

\* Mit globaler (freshcount) Information versehene
Nachrichten fuer interne Verwendung des Angreifers *\
defset MESS_FC = pro(MESSAGES,nat_0) ;


defob ABS = pro (PRINCIPALS,PRINCIPALS,PRINCIPALS) ;


\* Funktionen zur Abkuerzung in Transitionsinschriften *\

defcase A : PRINCIPALS >> BOOLE
     A(prinz) = if prinz = A  then 1
                               else 0 ;

defcase B : PRINCIPALS >> BOOLE
     B(prinz) = if prinz = B  then 1
                              else 0 ;

defcase C : PRINCIPALS >> BOOLE
     C(prinz) = if prinz = C  then 1
                              else 0 ;

defcase S : PRINCIPALS >> BOOLE
     S(prinz) = if prinz = S  then 1
                              else 0 ;
```

Stellenname	Definitionsbereich
ABS	ABS
B	<PRINCIPALS>
FRESH_A	<nat_0>
KAB_A	<nat_0>
KAS	<nat_0>
K_TEMP	<nat_0>
M	<MESSAGES>
NEU	<nat_1>
N_A	<nat_0>
RUNS	<nat_1>

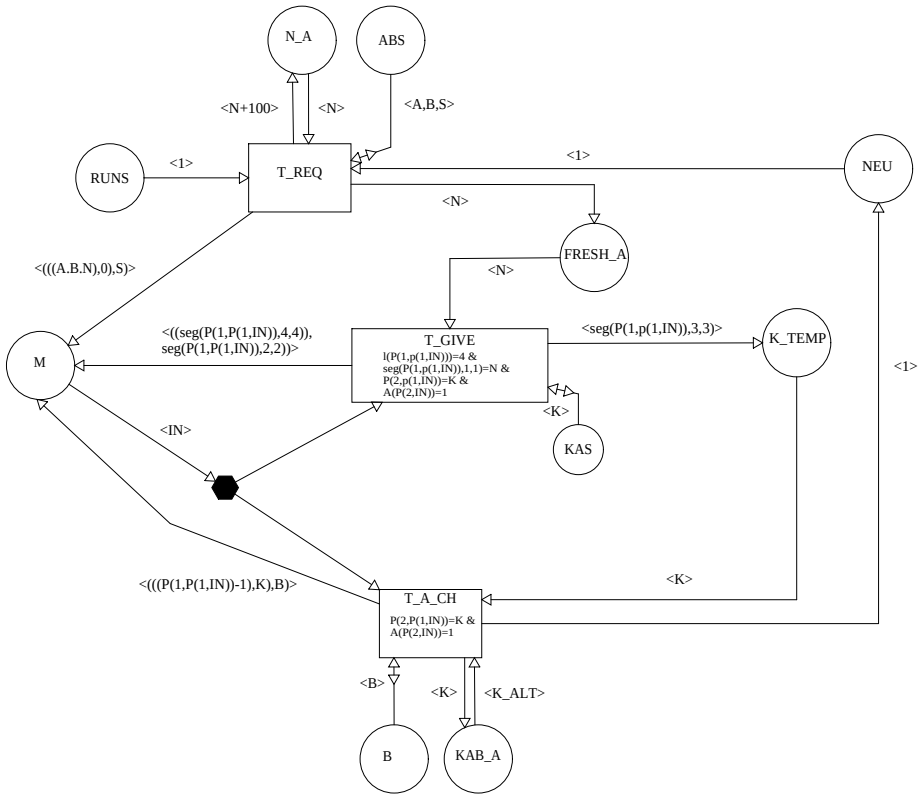


Abbildung 13: Teilnehmer A

Stellenname	Definitionsbereich
B	<PRINCIPALS>
FRESH_B	<nat_0>
KAB_B	<nat_0>
KBS	<nat_0>
K_ALT	<nat_0>
M	<MESSAGES>
N_B	<nat_0>
TEMP_B	<nat_0>

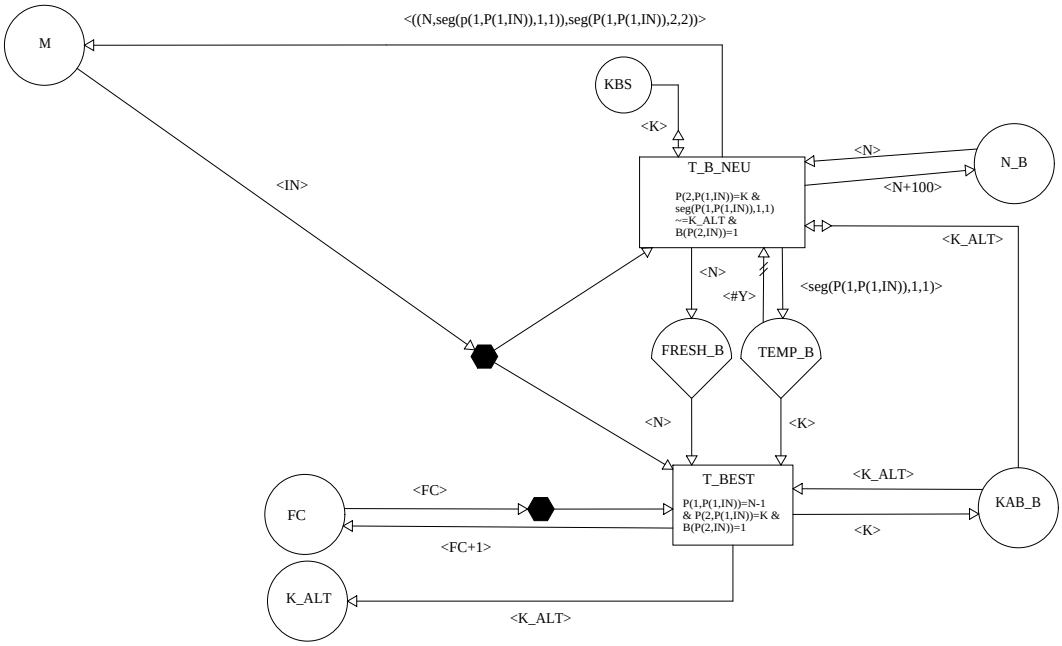


Abbildung 14: Teilnehmer B

Stellenname	Definitionsbereich
KAB	<nat_0>
KAS	<nat_0>
KBS	<nat_0>
M	<MESSAGES>

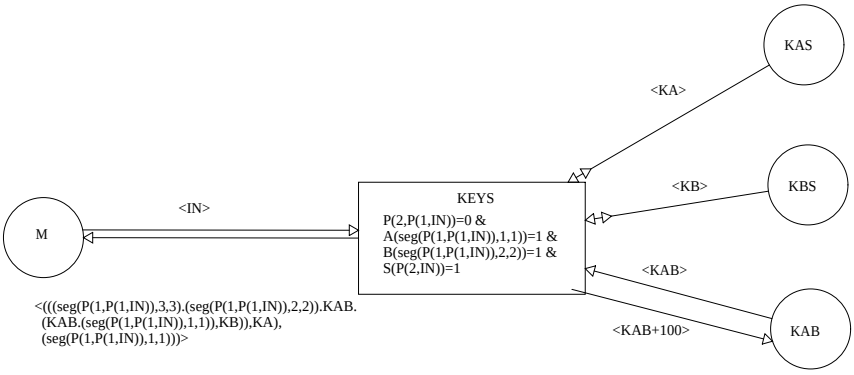


Abbildung 15: Keyserver S

Stellenname	Definitionsbereich
FC	<nat_0>
K_ALT	<nat_0>
M	<MESSAGES>
MEM	<MESSAGES>
MEM_FC	<MESS_FC>
REP_Ps	<PRINCIPALS>
RESP	<MESSAGES>
STOP	<nat_0>

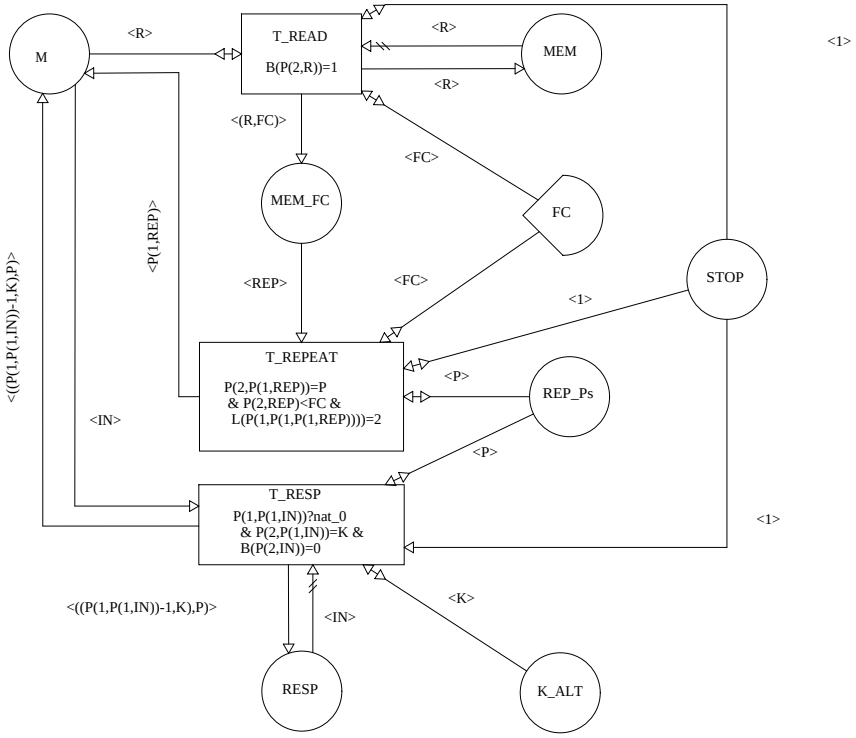


Abbildung 16: Angreifer C

A.2 Analyse

Angriffe zum Needham-Schroeder Protokoll:
(Gerechnet mit einem Sparc Ultra 1 Prozessor)

- Keine parallelen Protokolläufe
- Angreifer prüft die Syntax vor der Wiederholung einer Nachricht

```
 New Initial Marking:


 ABS: 1<A,B,S>
 B: 1<B>
 FC: 1<0>
 KAB: 1<101>
 KAB_A: 1<0>
 KAB_B: 1<0>
 KAS: 1<111>
 KBS: 1<222>
 NEU: 1<1>
 N_A: 1<100>
 N_B: 1<100>
 REP_PS: 1<B>
 RUNS: 2<1>
 STOP: 1<1>

Analysis
Start: 5.12.1996 10:53:10
Stop:  5.12.1996 10:53:16

 Reachability Graph needattackshort

 240 Markings computed.
 (24 Dead Markings)
 (0 Pseudo Occurrence Steps)
 (360 Occurrence Steps)

Minimal Automaton
Start: 5.12.1996 10:54:04
Stop:  5.12.1996 10:54:04

 Minimal Automaton needattackshort

 5 States computed.
 (36 critical Markings)
 (4 Edges, 4 Labels)

 Q-Markings : needattackshort

 Q-1
 KAB_B: 1<101>
 K_ALT: 1<0> + 1<101> + 1<201>
```

```
 Q-2
 KAB_B: 1<201>
 K_ALT: 1<0> + 1<101>

 Q-3
 KAB_B: 1<101>
 K_ALT: 1<0>

 Q-4
 KAB_B: 1<0>


 End Q-Markings : needattackshort
```

- Keine parallelen Protokolläufe
- Angreifer wiederholt Nachrichten beliebig

```
New Initial Marking:


 ABS: 1<A,B,S>
 B: 1<B>
 COUNT: 1<1>
 FC: 1<0>
 KAB: 1<101>
 KAB_A: 1<0>
 KAB_B: 1<0>
 KAS: 1<111>
 KBS: 1<222>
 NEU: 1<1>
 N_A: 1<100>
 N_B: 1<100>
 REP_PS: 1<B>
 RUNS: 2<1>
 STOP: 1<1>
 ZAEHL: 1<1>


Analysis
Start: 5.12.1996 8:46:43
Stop:  5.12.1996 10:12:32

 Reachability Graph needattackmid

 34993 Markings computed.
 (9468 Dead Markings)
 (0 Pseudo Occurrence Steps)
 (58004 Occurrence Steps)

Minimal Automaton
```

```
Start: 5.12.1996 10:35:39
Stop:  5.12.1996 10:58:59

 Minimal Automaton needattackmid

 9 States computed.
 (14284 critical Markings)
 (10 Edges, 10 Labels)

 Q-Markings : needattackmid

 Q-1
 KAB_B: 1<101>
 K_ALT: 1<0> + 1<101> + 1<301>

 Q-2
 KAB_B: 1<301>
 K_ALT: 1<0> + 1<101>

 Q-3
 KAB_B: 1<101>
 K_ALT: 1<0> + 1<101> + 1<401>

 Q-4
 KAB_B: 1<401>
 K_ALT: 1<0> + 1<101>

 Q-5
 KAB_B: 1<101>
 K_ALT: 1<0> + 1<101> + 1<201>

 Q-6
 KAB_B: 1<201>
 K_ALT: 1<0> + 1<101>

 Q-7
 KAB_B: 1<101>
 K_ALT: 1<0>

 Q-8
 KAB_B: 1<0>


 End Q-Markings : needattackmid
```

B Implementation der Tableau-Methode

B.1 Regeln

Protokollunabhängige Regeln		Nr
Nonce-Verification	$\frac{P \models Q \models X}{P \models \sharp(X) \wedge P \models Q \mid\sim X}$	1
Freshness	$\frac{P \models \sharp(X.Y)}{P \models \sharp(X)}$	6
	$\frac{P \models Q \models \sharp(X.Y)}{P \models Q \models \sharp(X)}$	7
Belief	$\frac{P \models (X.Y)}{P \models X \wedge P \models Y}$	8
	$\frac{P \models Q \models (X.Y)}{P \models Q \models X \wedge P \models Y}$	9
Jurisdiction	$\frac{P \models X}{P \models Q \Longrightarrow X, P \models Q \models X}$	3

Protokollabhängige Regeln		Nr.
Message-Meaning	$\frac{P \models Q \mid\sim X}{(P \models Q \overset{K}{\longleftrightarrow} P) \wedge (P \triangleleft \{X; K\})}$	2
Belief	$\frac{P \models X}{P \models (X.Y)}$	4
	$\frac{P \models Q \models X)}{P \models Q \models (X.Y)}$	5

B.2 Vorspann und Netze

Vorspann

```
/*--------------------------------------*/
/* Mengendefinitionen    */
/*--------------------------------------*/

/* grundlegende Datentypen  */

defset F = {F} ;
defset ZERO = {0} ;

defset PRINCIPALS = { A,B,C,S,Z,P,Q } ;  /* je nach Anzahl der Teilnehmer erweitern !!! */
defset FRESH = { N,T } ;  /* N=Nonce T=Timestamp */
defset NONCES = pro(FRESH,PRINCIPALS) ;  /* PRINCIPAL = Erzeuger der Nonce N  */


defset KEYNAME = { K,KAB,KABnew,KAS,KBS,KASEC,KAPUB,KBSEC,KBPUB,KSEC,KPUB } || NONCES ;
 /* je nach Bedarf erweitern , kein Informationsgehalt !!!  */

defset KEYTYPE = { SYM,PUB,SEC } ;
defset SAFEKEY = pro (KEYNAME,KEYTYPE,PRINCIPALS,PRINCIPALS) ;
/* Bedeutung: KEYNAME ist sicherer Schluessel */

defset KEYS = SAFEKEY || ZERO ;

/* FORMELN,die auch in idealisierten Protokollen vorkommen koennen */

defset SYMBOLE1 = { glaubt,fresh,sees,judic,said }  ;   /**/
defset DATEN1 = nat_0 || PRINCIPALS || NONCES || KEYS  ;
defset SYMBOLE3 = SYMBOLE1 || DATEN1 ;
defset FORMELN1 = seq(SYMBOLE3) ;
defset FORMELN2 = pro(FORMELN1,F)  ;
defset DATEN = DATEN1 || FORMELN2 ;

defset DATENSEQ = seq(DATEN) ;
defset CRYPT1 =  pro(DATENSEQ,KEYS) ;
defset CRYPT2 =  CRYPT1 || DATEN ;
defset CRYPT3 =  seq(CRYPT2) ;
defset CRYPT = pro(CRYPT3,KEYS) ;

/* CRYPT =        pro(seq(pro(seq(DATEN),KEYS)||DATEN),KEYS) */

defset FORMELCRYPT = CRYPT2 || CRYPT ;

defset SYMBOLE2 = SYMBOLE1 || FORMELCRYPT  ;
defset SYMBOLE = SYMBOLE2 || DATEN ;
defset FORMELN = seq(SYMBOLE) ;
defset PROTOKOLL1 = pro (PRINCIPALS,CRYPT) ;
defset PROTFORM = pro(FORMELN,CRYPT) ;

defob PROTOKOLL = pro (PRINCIPALS,CRYPT) ;

/*----------------------------------*/
/*  Funktionen                     */
/*----------------------------------*/

defcase LASTd : DATENSEQ >> DATENSEQ
        LASTd(Z) = seg(Z,1(Z),1(Z))
                  else :: ;
```

```
defcase RESTd : DATENSEQ >> DATENSEQ
        RESTd(Z) = seg(Z,1,l(Z))
                  else :: ;

defcase LASTc : CRYPT3 >> CRYPT3
        LASTc(Z) = seg(Z,l(Z),l(Z))
                  else :: ;

defcase RESTc : CRYPT3 >> CRYPT3
        RESTc(Z) = seg(Z,1,l(Z))
                  else :: ;

defcase UND : pro(nat_0,nat_0) >> nat_0
        UND(Z,Y) = if (Z=1)&(Y=1) then 1
                                  else 0 ;

defcase ODER : pro(nat_0,nat_0) >> nat_0
ODER(Z,Y) = if (Z=1)|(Y=1) then 1
   else 0 ;

defcase COMPARE : pro(CRYPT2,DATEN) >> nat_0
COMPARE(Z,Y) = if Z=Y then 1

        else 0 ;

defcase FORMELART : FORMELN >> FORMELN
        FORMELART(F) = if seg(F,2,2)='glaubt' then glaubt
                                              else 0 ;

defcase DATA : FORMELN >> CRYPT3
DATA(F) = if FORMELART(F)='glaubt' & (seg(F,4,4)='said' | seg(F,4,4)='glaubt')
 then seg(F,5,l(F)) ,
  if FORMELART(F)='glaubt' & seg(F,3,l(F))?CRYPT3 then seg(F,3,l(F))
  else 0 ;

/*-----------------------------------------------------------*/
/* Funktion KEYUSER (kann P mit KEY k verschluesselt haben)*/
/*-----------------------------------------------------------*/

defcase KEYUSER : pro(PRINCIPALS,KEYS) >> nat_0
KEYUSER(P,K) =  if K=0   then 1 ,
if p(2,K)=PUB   then 1 ,
if p(2,K)=SEC & p(3,K)=p(4,K) & p(3,K)=P then 1 ,
if (p(2,K)=SYM|p(2,K)=SEC) & (P=p(3,K)|P=p(4,K))      then 1
         else 0 ;

/*-----------------------------------------------------------*/
/* Funktion CONT (ist Z in einer Message M  enhalten ?)    */
/* Rueckgabe von 1 oder 0 (nat_0)                           */
/*-----------------------------------------------------------*/


defrecs CONT8 : pro(CRYPT3,DATEN) >> nat_0
        CONT8(::,Y) = 0,
        CONT8(M.N,Y) = ODER(CONT8(M,Y),COMPARE(N,Y)) ;

defrecs CONT7 : pro(DATENSEQ,DATEN) >> nat_0
        CONT7(::,Y) = 0,
        CONT7(N.K,Y) = ODER(CONT7(N,Y),COMPARE(K,Y)) ;

defrecs CONT6 : pro(DATENSEQ,CRYPT3,CRYPT2) >> nat_0
```

```
        CONT6(::,M,N) = 1,
        CONT6(Z.Y,M,N) = UND(CONT6(Z,M,N),CONT8(M.N,Y)) ;

defrecs CONT5 : pro(DATENSEQ,DATENSEQ) >> nat_0
        CONT5(::,N) = 1,
        CONT5(Z.Y,N) = UND(CONT5(Z,N),CONT7(N,Y)) ;

defcase CONT4 : pro(CRYPT3,CRYPT2,DATENSEQ) >> nat_0
        CONT4(M,N,Z) = if N ? CRYPT then CONT5(Z,p(1,N)) ,
                       if N ? DATEN then CONT6(Z,M,N)
                                    else 0 ;

defrecs CONT3 : pro(CRYPT3,DATENSEQ) >> nat_0
CONT3(::,Z) = 0,
CONT3(M.N,Z) = ODER(CONT3(M,Z),CONT4(M,N,Z)) ;

defcase CONT : pro(CRYPT,DATENSEQ) >> nat_0
CONT(M,Z) = if l(p(1,M)) >= l(Z)  then  CONT3(p(1,M),Z)
               else 0 ;

/*----------- Ende der Funktion CONT ---------------*/


/*----------- Funktion KEYOF-----------------------*/
/*-- Gibt den Key an, mit dem ein DATENSEQ in ------*/
/*-- Z verschluesselt ist --------------------------*/
/*------------------------------------------------*/

defcase KEYOF5 : pro(CRYPT3,DATENSEQ) >> KEYS
KEYOF5(N,Z) = if (seg(N,1,1) ? CRYPT) & (CONT5(p(1,seg(N,1,1)),Z)=1) then p(2,seg(N,1,1))
      else 0 ;

defcase KEYOF4 :pro(CRYPT2,DATENSEQ) >> nat_0
KEYOF4(N,Z) = if N ? CRYPT then CONT5(p(1,N),Z)
else 0 ;

defrecs KEYOF3 : pro(CRYPT3,DATENSEQ) >> nat_0
KEYOF3(::,Z) = 0,
KEYOF3(M.N,Z) = ODER(KEYOF3(M,Z),KEYOF4(N,Z)) ;

defcase KEYOF2 : pro(CRYPT3,CRYPT2,DATENSEQ) >> nat_0
        KEYOF2(M,N,Z) = if N ? DATEN then CONT6(Z,M,N)
                                    else 0 ;

defrecs KEYOF1 : pro(CRYPT3,DATENSEQ) >> nat_0
KEYOF1(::,Z) = 0,
KEYOF1(M.N,Z) = ODER(KEYOF1(M,Z),KEYOF2(M,N,Z)) ;

defcase KEYOF : pro (DATENSEQ,CRYPT) >> CRYPT1
KEYOF(Z,M) = if KEYOF1(p(1,M),Z)=1|p(2,p(2,p(1,M)))='SEC' then (Z,p(2,M)) ,
                     if KEYOF3(p(1,M),Z)=1 then (Z,KEYOF5(p(1,M),Z))
     else (0,0) ;

/*----------- Funktion PARTOF----------------------*/
/*-- Gibt den Teil einer Nachricht M an,----------*/
/*-- in dem ein Datenseq Z enthalten ist,---------*/
/*-- und mit denm es gemeinsam verschluesselt ist.--*/
/*------------------------------------------------*/

defcase AUSSEN1 : CRYPT2 >> DATENSEQ
        AUSSEN1(N) = if N?DATENSEQ then N
                     else :: ;
```

```
defrecs AUSSEN : CRYPT3 >> DATENSEQ
        AUSSEN(::)  = ::,
        AUSSEN(M.N) = AUSSEN(M).AUSSEN1(N) ;



defcase AUSSENTEIL1 : pro(CRYPT3,CRYPT2,DATENSEQ) >> nat_0
        AUSSENTEIL1(M,N,Z) = if N ? CRYPT then 0 ,
                        if N ? DATEN then CONT6(Z,M,N)
                                     else 0 ;


defrecs AUSSENTEIL : pro(CRYPT3,DATENSEQ) >> nat_0
AUSSENTEIL(::,Z) = 0,
AUSSENTEIL(M.N,Z) = ODER(AUSSENTEIL(M,Z),AUSSENTEIL1(M,N,Z)) ;


defcase INNENTEIL1 : pro(CRYPT2,DATENSEQ) >> nat_0
INNENTEIL1(N,Z) = if N ? CRYPT then CONT5(Z,p(1,N))
else 0 ;

defrecs INNENTEIL  : pro(CRYPT3,DATENSEQ) >> nat_0
INNENTEIL(::,Z) = 0,
INNENTEIL(M.N,Z) = ODER(INNENTEIL(M,Z),INNENTEIL1(N,Z)) ;

defcase PARTOF : pro (DATENSEQ,CRYPT) >> FORMELCRYPT
PARTOF(Z,M) = if ((AUSSENTEIL(p(1,M),Z)=1)&(l(AUSSEN(p(1,M)))>l(Z)))
then (AUSSEN(p(1,M)),p(2,M)) ,
      if ((INNENTEIL(p(1,M),Z)=1)) then M
     else (0,0) ;

/* Funktionen zum Vergleich der Formeln mit Protokollnachrichten  */


defcase WIEOFT2 : pro(CRYPT2,DATENSEQ) >> nat_0
WIEOFT2(N,Z) = if N?CRYPT then CONT(N,Z)
else 0 ;


defrecs WIEOFT1 : pro(CRYPT3,DATENSEQ) >> nat_0
WIEOFT1(::,Z) = 0,
WIEOFT1(M.N,Z) = WIEOFT1(M,Z) + WIEOFT2(N,Z) ;

defcase WIEOFT : pro(CRYPT3,DATENSEQ) >> nat_0
WIEOFT(M,Z) = AUSSENTEIL(M,Z) + WIEOFT1(M,Z) else 0;


defcase ENTHAELT2 : pro(CRYPT2,DATENSEQ) >> CRYPT3
ENTHAELT2(N,Z) = if (N?CRYPT1)&(CONT(N,Z)=1) then N
else :: ;

defrecs ENTHAELT : pro(CRYPT3,DATENSEQ) >> CRYPT3
ENTHAELT(::,Z) = :: ,
ENTHAELT(M.N,Z) = ENTHAELT(M,Z).ENTHAELT2(N,Z) ;



defcase PART1 : pro(CRYPT,DATENSEQ) >> CRYPT
PART1(M,Z) =  if AUSSENTEIL(p(1,M),Z)=1 then (AUSSEN(p(1,M)),p(2,M))
      else (seg(ENTHAELT(p(1,M),Z),1,1),p(2,M))   ;
```

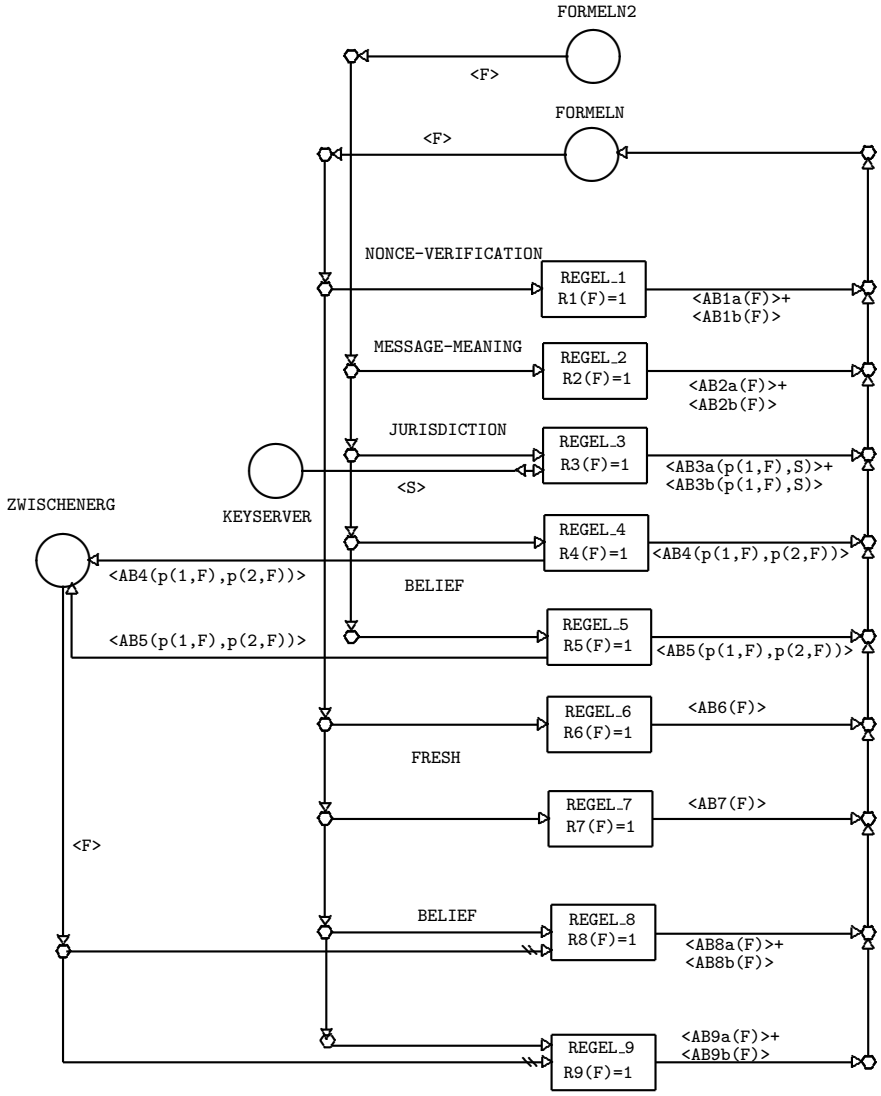
```
defcase PART2 : pro(CRYPT,DATENSEQ) >> CRYPT
PART2(M,Z) = (seg(ENTHAELT(p(1,M),Z),2,2),p(2,M)) else (::,0) ;

defcase PART3 : pro(CRYPT,DATENSEQ) >> CRYPT
PART3(M,Z) = (seg(ENTHAELT(p(1,M),Z),3,3),p(2,M)) else (::,0) ;

defcase PART4 : pro(CRYPT,DATENSEQ) >> CRYPT
PART4(M,Z) = (seg(ENTHAELT(p(1,M),Z),4,4),p(2,M)) else (::,0) ;

defcase PART5 : pro(CRYPT,DATENSEQ) >> CRYPT
PART5(M,Z) = (seg(ENTHAELT(p(1,M),Z),5,5),p(2,M)) else (::,0) ;


/* Inschrift fuer die Transition CONT   */
/* CONT soll nur F,M verarbeiten, die   */
/* Regel2 und 3 aktivieren              */



defcase C :  pro(FORMELN,PRINCIPALS,CRYPT) >> nat_0
C(F,Q,M) = if (FORMELART(F)='glaubt')&
             (  (  (seg(F,4,4)='said')
                 & (seg(F,5,l(F))?DATENSEQ)
                        & (CONT(M,seg(F,5,l(F)))=1)
         & (seg(F,1,1)=Q)
         & (KEYUSER(seg(F,3,3),p(2,KEYOF(seg(F,5,l(F)),M )))=1))
                 | (  (seg(F,3,l(F))?DATENSEQ)
         & (CONT(M,seg(F,3,l(F)))=1)
         & (seg(F,1,1)=Q))
      | (  (FORMELART(seg(F,3,l(F)))='glaubt')
 & (seg(F,5,l(F))?CRYPT2)
 & (CONT(M,seg(F,5,l(F)))=1)
   & (seg(F,1,1)=Q)
 & (PARTOF(seg(F,5,l(F)),M)~=(0,0))
 &(KEYUSER(seg(F,3,3),p(2,M))=1)))

     then 1
else 0;



/* Regel 1:     Nonce-Verification      */


defcase R1 : FORMELN  >> nat_0
        R1(F) = if (FORMELART(F)='glaubt')&(seg(F,4,4)='glaubt')&(seg(F,5,l(F))?DATENSEQ) then 1
  else 0 ;


defterm AB1a : FORMELN >> FORMELN
AB1a(F) = seg(F,1,2).'fresh'.seg(F,5,l(F)) ;

defterm AB1b : FORMELN >> FORMELN
AB1b(F) = seg(F,1,3).'said'.seg(F,5,l(F)) ;



/* Regel 2:     Message-Meaning-*/

defcase R2 : PROTFORM >> nat_0
```

```
R2(F) = if (seg(p(1,F),4,4)='said')          then 1
               else 0 ;


defcase AB2a : PROTFORM >> FORMELN
AB2a(F) = seg(p(1,F),1,1).'sees'.p(2,F) else 0;

defcase AB2b : PROTFORM >> FORMELN
AB2b(F) = seg(p(1,F),1,1).'glaubt'.p(2,p(2,F)) else 0;




/* Regel 3:  Jurisdiction  */

defcase R3 : PROTFORM >> nat_0
R3(F) = if (seg(p(1,F),3,l(p(1,F)))?DATENSEQ)    then 1
    else 0 ;


defcase AB3a : pro(FORMELN,PRINCIPALS) >> FORMELN
AB3a(F,S) = seg(F,1,2).S.judic.seg(F,3,l(F)) else 0 ;

defcase AB3b : pro(FORMELN,PRINCIPALS) >> FORMELN
AB3b(F,S) = seg(F,1,2).S.glaubt.seg(F,3,l(F)) else 0 ;


/* Regel 4: P.glaubt.Z.Y => P.glaubt.Z     */

defcase R4 :  PROTFORM >> nat_0
R4(F) = if (FORMELART(p(1,F))='glaubt')&(seg(p(1,F),3,l(p(1,F)))?CRYPT2)
&(CONT(p(2,F),seg(p(1,F),3,l(p(1,F))))=1)&
               (PARTOF(seg(p(1,F),3,l(p(1,F))),p(2,F))~=(0,0))    then 1
               else 0  ;

defcase AB4 : pro(FORMELN,CRYPT) >> FORMELN
        AB4(F,M) = if p(1,PARTOF(seg(F,3,l(F)),M))?CRYPT
then seg(F,1,2).p(1,p(1,PARTOF(seg(F,3,l(F)),M))) ,
   if p(1,PARTOF(seg(F,3,l(F)),M))?CRYPT3
then seg(F,1,2).p(1,PARTOF(seg(F,3,l(F)),M))
   else 0 ;



/* Regel 5: P.glaubt.Q.glaubt.Z.Y => P.glaubt.Q.glaubt.Z */


defcase R5 :  PROTFORM >> nat_0
R5(F) = if (FORMELART(seg(p(1,F),3,l(p(1,F))))='glaubt')&(seg(p(1,F),5,l(p(1,F)))?CRYPT2)&
   (CONT(p(2,F),seg(p(1,F),5,l(p(1,F))))=1)&
   (PARTOF(seg(p(1,F),5,l(p(1,F))),p(2,F))~=(0,0))&
   (KEYUSER(seg(p(1,F),3,3),p(2,p(2,F)))=1)           then 1
               else 0  ;


defcase AB5 : pro(FORMELN,CRYPT) >> FORMELN
        AB5(F,M) = if p(1,PARTOF(seg(F,5,l(F)),M))?CRYPT
then seg(F,1,4).p(1,p(1,PARTOF(seg(F,5,l(F)),M))) ,
   if p(1,PARTOF(seg(F,5,l(F)),M))?CRYPT3
then seg(F,1,4).p(1,PARTOF(seg(F,5,l(F)),M))
   else 0 ;
```

```
/* Regel 6: P.glaubt.fresh.Z => P.glaubt.fresh.Z.Y    */

defcase R6 : FORMELN >> nat_0
        R6(F) = if (FORMELART(F)='glaubt')&(seg(F,3,3)='fresh')&(l(F)>4) then 1
                                                                  else 0 ;

defcase AB6 : FORMELN >> FORMELN
        AB6(F) = seg(F,1,4) else 0 ;

/* Regel 7: P.glaubt.fresh.Y => P.glaubt.fresh.Z.Y    */

defcase R7 : FORMELN >> nat_0
        R7(F) = if (FORMELART(F)='glaubt')&(seg(F,3,3)='fresh')&(l(F)>4) then 1
                                                                  else 0 ;

defcase AB7 : FORMELN >> FORMELN
        AB7(F) = seg(F,1,3).seg(F,5,l(F)) else 0 ;


/* Regel 8: P.glaubt.Z & P.glaubt.Y +> P.glaubt.Z.Y   */

defcase R8 : FORMELN >> nat_0
R8(F) = if (FORMELART(F)='glaubt')&(l(F)>3)&seg(F,3,l(F))?CRYPT3 then 1
            else 0 ;

defcase AB8a : FORMELN >> FORMELN
AB8a(F) = seg(F,1,3) else 0 ;

defcase AB8b : FORMELN >> FORMELN
AB8b(F) = seg(F,1,2).seg(F,4,l(F)) else 0  ;
```

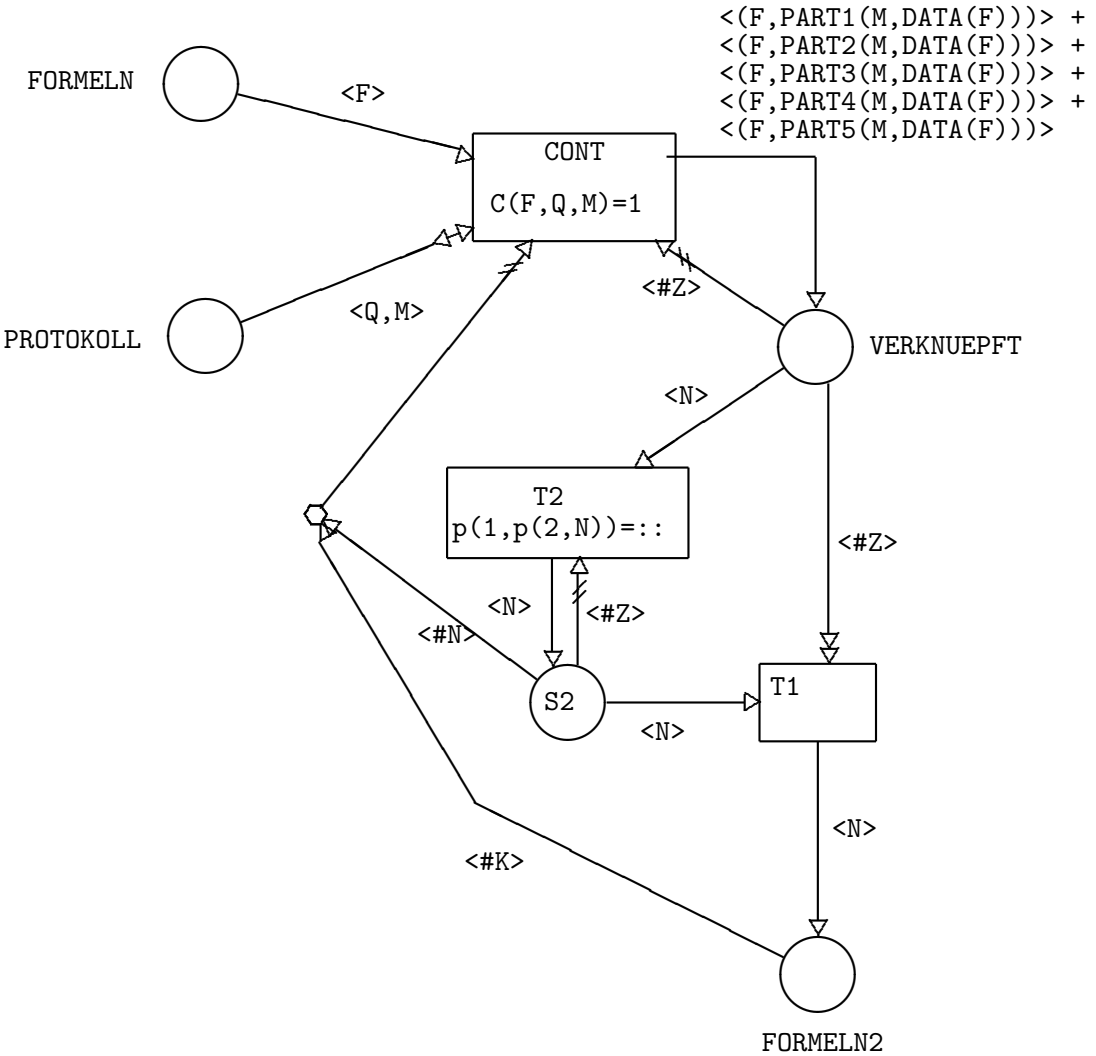
Netz: tableau.lisp

Stellenname	Definitionsbereich
FORMELN	<FORMELN>
FORMELN2	<PROTFORM>
KEYSERVER	<PRINCIPALS>
ZWISCHENERG	<FORMELN>



Netz: auswahl.lisp

Stellenname	Definitionsbereich
FORMELN	<FORMELN>
FORMELN2	<PROTFORM>
PROTOKOLL	PROTOKOLL
S2	<PROTFORM>
VERKNUEPFT	<PROTFORM>



B.3 Anfangsmarkierungen

```
\* Needham-Schroeder Protokoll *\

 FORMELN:   1<A.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<S>,
 PROTOKOLL: 1<A,((N,A).B.(KAB,SYM,A,B).
                ((KAB,SYM,A,B).A,(KBS,SYM,B,S)),(KAS,SYM,A,S))> +
    1<A,((N,B),(KAB,SYM,A,B))> +
    1<B,((KAB,SYM,A,B).A,(KBS,SYM,B,S))> +
    1<B,((N,B),(KAB,SYM,A,B))> + 1<S,(A.B.(N,A),0)>;


\* CCITTX509 Protokoll *\

 FORMELN:   1<A.GLAUBT.B.GLAUBT.201>,
 KEYSERVER: 1<S>,
 PROTOKOLL: 1<A,((T,B).(N,B).(N,A).201.(200,(KPUB,PUB,A,A)),(KBSEC,SEC,B,B))> +
            1<B,((N,B),(KASEC,SEC,A,A))> +
            1<B,((T,A).(N,A).100.(101,(KBPUB,PUB,B,B)),(KASEC,SEC,A,A))>;


\* AndrewRPC Handshake *\

 FORMELN:   1<A.GLAUBT.(KABNEW,SYM,A,B)>,
 KEYSERVER: 1<B>,
 PROTOKOLL: 1<A,((N,B).(KABNEW,SYM,A,B),(KAB,SYM,A,B))> +
            1<A,((N,B).(N,A),(KAB,SYM,A,B))> + 1<B,((N,A),(KAB,SYM,A,B))> +
            1<B,((N,B),(KAB,SYM,A,B))>;


\* Widemouthed Frog    *\

 FORMELN:   1<B.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<S> + 1<A>,
 PROTOKOLL: 1<B,((T,S).(KAB,SYM,A,B).(A.GLAUBT.(KAB,SYM,A,B),F),(KBS,SYM,B,S))> +
            1<S,((T,A).(KAB,SYM,A,B),(KAS,SYM,A,S))>;

\* Kerberos            *\

 FORMELN:   1<A.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<S>,
 PROTOKOLL: <A,((T,S).(KAB,SYM,A,B).((T,S).(KAB,SYM,A,B),
               (KBS,SYM,B,S)),(KAS,SYM,A,S))> +
    <B,((T,S).(KAB,SYM,A,B),(KBS,SYM,B,S))> +
    <B,((T,A).(KAB,SYM,A,B),(KAB,SYM,A,B))> +
    <A,((T,A).(KAB,SYM,A,B),(KAB,SYM,A,B))> ;
```

```
\* Otway-Rees Protokoll   *\

 FORMELN:   1<A.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<S>,
 PROTOKOLL: 1<A,((N,A).(KAB,SYM,A,B).(B.SAID.(N,C),F),(KAS,SYM,A,S))> +
            1<B,(((N,A).(KAB,SYM,A,B).(B.SAID.(N,C),F),(KAS,SYM,A,S)).
                ((N,B).(KAB,SYM,A,B).(A.SAID.(N,C),F),(KBS,SYM,B,S)),0)> +
            1<B,((N,A).(N,C).A.B,(KAS,SYM,A,S))> +
            1<S,(((N,A).(N,C).A.B,(KAS,SYM,A,S)).
                ((N,B).(N,C).A.B,(KBS,SYM,B,S)),0)>;


\* Yahalom Protokoll   *\

 FORMELN:   1<B.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<A> + 1<S>,
 PROTOKOLL: 1<A,((KAB,SYM,A,B).(FRESH.(KAB,SYM,A,B),F).(N,A).(N,B).
                (B.SAID.(N,A),F),(KAS,SYM,A,S))> +
            1<B,(((N,B).(S.GLAUBT.FRESH.(KAB,SYM,A,B),F),
                ((N,B),SEC,A,B)),(KAB,SYM,A,B))> +
            1<B,((KAB,SYM,A,B),(KBS,SYM,B,S))> +
            1<S,((N,A).(N,B),(KBS,SYM,B,S))>;
```

B.4 Analyse am Beispiel des Needham-Schroeder Protokolls

Anfangsmarkierung für $A \models A \overset{K_{AB}}{\longleftrightarrow} B$:

```
 FORMELN  : 1<A.GLAUBT.(KAB,SYM,A,B)>,
 KEYSERVER: 1<S>,
 PROTOKOLL: 1<A,((N,A).B.(KAB,SYM,A,B).((KAB,SYM,A,B).A,(KBS,SYM,B,S)),(KAS,SYM,A,S))> +
            1<A,((N,B),(KAB,SYM,A,B))> +
            1<B,((KAB,SYM,A,B).A,(KBS,SYM,B,S))> +
            1<B,((N,B),(KAB,SYM,A,B))> + 1<S,(A.B.(N,A),0)>;



Analysis
Start: 5.12.1996 9:44:12
Stop:  5.12.1996 9:44:16


 Reachability Graph tab

 68 Markings computed.
 (7 Dead Markings)
 (0 Pseudo Occurrence Steps)
 (100 Occurrence Steps)

Place:       1  2  3  4  5 6 7 8 9 >9 Sum   (Frequency)
FORMELN      5 36 11 11  - - - - -  -  63
```

```
FORMELN2    14  -  -  -  - - - - - -  -  14
KEYSERVER   69  -  -  -  - - - - - -  -  69
PROTOKOLL   **  -  -  - 69 - - - - -  -  69
S2          14  -  -  -  - - - - - -  -  14
VERKNUEPFT  ** 14 14  -  - - - - - -  -  28
ZWISCHENERG 56  -  -  -  - - - - - -  -  56
```

Inhalt der Totmarkierungen (wichtig ist nur die Stelle FORMELN):

M-69:
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.B>
+ 1<A.GLAUBT.S.JUDIC.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

M-67
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.B>
+ 1<A.GLAUBT.S.JUDIC.(N,A).B.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

M-66:
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.(KAB,SYM,A,B)>
+ 1<A.GLAUBT.S.JUDIC.(N,A).B.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

M-62:
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.(N,A)>
+ 1<A.GLAUBT.S.JUDIC.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

M-58:
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.(N,A)>
+ 1<A.GLAUBT.S.JUDIC.(N,A).B.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

M-21:
FORMELN: 1<A.GLAUBT.(KAS,SYM,A,S)> + 1<A.GLAUBT.FRESH.(KAB,SYM,A,B)>
+ 1<A.GLAUBT.S.JUDIC.(KAB,SYM,A,B)>
+ 1<A.SEES.((N,A).B.(KAB,SYM,A,B),(KAS,SYM,A,S))>

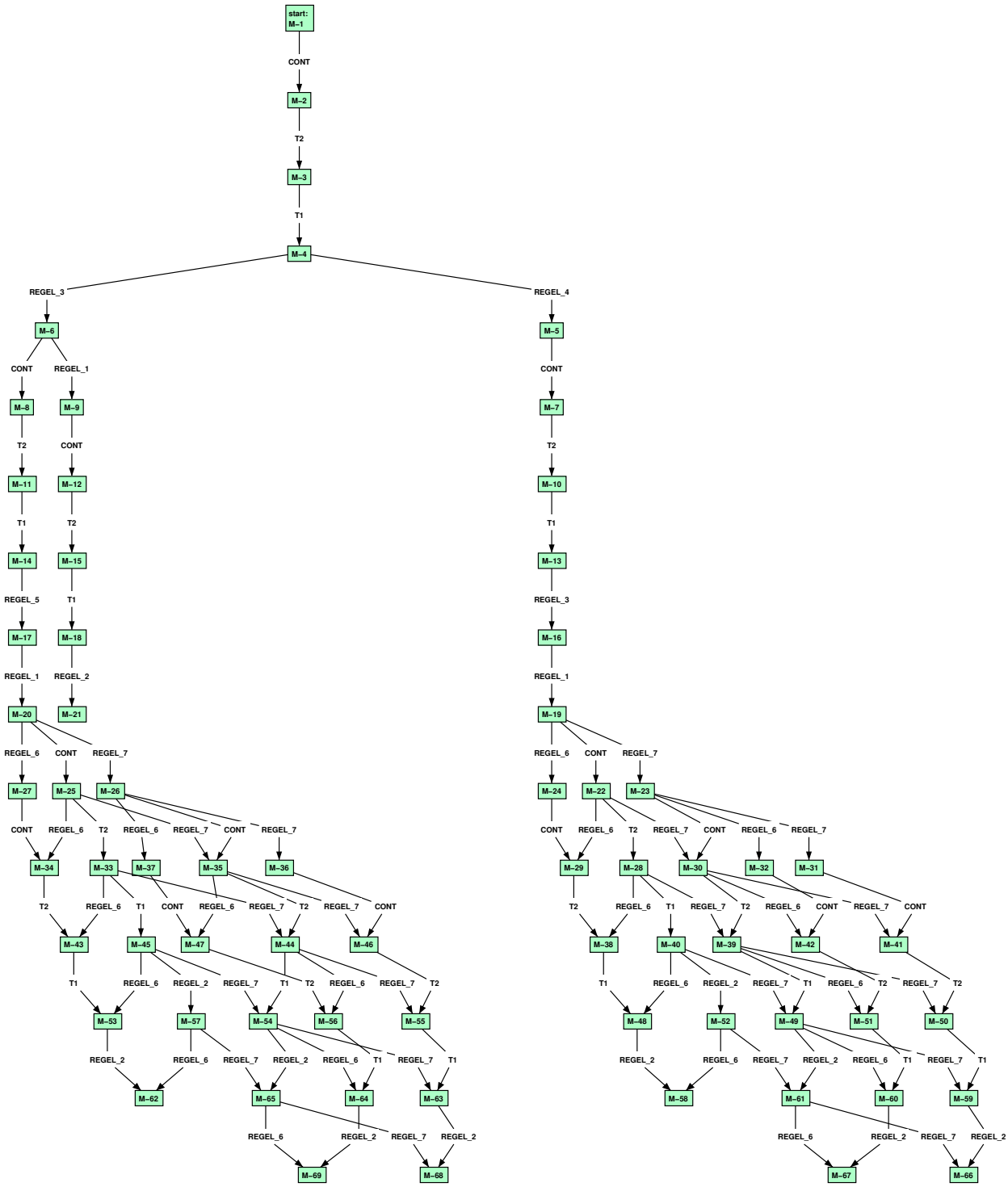


Abbildung 17: Erreichbarkeitsgraph Needham-Schröder Protokoll $A \models\!\models A \overset{K_{AB}}{\longleftrightarrow} B$

Literatur

[1] B.Alpern, F.B. Schneider. "Defining Liveness" *Information Processing Letters*, 21(4):181-185, (Okt. 1985)

[2] R. Anderson und R. Needham "Programming Satan's Computer". *LNCS 1000 426-440* Springer (1995).

[3] T. Aura "Modelling the Needham-Schröder authentication protocol with high level petri nets" *Technical Reports No.14* Helsinki University of Technology (1995)

[4] M. Burrows, M. Abadi und R. Needham "A Logic of Authentication". *Report 39 Digital Systems Research Center, Palo Alto, California* (Feb. 1989)

[5] M. Fitting "Proof methods for modal and intuitionistic logics". Reidel (1983), ISBN 90-277-1573-4

[6] L. Gong, R. Needham und R. Yahalom "Reasoning about Belief in Cryptographic Protocols" *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 234-248, IEEE Press (1990)

[7] J.E. Hopcroft, J.D. Ullmann "Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie". Addison-Wesley (1988)

[8] W. Mao, C. Boyd "Towards Formal Analysis of Security Protocols" *Proceedings of IEEE Computer Security Foundations Workshop VI*, pages 147-158, IEEE Press (1993).

[9] Peter Ochsenschläger, Rainer Prinoth "Modellierung verteilter Systeme". Vieweg (1995), ISBN 3-528-05433-6

[10] C. Petri "Kommunikation mit Automaten". Dissertation Universität Bonn (1962)

[11] P. Starke "Petrinetze". VEB Deutscher Verlag der Wissenschaften, Berlin DDR (1981)

[12] H. Tzonelin , L. Narn–Yih , L. Chuan-Ming , Ming-Yung Ko, Yung-Hsiang Chen "Two attacks on Neumann-Stubblebine authentication protocols" *Information Processing Letters* 53 (1995) 103-107.